Τμήμα Πληροφορικής
Πανεπιστήμιο Δυτικής Μακεδονίας

Δημιουργία ευρετηρίων με κατακερματισμό και ισορροπημένα δέντρα, χρησιμοποιώντας πρότυπες κλάσεις στη C++.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

Δημήτρης Καρσλίδης

(ΑΕΜ:2659)

Επιβλέπων: Μιχάλης Δόσης

February 7, 2023

Εισαγωγή

Στο παρόν έγγραφο θα αναλυθεί μια βιβλιοθήκη η οποία θα παρέχει ποικίλες ενέργειες πάνω σε μια βάση δεδομένων. Η ενέργειες θα πρέπει να εκτελούνται σε αποδοτικό χρόνο ενώ παράλληλα δεν θα πρέπει να υπάρχει απώλεια μνήμης.

Περιεχόμενα

Κεφάλαιο 1

Γιατί C++?

1.1 Με μια γρήγορη ματιά

Η C++ ως γλώσσα προγραμματισμού είναι δομημένη στα θεμέλια της C. Συγκεκριμένα χαρακτηρίζεται και η αναβαθμισμένη έκδοσή της λόγω των βιβλιοθηκών που έχουν προστεθεί. Η ανάγκη για αυτήν την εξέλιξη οφείλεται στην σταδιακή αύξησης της πολυπλοκότητας των προγραμμάτων-λογισμικών η οποία επηρεαζόταν άμεσα από την πρόοδο της τεχνολογίας των υπολογιστών. Μια άλλη πιο εξειδικευμένη προγραμματιστικά προσέγγιση είναι ότι η μέθοδος του δομημένου προγραμματισμού είναι κατάλληλη για προγράμματα μικρού μεγέθους ενώ η μέθοδος αντικειμενοστραφής προγραμματισμού είναι δραματικά πιο πρακτική για προγράμματα μεγάλου μεγέθους. Σημαντικό είναι να σημειωθεί ότι η C όπως και η C++ είναι αντικειμενοστραφής γλώσσες με μια μεγάλη διαφορά ότι η C περιορίστηκε στην εξέλιξη της αντικειμενοστραφής ενώ η C++ στοχεύει στην αόριστη εξέλιξή της. Άρα μπορεί να τυπωθεί ότι η C++ είναι η αντικειμενοστραφής έκδοση της C [1].

1.2 Ιδιότητες - Ιδιαιτερότητες

Όσον αφορά τις βιβλιοθήκες, αυτές αποτελούν τον συνδιασμό των καλύτερων ιδιοτήτων του δομημένου προγραμματισμού με χαρακτηριστηκά που βοηθούν στην οργάνωση και διαχείρηση του προγράμματος. Για να επιτεχθεί αυτό ο αντικειμενοστραφείς προγραμματιστμός στιρίζεται σε

τρεις βασικές ιδιοτητες: ενθυλακωση, πολυμορφησμος, κληρονομικοτητα.

1.2.1 Ενθυλάκωση

Η ενθυλάκωση είναι ένας προγραμματιστικός μηχανισμός που ενώνει τα δεδομένα και τις συναρτήσεις με τέτοιων τρόπο έτσι ώστε να μένουν ανεπηρέαστα από εξωτερικές παρεμβάσεις. Στην C++ υπάρχουν οι κλάσεις, οι δομές και οι ενώσεις που χρησιμοποιούν τέτοιου είδους μηχανισμούς. Αυτοί οι σύνθετοι τύποι δεδομένων συχνά χρησιμοποιούνε κάποιες από τις συναρτήσεις τους σαν διεπαφές προς τον εξωτερικό κώδικα διασφαλίζοντας με αυτόν τον τρόπο την κατάλληλη προσπέλαση των δεδομένων.

1.2.2 Πολυμορφισμός

Η φράση "μια διασύνδεση πολλές μέθοδοι" περιγράφει πλήρως την δυνατότητα αυτήν. Παρά την περιγραφή της που είναι τόσο απλή, ο προγραμματισμός γίνεται πιο ευνοϊκός. Για παράδειγμα θα έπρεπε να συνταχθεί για κάθε τύπο δεδομένου από μια συνάρτηση ενώ με την παρούσα δυνατότητα και με τις κατάλληλες γνώσεις συντάσσεται μόνο μια για πολλούς τύπους δεδομένων κάνοντας τον κώδικα να φαίνεται πιο συνοπτικός και απόλυτος. Αυτές οι συναρτήσεις ονομάζονται πρότυπες συναρτήσεις. Πολυμορφισμός επίσης υφίσταται και σε κλάσεις, σε πρότυπες κλάσεις.

1.2.3 Κληρονομικότητα

Εν τέλει η κληρονομικότητα είναι μια διεργασία μέσω της οποίας οι σύνθετοι τύποι δεδομένων (όπως τους συγκεκριμένους που αναφέραμε στα πλαίσια τις ενθυλάκωσης) κληρονομούν όλα τα χαρακτηριστικά από άλλους τέτοιους τύπους δεδομένων. Για παράδειγμα μπορεί να γραφεί κώδικας με τέτοιων τρόπο ώστε μια κλάση Α μπορεί να κληρονομήσει χαρακτηριστικά στις κλάσεις Β,Γ,Δ κ.ο.κ.. Διαφορετικά θα έπρεπε να γραφούν τα χαρακτηριστικά της Α κλάσης ξεχωριστά για κάθε άλλη κλάση με αποτέλεσμα να έκανε τον κώδικα πιο περιεκτικό και δυσανάγνωστο.

Κεφάλαιο 2

Σκοπός και προσχέδιο

Ο σκοπός είναι η δημιουργία μιας βιβλιοθήκης η οποία θα υποστηρίζει την εκφόρτωση πολλών δεδομένων στην μνήμη με λειτουργίες αναζήτησης, εισαγωγής και κατάργησης δεδομένων σε αποδοτικούς χρόνους. Γι αυτό και επιλέχτηκε η διαμόρφωση της μνήμης να στηρίζεται σε δέντρα AVL τα οποία αποτελούν έναν πίνακα πολλών θέσεων λόγω μεγάλου όγκου δεδομένων. Τα δέντρα είναι δομές που εκτελούνται γρήγορα οι αναζητήσεις σε σχέση με τις συνδεδεμένες λίστες. Οι εισαγωγές και οι εξαγωγές στοιχείων γίνονται επίσης πιο γρήγορα από τους ταξινομημένους πινάκες. Το μοναδικό μειονέκτημα στα δέντρα είναι η διάσχιση, προσπέλαση των στοιχειών η οποία επηρεάζεται αρνητικά από τον μεγάλο αριθμό βάθους του δέντρου. Είναι δυνατών ένα δέντρο να χάσει την ιδιότητα της γρήγορης εισαγωγής και διαγραφής στοιχειών. Σε ένα υποθετικό σενάριο αν όλα τα στοιχεία που εισάγονται είναι ταξινομημένα, το τελικό αποτέλεσμα είναι το δέντρο να έχει την μορφή μιας συνδεδεμένης λίστας. Για να αποφεύγουμε τέτοιες περιπτώσεις πρέπει να προσέχουμε το δέντρο να είναι πάντα ισοζυγισμένο-ισορροπημένο. Ένα δέντρο θεωρείτε ισορροπημένο μόνο όταν το αριστερό με το δεξί υπόδεντρο έχουν μοναδιαία ή καμία διαφορά στο βάθος. Το στοιχείο κλειδί που θα καθορίζει την διαμόρφωση των κόμβων στο δέντρο είναι ένας ακέραιος θετικός αριθμός που επιστρέφεται από την συνάρτηση hash. Η συνάρτηση hash δέχεται ως όρισμα ένα δεδομένο ως συμβολοσειρά και παράγει από αυτήν μια ακολουθία αριθμών. Όσον αφορά την κατάταξη της θέσης του πίνακα, δηλαδή σε ποια θέση του πίνακα θα γίνει η εισαγωγή του δεδομένου, γίνεται η διαίρεση της ακολουθίας αυτής με τον αριθμό των θέσεων του πίνακα και το ακέραιο μέρος της διαίρεσης καθορίζει την θέση αυτή.

2.1 AVL δέντρο. Δυαδικό δέντρο με μηχανισμούς ισορροπίας

Το AVL δέντρο πήρε το όνομα από τους ιδρυτές Adelson-Velsky και Landis. Ένα ξεχωριστό γνώρισμα που έχει είναι ότι μετά από κάθε ενέργεια εισαγωγής και κατάργησης κόμβου το δέντρο πάντα καταλήγει να είναι ισορροπημένο. Για να γίνει κάτι τέτοιο υπάρχουν επτά βοηθητικοί μέθοδοι [2].

2.2 Περιστροφές και αναδιαμόρφωση

Οι μέθοδοι αυτοί υλοποιήθηκαν εντός της κλάσης του δέντρου και αφορούν αποκλειστικά τους κόμβους και την αναδιαμόρφωσή τους. Αυτοί διατηρούν το δέντρο ισορροπημένο και είναι θεμελιώδεις για την απόδοση των ενεργειών εντός δέντρου καθώς αλλάζουν την διαμόρφωση του δέντρου χωρίς να αλλάζουν την ταξινόμηση των στοιχείων.

1. Η μέθοδος height υπολογίζει το βάθος του δοθέν δέντρου.

2. Η μέθοδος difference υπολογίζει την διαφορά του βάθους των υποδέντρων.

3. Η μέθοδος balance κάνει συγκρίσεις και καλεί την κατάλληλη συνάρτηση περιστροφής αναλόγως των περιπτώσεων.
   Στην περίπτωση που το αριστερό υπόδεντρο έχει μεγαλύτερο βάθος από το δεξί τότε:

   - Αν το αριστερό υπόδεντρο του αριστερού υποδέντρου έχει μεγαλύτερο βάθος από το δεξί υπόδεντρο τότε θα κληθεί η συνάρτηση της περιστροφής αριστερά αριστερά (left left rotate).
   - Αν το δεξί υπόδεντρο του αριστερού υποδέντρου έχει μεγαλύτερο βάθος από το αριστερό υπόδεντρο τότε θα κληθεί η συνάρτηση της περιστροφής αριστερά δεξιά (left right rotate).

   Στην περίπτωση που το δεξί υπόδεντρο έχει μεγαλύτερο βάθος από το αριστερό τότε:

   - Αν το αριστερό υπόδεντρο του δεξί υποδέντρου έχει μεγαλύτερο βάθος από το αριστερό υπόδεντρο τότε θα κληθεί η συνάρτηση της περιστροφής αριστερά δεξιά (right left rotate).

- Αν το δεξί υπόδεντρο του δεξί υποδέντρου έχει μεγαλύτερο βάθος από το αριστερό υπόδεντρο τότε θα κληθεί η συνάρτηση της περιστροφής δεξιά δεξιά (right right rotate).

4. Η μέθοδος περιστροφής αριστερά αριστερά (left left rotate) χειρίζεται την περίπτωση που το δέντρο στη ρίζα του έχει ένα κόμβο στα αριστερά και αυτό διαδοχικά έναν κόμβο στα αριστερά του.

```
1 Node* ll_rotat(Node* p)
2 {
3  Node* t;
4  t = p->Left;
5  p->Left = t->Right;
6  t->Right = p;
7  return t;
8 }
```

Σχήμα 2.1: Με την περιστροφή στον κόμβο Β το δέντρο επανέρχεται σε κατάσταση ισορροπίας.

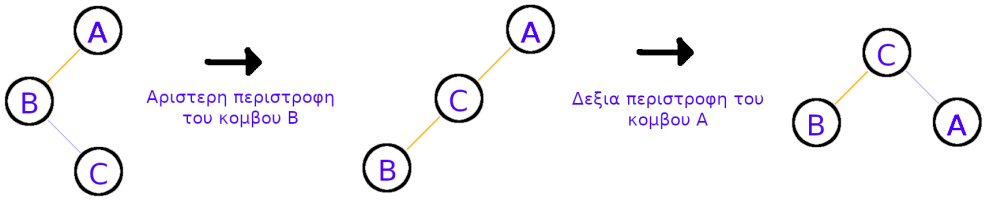
5. Η μέθοδος περιστροφής δεξιά δεξιά (right right rotate) χειρίζεται την περίπτωση που το δέντρο στη ρίζα του έχει ένα κόμβο στα δεξιά και αυτό διαδοχικά έναν κόμβο στα δέξια του.

```
1 Node* rr_rotat(Node* p)
2 {
3  Node* t;
4  t = p->Right;
5  p->Right = t->Left;
6  t->Left = p;
7  return t;
8 }
```

Σχήμα 2.2: Παρόμοιος με την προηγούμενη συνάρτηση, με την περιστροφή στον κόμβο Α το δέντρο επανέρχεται σε κατάσταση ισορροπίας.

6. Η μέθοδος περιστροφής αριστερά δεξιά (left right rotate) χειρίζεται την περίπτωση που το δέντρο στη ρίζα του έχει ένα κόμβο στα αριστερά και αυτό διαδοχικά έναν κόμβο στα δεξιά του.

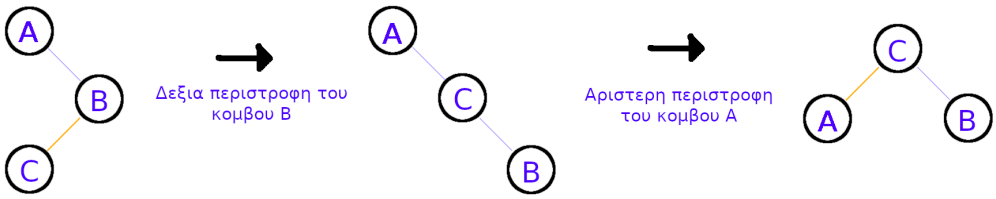
```
1 Node* lr_rotat(Node* p)
2 {
3  Node* t;
4  t = p->Left;
5  p->Left = rr_rotat(t);
6  return ll_rotat(p);
7 }
```



Σχήμα 2.3: Με την περιστροφή στον κόμβο Β το δέντρο έρχεται σε μια κατάσταση που είναι αρχική για τις δυο πρώτες συναρτήσεις. Οπότε και με άλλη μια τελική περιστροφή το δέντρο έρχεται σε ισορροπία.

7. Η μέθοδος περιστροφής δεξιά αριστερά (right left rotate) χειρίζεται την περίπτωση που το δέντρο στη ρίζα του έχει ένα κόμβο στα δεξιά και αυτό διαδοχικά έναν κόμβο στα αριστερά του.

```
1 Node* rl_rotat(Node* p)
2 {
3  Node* t;
4  t = p->Right;
5  p->Right = ll_rotat(t);
6  return rr_rotat(p);
7 }
```



Σχήμα 2.4: Αντίστοιχα με την προηγούμενη περίπτωση, το δέντρο θα έρθει πρώτα σε μια κατάσταση που είναι αρχική για τις δυο πρώτες συναρτήσεις και ύστερα το δέντρο θα έρθει σε ισορροπία.

Κεφάλαιο 3

Μέθοδοι διαχείρισης δεδομένων

Εκτός από τις παραπάνω λειτουργίες που αποτελούν κομμάτι του αρχείου AVL.h, η βιβλιοθήκη έχει επίσης και άλλα τρία αρχεία. Το αρχείο allFactStructure.h, AVLHashTable.h και interfacetxt.h.

3.1 AVL.h

Το αρχείο AVL.h αποτελείται ολοκλήρως από την κλάση AVLTree που καθορίζει την δομή του δέντρου η οποία με την σειρά της ενθυλακώνει την κλάση Node που καθορίζει την δομή του κόμβου. Επιπροσθέτως η κλάση έχει δυο μέλη, το ένα μέλος είναι ένας δείκτης Root τύπου Node και το άλλο είναι ένας ακέραιος NumNodes που θα έχει καταχωρημένο τον αριθμό των κόμβων στο δέντρο. Εκτός από τις γνωστές πλέον μεθόδους η κλάση αυτή περιέχει όλους τους απαραίτητους οικοδομητές, αποικοδομητές και υπερφορτωμένους τελεστές. Παράλληλα με τις μεθόδους εισαγωγής και κατάργησης κόμβων, η κλάση έχει πολλές φίλιες συναρτήσεις προσπέλασης κόμβων. Σχετικά με την κλάση Node, η κλάση αυτή περιέχει δυο δείκτες τύπου Node, έναν δείκτη που θα αποθηκεύει στην διεύθυνσή του το δεδομένο και έναν θετικό ακέραιο που θα συγκρατεί την ακολουθία αριθμών της συνάρτησης hash.

3.2 allFactStructure.h

Το αρχείο allFactStructure.h περιέχει τους οικοδομητές, αποικοδομητές και υπερφορτωμένους τελεστές για 131 τύπους δεδομένων. Οι τύποι

δεδομένων έχουν την μορφή τύπος-δεδομένου(στοιχείο1,στοιχείο2,στοιχείο3,...,στοιχείοN) όπου κάθε στοιχείο μπορεί να είναι εκτός από τους βασικούς τύπους δεδομένων(συμβολοσειρά, ακέραιος) να είναι και μια λίστα ή ακόμα και άλλος τύπος δεδομένου ή και μια λίστα από τύπους δεδομένων από τους 131. Όλες αυτές οι κλάσεις κληρονομούν από μια εικονική κλάση την GeneralFact τον υπερφορτωμένο τελεστή ανάθεσης και τον αποικοδομητή της. Τέλος για κάθε μια από τις κλάσεις ορίζονται δυο φίλιες συναρτήσεις η makeStringOf και matchfactsstar που ανήκουν στο αρχείο interfacetxt.h. Ακολουθεί απόσπασμα του παρών αρχείου:

```
1  class GeneralFact
2  {
3   bool fordestr = 1;
4  public:
5   virtual void operator=(GeneralFact& other) = 0;
6   virtual ~GeneralFact() = 0;
7  };
8  GeneralFact::~GeneralFact()
9  {
10  fordestr = 1;
11 }
12
13 class global_declarations :public GeneralFact
14 {
15  int    w{};
16  vector<local_object> q{};
17 public:
18  global_declarations(const vector<local_object> q1,
19  const int w1)
20   : q(q1), w(w1) {
21   q.shrink_to_fit();
22  }
23
24  global_declarations(const global_declarations& other)
25  {
26   if (this != &other)
27   {
28    this->q = other.q; this->w = other.w;
29   }
30  }
```

```
31
32  ~global_declarations()
33  {
34   this->q.clear();this->w = NULL;
35  }
36
37  void operator=(GeneralFact& other) override
38  {
39   global_declarations* ptr =
40   dynamic_cast<global_declarations*>(&other);
41   this->q = ptr->q; this->w = ptr->w;
42  }
43
44  global_declarations(global_declarations&& other)
45  noexcept
46   : q{ other.q }, w{ other.w } {}
47
48  void operator=(global_declarations&& other)  noexcept
49  {
50   std::swap(q, other.q); std::swap(w, other.w);
51  }
52
53  friend string makeStringOf(GeneralFact* obj);
54
55  friend size_t matchfactsstar(GeneralFact* Treesfact,
56  factstar* obj);
57 };
```

Η κλάση global_declarations που κληρονομεί δημόσια από την κλάση GeneralFact(γραμμή 13) έχει δυο στοιχεία(γραμμή 15,16). Το πρώτο είναι μια λίστα από τύπους δεδομένων local_object και το δεύτερο είναι ένας ακέραιος αριθμός. Η local_object είναι επίσης μια κλάση που κληρονομεί από την GeneralFact και έχει την δικιά της δομή. Ο υπερφορτωμένος τελεστής ανάθεσης(γραμμή 37) επικαλύπτει τον αντίστοιχο τελεστή της κληροδοτούμενης κλάσης(γραμμή 5). Το μέλος της κλάσης GeneralFact (γραμμή 3) χρησιμοποιείται αποκλειστικά για το σώμα του αποικοδομητή και είναι τύπου bool για να δεσμεύει όσον το δυνατών λιγότερο μνήμη που στη παρούσα περίπτωση είναι ένα byte. Μια πρόσθετη κλάση είναι η factstar. Η δομή της αποτελείτε από δυο μέλη έναν δείκτη τύπου GeneralFact και έναν ακέραιο ως params. Ο λόγος ύπαρξης αυτής της κλάσης οφείλεται στη δυνατότητα δημιουργίας ενός δεδομένου τύπου

GeneralFact και αποθήκευσης τον αριθμό των στοιχείων για σκοπούς σύγκρισης που θα αναλυθούν στο κεφάλαιο που ακολουθεί. Η κλάση GenfactError εκτυπώνει το fact που δόθηκε σε περίπτωση που είναι λανθασμένο. Δηλαδή δημιουργείται οντότητα τύπου GenfactError σε περίπτωση που δόθηκε τύπος δεδομένου με λεκτικό λάθος ή δεν υπάρχει η δομή της κλάσης αυτού του δεδομένου.

3.3 interfacetxt.h

Το αρχείο interfacetxt.h είναι το πιο περιεκτικό αρχείο σε κώδικα και έχουν οριστεί επτά σταθερές για την διευκόλυνση συγγραφής των συναρτήσεων.

```
1 const char pa   = '('; // parenthesis open
2 const char pacl = ')'; // parenthesis close
3 const char co   = ','; // coma
4 const char br   = '['; // bracket open
5 const char brcl = ']'; // bracket close
6 const char us   = '_'; // underscore
7 const char str  = '*'; // star
```

Για αρχή περιέχει δυο πολύ βασικές συναρτήσεις.

- Η συνάρτηση makeInstanceOf δέχεται ως όρισμα το δεδομένο ως συμβολοσειρά και επιστρέφει έναν δείκτη τύπου GeneralFact στον οποίο κατά συνέπεια έχει γίνει ανάθεση η διεύθυνση της αντίστοιχης κλάσης από αυτές που κληροδοτεί. Άμα θα χρειαστεί να φτιαχτεί στιγμιότυπο τύπου type_def θα εκτελεστεί το παρακάτω απόσπασμα κώδικα.

```
1  string          ALine, subl;
2  size_t          pos;
3  pos = 0;
4  ALine = inputline.substr(0,
5  inputline.find(pa, 0));
6  pos += ALine.length();
7  subl = inputline.substr(++pos,
8  inputline.length() - pos);
9
10 if (ALine == "type_def")
11 {
```

```
12    int       q, e, t, u, i, o;
13    string    w, r, y;
14
15    ALine = subl.substr(0, subl.find(co, 0));
16    q = stoi(ALine);
17    pos += ALine.length();
18    subl = inputline.substr(++pos,
19    inputline.length() - pos);
20
21    ALine = subl.substr(0, subl.find(co, 0));
22    w = ALine;
23    pos += ALine.length();
24    subl = inputline.substr(++pos,
25    inputline.length() - pos);
26
27    ALine = subl.substr(0, subl.find(co, 0));
28    e = stoi(ALine);
29    pos += ALine.length();
30    subl = inputline.substr(++pos,
31    inputline.length() - pos);
32
33    ALine = subl.substr(0, subl.find(co, 0));
34    r = ALine;
35    pos += ALine.length();
36    subl = inputline.substr(++pos,
37    inputline.length() - pos);
38
39    ALine = subl.substr(0, subl.find(co, 0));
40    t = stoi(ALine);
41    pos += ALine.length();
42    subl = inputline.substr(++pos,
43    inputline.length() - pos);
44
45    ALine = subl.substr(0, subl.find(co, 0));
46    y = ALine;
47    pos += ALine.length();
48    subl = inputline.substr(++pos,
49    inputline.length() - pos);
50
51    ALine = subl.substr(0, subl.find(co, 0));
52    u = stoi(ALine);
```

```
53   pos += ALine.length();
54   subl = inputline.substr(++pos,
55   inputline.length() - pos);
56
57   ALine = subl.substr(0, subl.find(co, 0));
58   i = stoi(ALine);
59   pos += ALine.length();
60   subl = inputline.substr(++pos,
61   inputline.length() - pos);
62
63   ALine = subl.substr(0, subl.rfind(pacl, 0));
64   ALine.resize(ALine.size() - 1);
65   o = stoi(ALine);
66   return new type_def(q, e, t, u, i, o, w, r, y);
67  }
```

Στην αρχή ξεχωρίζεται από την συμβολοσειρά ο τύπος δεδομένου(γραμμή 2). Στην συνεχεία αφού χωριστεί το στοιχείο του δεδομένου(γραμμή 15) και ενημερώνεται το μήκος της συμβολοσειράς(γραμμή 17), γίνονται οι μετατροπές τύπων όπου χρειάζεται(γραμμή 16) και κατασκευάζεται δυναμικά η οντότητα η οποία και επιστρέφεται από την συνάρτηση(γραμμή 66).

- Η συνάρτηση makeStringOf αντιθέτως δέχεται ως όρισμα έναν δείκτη τύπου GeneralFact και επιστρέφει την αντίστοιχη συμβολοσειρά. Άμα θα χρειαστεί να φτιαχτεί συμβολοσειρά από οντότητα τύπου type_def θα εκτελεστεί το παρακάτω απόσπασμα κώδικα.

```
1   size_t   prlst;
2   stringstream ss;
3   string   ALine;
4
5   if (typeid(type_def) == typeid(*obj))
6   {
7    type_def* ptr = dynamic_cast<type_def*>(obj);
8    ss << "type_def("  << ptr->q << co << ptr->w
9    << co << ptr->e << co << ptr->r << co << ptr->t
10   << co << ptr->y << co << ptr->u << co << ptr->i
11   << co << ptr->o;
12  }
13  ...
14  ss << pacl;
```

```
15  ALine = ss.str();
16  return ALine;
```

Χάρη στον μηχανισμό RTTI(Run-time type information) φανερώνεται ο τύπος δεδομένου(γραμμή 5). Στην συνέχεια γίνεται η δυναμική μετατροπή του τύπου από την βασική κλάση στην ειδική και εκχωρείται στον κατάλληλο δείκτη(γραμμή 8). Τέλος αφού εκχωρηθούν σε σωστή σειρά τα στοιχεία στη ροή συμβολοσειράς ss(γραμμή 8), αυτή με την σειρά της θα εκχωρηθεί σε μια συμβολοσειρά(γραμμή 15) η οποία θα επιστραφεί από την συνάρτηση.

- Στην συνέχεια η συνάρτηση makefactstar δέχεται ως όρισμα ένα δεδομένο ως συμβολοσειρά. Η διαφορά της με την συνάρτηση makeIsntanceOf βασίζεται στη λίστα των στοιχείων που δίνεται. Αντί να δοθούν όλα τα στοιχεία θα δοθούν τα στοιχεία που θα πρέπει να διακριθούν και για όλα τα υπόλοιπα στοιχεία θα υπάρχει το "*". Δηλαδή για παράδειγμα άμα δοθεί η μορφή τύπος-δεδομένου(στοιχείο1,στοιχείο2,*) η συνάρτηση θα επιστρέψει μια οντότητα factstar με δείκτη GeneralFact ο οποίος στην διεύθυνσή του θα έχει αποθηκεύσει τα δυο πρώτα στοιχεία και με τον ακέραιο params ο οποίος θα είναι ίσος με την τιμή δυο.

```
1   int       params = 0;
2   string      ALine, subl;
3   size_t      pos, brpos;
4   bool      flag = 1, stop = 1;
5
6   GeneralFact* ptr = nullptr;
7
8   pos = 0;
9   ALine = inputline.substr(0,
10  inputline.find(pa, 0));
11  pos += ALine.length();
12
13  if (ALine == "type_def")
14  {
15   flag = 0;
16   int     q{}, e{}, t{}, u{}, i{}, o{};
17   string    w{}, r{}, y{};
18   subl = inputline.substr(++pos,
19   inputline.length() - pos);
```

```
20
21    if ((subl.find(str, 0)) != 0)
22    {
23     params++;
24     ALine = subl.substr(0, subl.find(co, 0));
25     q = stoi(ALine);
26     pos += ALine.length();
27     subl = inputline.substr(++pos,
28     inputline.length() - pos);
29    }
30    else stop = 0;
31    if (stop)
32    {
33     if ((subl.find(str, 0)) != 0)
34     {
35      params++;
36      ALine = subl.substr(0, subl.find(co, 0));
37      w = ALine;
38      pos += ALine.length();
39      subl = inputline.substr(++pos,
40      inputline.length() - pos);
41     }
42     else stop = 0;
43    }
44    if (stop)
45    {
46     if ((subl.find(str, 0)) != 0)
47     {
48      params++;
49      ALine = subl.substr(0, subl.find(co, 0));
50      e = stoi(ALine);
51      pos += ALine.length();
52      subl = inputline.substr(++pos,
53      inputline.length() - pos);
54     }
55     else stop = 0;
56    }
57    if (stop)
58    {
59     if ((subl.find(str, 0)) != 0)
60     {
```

```
61      params++;
62      ALine = subl.substr(0, subl.find(co, 0));
63      r = ALine;
64      pos += ALine.length();
65      subl = inputline.substr(++pos,
66      inputline.length() - pos);
67     }
68     else stop = 0;
69    }
70    if (stop)
71    {
72     if ((subl.find(str, 0)) != 0)
73     {
74      params++;
75      ALine = subl.substr(0, subl.find(co, 0));
76      t = stoi(ALine);
77      pos += ALine.length();
78      subl = inputline.substr(++pos,
79      inputline.length() - pos);
80     }
81     else stop = 0;
82    }
83    if (stop)
84    {
85     if ((subl.find(str, 0)) != 0)
86     {
87      params++;
88      ALine = subl.substr(0, subl.find(co, 0));
89      y = ALine;
90      pos += ALine.length();
91      subl = inputline.substr(++pos,
92      inputline.length() - pos);
93     }
94     else stop = 0;
95    }
96    if (stop)
97    {
98     if ((subl.find(str, 0)) != 0)
99     {
100     params++;
101     ALine = subl.substr(0, subl.find(co, 0));
```

```
102      u = stoi(ALine);
103      pos += ALine.length();
104      subl = inputline.substr(++pos,
105      inputline.length() - pos);
106     }
107     else stop = 0;
108    }
109    if (stop)
110    {
111     if ((subl.find(str, 0)) != 0)
112     {
113      params++;
114      ALine = subl.substr(0, subl.find(co, 0));
115      i = stoi(ALine);
116      pos += ALine.length();
117      subl = inputline.substr(++pos,
118      inputline.length() - pos);
119     }
120     else stop = 0;
121    }
122    if (stop)
123    {
124     if ((subl.find(str, 0)) != 0)
125     {
126      params++;
127      ALine = subl.substr(0, subl.rfind(pacl, 0));
128      ALine.resize(ALine.size() - 1);
129      o = stoi(ALine);
130     }
131    }
132    ptr = new type_def(q, e, t, u, i, o, w, r, y);
133   }
134   ...
135   return factstar(ptr, params);
```

Για κάθε θέση στοιχείου, στην περίπτωση που βρεθεί "*" τότε το stop ενημερώνεται και παραλείπονται όλα τα υπόλοιπα στοιχεία ενώ σε περίπτωση που βρεθεί στοιχείο η params αυξάνει την τιμή του κατά ένα. Τέλος επιστρέφεται το factstar με τον ενημερωμένο δείκτη και η ποσότητα των στοιχειών. Ο σκοπός της παρούσας συνάρτησης αποκαλύπτεται στην ανάλυση της τελευταίας συνάρτησης αυτού του

αρχείου, της συνάρτησης matchfactsstar.

- Η συνάρτηση matchfactsstar δέχεται δυο ορίσματα το ένα είναι δείκτης τύπου GeneralFact και το άλλο δείκτης τύπου factstar.

```
1  bool    flag = 1;
2  GeneralFact* fact = obj->fact;
3  size_t   p = obj->params;
4
5  if (typeid(type_def) == typeid(*fact) &&
6  typeid(*fact) == typeid(*Treesfact))
7  {
8   flag = 0;
9   type_def* ptr = dynamic_cast<type_def*>(fact);
10  type_def* ptr2 =
11  dynamic_cast<type_def*>(Treesfact);
12  if (p)
13  {
14   switch (p)
15   {
16   case 9:
17    if (ptr->o != ptr2->o) return 0;
18   case 8:
19    if (ptr->i != ptr2->i) return 0;
20   case 7:
21    if (ptr->u != ptr2->u) return 0;
22   case 6:
23    if (ptr->y != ptr2->y) return 0;
24   case 5:
25    if (ptr->t != ptr2->t) return 0;
26   case 4:
27    if (ptr->r != ptr2->r) return 0;
28   case 3:
29    if (ptr->e != ptr2->e) return 0;
30   case 2:
31    if (ptr->w != ptr2->w) return 0;
32   case 1:
33    if (ptr->q != ptr2->q) return 0;
34   }
35  }
36 }
37 ...
```

```
38 return 1;
```

Στην εμβέλεια της συνάρτησης καταγράφεται κώδικας ο οποίος κάνει σύγκριση των δυο ορισμάτων βασισμένος στον δείκτη factstar. Ποιο συγκεκριμένα από το params κρίνεται πόσα στοιχεία είναι για σύγκριση και από το GeneralFact γίνεται η σύγκριση των στοιχείων. Σε περίπτωση που τα κατάλληλα στοιχεία είναι ανόμοια η συνάρτηση επιστρέφει 0 διαφορετικά 1.

3.4 AVLHashTable.h

Το αρχείο AVLHashTable.h αποτελείτε από την κλάση HashTable η οποία αποτελεί την διεπαφή του χρήστη με την μνήμη μέσω της οντότητας HT. Αυτή η κλάση αποτελείτε από ένα ιδιωτικό μέλος, έναν οικοδομητή και 11 λειτουργίες-ενέργειες πάνω στην διαχείριση των δεδομένων της μνήμης.
Ακολουθεί απόσπασμα της κλάσης HashTable.

```
1 std::vector<AVLTree> Table;
2 public:
3 HashTable()
4 {
5  Table.reserve(N);
6  for (int i = 0; i < N; i++)
7   Table.push_back(AVLTree());
8 }
```

Υπό το όνομα Table ορίστηκε ο πίνακας με περιεχόμενα τύπου AVLTree(γραμμή 1) και στην συνέχεια ο κατασκευαστής που δημιουργεί την οντότητα HT(μετά την γραμμή 3) με N ποσότητα δέντρων που έχει οριστεί στο προοίμιο του αρχείου.
Οι 11 λειτουργίες είναι οι εξής:

- assertz: Η λειτουργία assertz δέχεται ένα όρισμα το δεδομένο που είναι για καταχώρηση στην μνήμη ως συμβολοσειρά.

```
1 void HashTable::assertz(string Line)
2 {
3  string factInput;
4  int  Treepos;
5  size_t  fullhash;
```

```
6   fullhash = (hash<string>{}(factInput));
7   Treepos = fullhash % N;
8   Table[Treepos].Root =
9   Table[Treepos].Insert(Table[Treepos].Root,
10  makeInstanceOf(Line), fullhash);
11  Table[Treepos].Root =
12  Table[Treepos].balance(Table[Treepos].Root);
13 }
```

Στο fullhash εκχωρείτε η ακολουθία που επιστρέφει η συνάρτηση hash(γραμμή 6). Στο Treepos εκχωρείτε η θέση του πίνακα(γραμμή 7) που υπολογίζεται από το ακέραιο μέρος της διαίρεσης του fullhash με το N, όπου N ο αριθμός των θέσεων του πίνακα. Τέλος αφού υπολογιστούν τα ανωτέρω γίνεται η εισαγωγή του δεδομένου στην κατάλληλη θέση του πίνακα(γραμμή 8) και ύστερα γίνεται ο έλεγχος και οι διαδικασίες ισορροπίας(γραμμή 11).
Η χρονική πολυπλοκότητα της συνάρτησης είναι $O(\log_2 n)$ οπού n η ποσότητα φορτωμένων δεδομένων του συγκεκριμένου δέντρου, της συγκεκριμένης θέσης του πίνακα.

- findfact: Η λειτουργία findfact δέχεται ένα όρισμα το δεδομένο ως συμβολοσειρά που είναι προς αναζήτηση. Άμα βρεθεί το στοιχείο αυτό στη μνήμη τότε η συνάρτηση επιστρέφει true διαφορετικά false.

```
1  bool HashTable::findfact(string Line)
2  {
3   size_t Treepos = 0;
4   bool flag = false;
5   bool* ptrflg = &flag;
6
7   if (Line.find("*", 0) != Line.npos)
8   {
9    factstar fs;
10   fs = makefactstar(Line);
11   factstar* ptrfs = &fs;
12   for (int i = N - 1; i >= 0; i--)
13   {
14    if (Table[i].NumNodes)
15    {
16     readnodes(Table[i].Root, ptrfs, ptrflg);
17     if (flag) return true;
18    }
```

```
19    }
20   }
21   else
22   {
23    size_t Hash = (hash<string>{}(Line));
24    Treepos = Hash % N;
25    if (Table[Treepos].NumNodes)
26    {
27     readnodes(Table[Treepos].Root, Hash, ptrflg);
28     if (flag) return true;
29    }
30   }
31   return false;
32  }
```

Σε περίπτωση που βρεθεί "*"(γραμμή 7) τότε το δεδομένο δόθηκε σε μορφή factstar. Άρα θα πρέπει να εξεταστούν όλα τα δέντρα διότι η συνάρτηση hash δεν θα έδινε την κατάλληλη ακολουθία από συμβολοσειρά τύπου factstar και οπότε η μόνη επιλογή που παραμένει είναι αυτή. Επίσης η χρονική πολυπλοκότητα θα είναι $O(n)$, όπου n τα φορτωμένα δεδομένα από όλα τα δέντρα. Διαφορετικά (γραμμή 21) το δεδομένο δόθηκε σε κανονική μορφή και επομένως η αναζήτηση θα γίνει στο κατάλληλο δέντρο με χρονική πολυπλοκότητα $O(\log_2 n)$, όπου n η ποσότητα φορτωμένων δεδομένων του συγκεκριμένου δέντρου.

- retractall: Η λειτουργία retractall είναι σχετική με την κατάργηση δεδομένων από την μνήμη. Δέχεται ένα όρισμα ως συμβολοσειρά υπό τρεις περιπτώσεις.

```
1   void HashTable::retractall(string Line)
2   {
3    bool duplicate = false;
4    bool* ptrdup = &duplicate;
5    size_t Factcount = 0, FactcountTree = 0,
6    Treepos = 0;
7
8    for (int k = 0; k < N; k++)
9     FactcountTree += Table[k].NumNodes;
10
11   if (!FactcountTree)
12   {
```

```
13    return;
14   }
15   else if (Line == "(*)")
16   {
17    for (int i = N - 1; i >= 0; i--)
18    {
19     if (Table[i].NumNodes)
20     {
21      Table[i].~AVLTree();
22     }
23    }
24   }
25   else if (Line.find("*", 0) != Line.npos)
26   {
27    factstar fs;
28    fs = makefactstar(Line);
29    factstar* ptrfs = &fs;
30    for (int i = N - 1; i >= 0; i--)
31    {
32     if (Table[i].NumNodes)
33     {
34      while (!duplicate)
35      {
36       *ptrdup = true;
37       Table[i].Root =
38       Table[i].delNode(Table[i].Root, ptrfs,
39       ptrdup);
40       Table[i].Root =
41       Table[i].balance(Table[i].Root);
42      }
43      *ptrdup = false;
44     }
45    }
46   }
47   else
48   {
49    size_t Hash = (hash<string>{}(Line));
50    Treepos = Hash % N;
51    while (!duplicate)
52    {
53     *ptrdup = true;
```

```
54      Table[Treepos].Root =
55      Table[Treepos].delNode(Table[Treepos].Root,
56      Hash, ptrdup);
57      Table[Treepos].Root =
58      Table[Treepos].balance(Table[Treepos].Root);
59    }
60    *ptrdup = false;
61  }
62 }
```

Στην αρχή γίνεται ένας έλεγχος για κάθε δέντρο να μην είναι όλα χωρίς δεδομένα(γραμμή 8). Στην περίπτωση που η παράμετρος Line έχει την τιμή «(*)»(γραμμή 15) τότε θα ξεκινήσει η διαδικασία κατάργησης όλων των δεδομένων με χρονική πολυπλοκότητα $O(n)$, όπου n ο αριθμός των δεδομένων όλων των δέντρων. Στην περίπτωση που το όρισμα Line είναι factstar(γραμμή 25) τότε θα ξεκινήσει η διαδικασία διαγραφής όλων των κατάλληλων κόμβων για κάθε δέντρο με χρονική πολυπλοκότητα $O(n)$, οπού n ο αριθμός των δεδομένων όλων των δέντρων. Στην περίπτωση που το όρισμα Line είναι κανονικό δεδομένο(γραμμή 47) τότε η κατάργηση των κατάλληλων κόμβων θα υλοποιηθεί στο κατάλληλο δέντρο με χρονική πολυπλοκότητα $O(\log_2 n)$, όπου n ο αριθμός των δεδομένων του συγκεκριμένου δέντρου. Μια πολύ σημαντική σημείωση είναι ότι η παρούσα συνάρτηση δεν διαγράφει μόνο ένα δεδομένο αλλά όλα τα αντίτυπα.

- consult: Η συνάρτηση consult δέχεται ένα όρισμα ως συμβολοσειρά ένα όνομα αρχείου και φορτώνει όλα τα περιεχόμενα δεδομένα του στην μνήμη.

```
1  void HashTable::consult(string Line)
2  {
3   vector<string>   AllLines;
4   string     ALine;
5   int      Treepos{};
6   size_t     fullhash;
7   ifstream File;
8
9   File.open(Line , fstream::in);
10
11  if (File.is_open())
12  {
```

```
13    while (getline(File, ALine))
14    {
15     AllLines.push_back(ALine);
16    }
17    File.close();
18    AllLines.shrink_to_fit();
19    for (int i = 0; i < AllLines.size(); i++)
20    {
21     fullhash = (hash<string>{}(AllLines[i]));
22     Treepos = fullhash % N;
23     Table[Treepos].Root =
24     Table[Treepos].Insert(Table[Treepos].Root,
25     makeInstanceOf(AllLines[i]), fullhash);
26     Table[Treepos].Root =
27     Table[Treepos].balance(Table[Treepos].Root);
28    }
29   }
30  }
```

Εκχωρούνται όλα τα περιεχόμενα σε έναν πίνακα(γραμμή 15) και μετά παρομοίως με την assertz εντός μιας ρουτίνας(γραμμή 19) εισάγονται στην μνήμη με την σειρά όλα τα περιεχόμενα του πίνακα.

- save:Η συνάρτηση save δέχεται ένα όρισμα ως συμβολοσειρά ένα όνομα αρχείου και εξάγει όλα τα φορτωμένα δεδομένα σε ένα αρχείο με όνομα που δόθηκε.

```
1  void HashTable::save(string filename)
2  {
3   size_t   ts, Factcount = 0;
4   vector<string> AllLines;
5   vector<string>* ptr = &AllLines;
6
7   for (int i = 0; i < N; i++)
8    Factcount += Table[i].NumNodes;
9   if (Factcount)
10  {
11   fstream File(filename, ios::out | ios::in |
12   ios::trunc);
13
14   if (!File.is_open())
15    return;
```

```
16
17    for (int i = 0; i < N; i++)
18    {
19     Table[i].storeToVector(Table[i].Root, ptr);
20    }
21
22    sort(AllLines.begin(), AllLines.end());
23
24    ts = AllLines.size();
25    for (int i = 0; i < ts; i++)
26    {
27     File << AllLines[i] << endl;
28    }
29    File.close();
30   }
31  }
```

Εντός μιας ρουτίνας γίνεται προσπέλαση όλων των δέντρων(γραμμή 17) και αποθηκεύονται όλα τα δεδομένα σε έναν πίνακα. Αφού γίνει στην αρχή μια αλφαβητική ταξινόμηση στον πίνακα(γραμμή 22) εξάγονται τα δεδομένα στο αρχείο(γραμμή 25). Σε περίπτωση που το όνομα του αρχείου υπάρχει ήδη τότε το αρχείο θα χάσει όλα τα παλιά δεδομένα τα οποία θα αντικατασταθούν με τα περιεχόμενα του πίνακα. Διαφορετικά θα δημιουργηθεί ένα νέο αρχείο με τα περιεχόμενα του πίνακα.

- concat: Η συνάρτηση concat δέχεται τρεις λίστες ορίσματα. Στην τρίτη λίστα φορτώνει τα περιχυμένα στην αρχή τα δεδομένα της πρώτης λίστας ύστερα τα δεδομένα της δεύτερης λίστας και την επιστρέφει.

```
1  template<typename T>
2  list<T> HashTable::concat(list<T> list1,
3  list<T> list2, list<T> list3)
4  {
5   if (!list1.empty())
6    for (T x : list1)
7     list3.push_back(x);
8   if (!list2.empty())
9    for (T x : list2)
10    list3.push_back(x);
11  return list3;
12 }
```

Αφού γίνει έλεγχος περιεχομένων για κάθε λίστα, εκχωρούνται στην λίστα που θα επιστραφεί.

- exportToList: Η συνάρτηση exportToList επιστρέφει μια διπλή συνδεδεμένη λίστα με περιεχόμενα που είναι φορτωμένα στη μνήμη.

```
1  list<string> HashTable::exportToList()
2  {
3   list<string> result{};
4
5   for (int i = 0; i < N; i++)
6   {
7    if (Table[i].Root != nullptr)
8     readnodes(Table[i].Root, &result);
9   }
10  return result;
11 }
```

Εντός μιας ρουτίνας προσπέλασης όλων των δέντρων αποθηκεύονται τα δεδομένα στην διπλή συνδεδεμένη λίστα η οποία στο τέλος επιστρέφεται.

- exportToFList: Η συνάρτηση exportToFList επιστρέφει μια απλή συνδεδεμένη λίστα με περιεχόμενα που είναι φορτωμένα στη μνήμη

```
1  forward_list<string> HashTable::exportToFList()
2  {
3   forward_list<string> result{};
4
5   for (int i = 0; i < N; i++)
6   {
7    if (Table[i].Root != nullptr)
8     readnodes(Table[i].Root, &result);
9   }
10  return result;
11 }
```

Παρομοίως με την προηγούμενη συνάρτηση με διαφορά την διπλή με την απλή συνδεδεμένη λίστα.

- exportToString: Η συνάρτηση exportToString επιστρέφει μια συμβολοσειρά με περιεχόμενα που είναι φορτωμένα στη μνήμη χωρισμένα με κόμμα(",").

```
string HashTable::exportToString()
{
 string result{};

 for (int i = 0; i < N; i++)
 {
  if (Table[i].Root != nullptr)
   readnodes(Table[i].Root, &result);
 }
 return result;
}
```

Παρομοίως με τις δυο προηγούμενες συναρτήσεις με διαφορά ότι η παρούσα επιστρέφει τα δεδομένα ως συμβολοσειρά.

- matchedToList: Η συνάρτηση matchedToList δέχεται ως συμβολοσειρά ένα όρισμα τύπου factstar και επιστρέφει μια διπλά συνδεδεμένη λίστα με τα δεδομένα που είναι όμοια με το όρισμα.

```
list<string> HashTable::matchedToList(string Line)
{
 list<string> result{};

 factstar fc;
 fc = makefactstar(Line);
 for (int i = 0; i < N; i++)
 {
  if(Table[i].Root != nullptr)
  {
   readnodes(Table[i].Root, &fc, &result);
  }
 }
 return result;
}
```

Μέσα από μια ρουτίνα προσπέλασης των δέντρων γίνεται η διαδικασία σύγκρισης και στο τέλος επιστρέφεται η διπλή συνδεδεμένη λίστα.

- matchedToFList: Η συνάρτηση matchedToFList δέχεται ως συμβολοσειρά ένα όρισμα τύπου factstar και επιστρέφει μια απλή συνδεδεμένη λίστα με τα δεδομένα που είναι όμοια με το όρισμα.

```
forward_list<string> HashTable::matchedToFList
(string Line)
{
 forward_list<string> result{};

 factstar fc;
 fc = makefactstar(Line);
 for (int i = 0; i < N; i++)
 {
  if (Table[i].Root != nullptr)
  {
   readnodes(Table[i].Root, &fc, &result);
  }
 }
 return result;
}
```

Παρομοίως με την προηγούμενη συνάρτηση με διαφορά την διπλή με την απλή συνδεδεμένη λίστα.

Τέλος στην βάση του αρχείου AVLHashTable.h υπάρχει η πολυμορφική συνάρτηση readnodes η οποία είναι φίλια στην κλάση AVLTree. Αυτή η συνάρτηση αποτελούσε δίαυλο στις ανωτέρω συναρτήσεις προς τους κόμβους των δέντρων.
Ακολουθούν όλες οι μορφές της συνάρτησης readnodes.

```
void readnodes(AVLTree::Node* n, list<string>* ptr)
{
 if (n->Left != nullptr)
  readnodes(n->Left, ptr);
 if (n->Right != nullptr)
  readnodes(n->Right, ptr);
 ptr->push_back(makeStringOf(n->Data));
}

void readnodes(AVLTree::Node* n, factstar* ptr,
list<string>* ptrL)
{
```

```
13  if (n->Left != nullptr)
14   readnodes(n->Left, ptr, ptrL);
15  if (n->Right != nullptr)
16   readnodes(n->Right, ptr, ptrL);
17  if (matchfactsstar(n->Data, ptr))
18   ptrL->push_back(makeStringOf(n->Data));
19 }
20
21 void readnodes(AVLTree::Node* n, factstar* ptr,
22 forward_list<string>* ptrFL)
23 {
24  if (n->Left != nullptr)
25   readnodes(n->Left, ptr, ptrFL);
26  if (n->Right != nullptr)
27   readnodes(n->Right, ptr, ptrFL);
28  if (matchfactsstar(n->Data, ptr))
29   ptrFL->push_front(makeStringOf(n->Data));
30 }
31
32 void readnodes(AVLTree::Node* n,
33 forward_list<string>* ptr)
34 {
35  if (n->Left != nullptr)
36   readnodes(n->Left, ptr);
37  if (n->Right != nullptr)
38   readnodes(n->Right, ptr);
39  ptr->push_front(makeStringOf(n->Data));
40 }
41
42 void readnodes(AVLTree::Node* n, string* ptr)
43 {
44  if (n->Left != nullptr)
45   readnodes(n->Left, ptr);
46  if (n->Right != nullptr)
47   readnodes(n->Right, ptr);
48  *ptr += makeStringOf(n->Data);
49  *ptr += ",";
50 }
51
52 void readnodes(AVLTree::Node* n, factstar* ptr,
53 bool* flag)
```

```
54 {
55  if (matchfactsstar(n->Data, ptr))
56   *flag = true;
57  if (!(*flag) && n->Left != nullptr)
58   readnodes(n->Left, ptr, flag);
59  if (!(*flag) && n->Right != nullptr)
60   readnodes(n->Right, ptr, flag);
61 }
62
63 void readnodes(AVLTree::Node* n, size_t hashv,
64 bool* flag)
65 {
66  if (n == nullptr) return;
67  if (n->HashV == hashv)
68  {
69   *flag = true;
70   return;
71  }
72  if (n->HashV > hashv && !(*flag))
73   readnodes(n->Left, hashv, flag);
74  if (n->HashV < hashv && !(*flag))
75   readnodes(n->Right, hashv, flag);
76 }
```

Κεφάλαιο 4

Αποτελέσματα και επιδόσεις

Όλες οι ενέργειες που θα ακολουθήσουν θα γίνουν σε ένα αρχείο που περιέχει 129809 δεδομένα. Σε τέτοιο πλήθος δεδομένων οι χρονικές αποδώσεις δεν ξεπερνούν τα 10 δευτερόλεπτα ενώ πολλές ενέργειες εκτελούνται σε κλάσματα δευτερολέπτων. Το όνομα του επεξεργαστή που πραγματοποιηθήκαν οι μετρήσεις είναι ο Intel Core i7-6500U με συχνότητα 2.5GHz.
Ο τρόπος που υπολογίζεται ο χρόνος που απαιτείται για να εκτελεστή κάποια ενέργεια είναι μέσω της βιβλιοθήκης chrono.

```
1 auto start = high_resolution_clock::now();
2 HT.function_name();
3 auto stop = high_resolution_clock::now();
4 auto duration =
5 duration_cast<microseconds>(stop - start);
6 cout << duration.count() << " ms" << endl;
```

Στην αρχή αποθηκεύεται ο παρόν χρόνος σε ένα στιγμιότυπο start. Μετά την κλήση της συνάρτησης σε ένα στιγμιότυπο stop πάλι αποθηκεύεται ο παρόν χρόνος. Τέλος υπολογίζεται η διαφορά των start και stop και εκτυπώνεται αυτή η διαφορά σε millisecond.

4.1 Εισαγωγή δεδομένων

Σχετικά με την εισαγωγή δεδομένων εκτελούνται δυο συναρτήσεις στην σειρά

1. HT.consult("όνομα_αρχείου.txt");
   Το αρχείο περιέχει 129809 δεδομένα τα οποία φορτώνονται στην μνήμη σε χρόνο 6,3 seconds.

2. HT.assertz("τύπος_δεδομένου(στοιχείο1,στοιχείο2,στοιχείο3,...)");
   Εξετάστηκαν 73 περιπτώσεις στην συνάρτηση assertz με χρόνους 29 milli seconds η πιο γρήγορη εκτέλεση και 1211 mili seconds η πιο αργή. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 71.02 mili seconds.

4.2 Αναζήτηση και κατάργηση δεδομένων

Με δεδομένο τα 129809 δεδομένα να είναι φορτωμένα στην μνήμη εκτελούνται 2 συναρτήσεις.

1. HT.findfact("συμβολοσειρά");
   Δυο περιπτώσεις, η μια με το τύπο δεδομένου, συγκεκριμένα στοιχεία με "*" και η άλλη ο τύπος δεδομένου με όλα τα στοιχεία.

   - HT.findfact("τύπος_δεδομένου(στοιχείο1,στοιχείο2,*)");
     Εξετάστηκαν 73 περιπτώσεις με χρόνους 277 milli seconds η πιο γρήγορη και 0.44 seconds η πιο αργή εκτέλεση. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 0.17 seconds.
   - HT.findfact("τύπος_δεδομένου(στοιχείο1,στοιχείο2,...)");
     Εξετάστηκαν 73 περιπτώσεις με χρόνους 2 milli seconds η πιο γρήγορη και 9 milli seconds η πιο αργή εκτέλεση. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 2.53 milli seconds.

2. HT.retractall("συμβολοσειρά");
   Τρεις περιπτώσεις, η πρώτη με το τύπο δεδομένου, συγκεκριμένα στοιχειά με "*", η δεύτερη ο τύπος δεδομένου με όλα τα στοιχεία και η τρίτη με όρισμα "(*)".

   - HT.retractall("τύπος_δεδομένου(στοιχείο1,στοιχείο2,*)");
     Εξετάστηκαν 73 περιπτώσεις με χρόνους 0.18 seconds η πιο γρήγορη και 0.65 seconds η πιο αργή εκτέλεση. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 0.48 seconds.

- HT.retractall("τύπος_δεδομένου(στοιχείο1,στοιχείο2,...)");
  Εξεταστήκαν 73 περιπτώσεις με χρόνους 22 milli seconds η πιο γρήγορη και 664 milli seconds η πιο αργή εκτέλεση. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 83.46 milli seconds.
- HT.retractall("(*)");
  Σε αυτήν την περίπτωση καταργούνται όλα τα δεδομένα σε 0.22 seconds

4.3 Εξαγωγή δεδομένων

Με δεδομένο τα 129809 δεδομένα να είναι φορτωμένα στην μνήμη εκτελούνται 6 συναρτήσεις.

1. HT.save("όνομα_αρχείου.txt");
   Εξάγονται στο αρχείο όλα τα δεδομένα σε χρόνο 4.53 seconds.

2. HT.exportToFList();
   Εξάγονται σε μια απλή συνδεδεμένη λίστα όλα τα δεδομένα σε χρόνο 4.10 seconds.

3. HT.exportToList();
   Εξάγονται σε μια διπλή συνδεδεμένη λίστα όλα τα δεδομένα σε χρόνο 3.74 seconds.

4. HT.exportToString();
   Εξάγονται σε μια συμβολοσειρά όλα τα δεδομένα σε χρόνο 2.70 seconds.

5. HT.matchedToList("τύπος_δεδομένου(στοιχείο1,στοιχείο2,*)");
   Εξάγονται σε μια διπλή συνδεδεμένη λίστα όλα τα επιθυμητά δεδομένα. Εξετάστηκαν 73 περιπτώσεις με χρόνους 0.19 seconds η πιο γρήγορη και 0.50 seconds η πιο αργή εκτέλεση. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 0.44 seconds.

6. HT.matchedToFList("τύπος_δεδομένου(στοιχείο1,στοιχείο2,*)");
   Εξάγονται σε μια απλή συνδεδεμένη λίστα όλα τα επιθυμητά δεδομένα. Εξετάστηκαν 73 περιπτώσεις με χρόνους 0.17 seconds η πιο γρήγορη και 0.66 seconds η πιο αργή εκτέλεση. Ο μέσος όρος εκτέλεσης της συνάρτησης υπολογίζεται στα 0.46 seconds.

Κεφάλαιο 5

Συμπεράσματα και μελλοντικές προεκτάσεις

Η βιβλιοθήκη διαχειρίζεται σε εξαιρετικούς χρόνους τα δεδομένα, με πιο αργή την λειτουργία consult που φορτώνει από αρχείο 129809 δεδομένα. Αν και καμία ενέργεια δεν χρειάστηκε 7 δευτερόλεπτα να εκτελεστεί, η βιβλιοθήκη προορίζεται για δεδομένα όγκου εκατομμυρίων. Σε περίπτωση που όταν αυξηθούν υπερβολικά τα δεδομένα ο χρόνος εκτέλεσης των συναρτήσεων πιθανών να αυξηθεί πολύ εξ αιτίας του μεγάλου βάθους των δέντρων. Αυτό μπορεί να λυθεί με την αύξηση των θέσεων των δέντρων οπού θα μειωθεί το βάθος του κάθε δέντρου. Μια πολύ σημαντική παρατήρηση είναι πως όταν καλούνται οι συναρτήσεις findfact, retractall, matchedToList, matchedToFList με όρισμα τύπου factstar, δηλαδή τύπο δεδομένου με διακεκριμένα στοιχεία (τύπος_δεδομένου(στοιχείο1,στοιχείο2,*)), η ανταπόκριση των συναρτήσεων αυτών καθυστερεί τουλάχιστον 100 φορές παραπάνω σε σχέση με ένα όρισμα τύπο δεδομένου με όλα τα στοιχεία. Αυτό οφείλεται στον αριθμό κατακερματισμού, που επιστρέφει η συνάρτηση hash από την συμβολοσειρά που δέχεται. Εφόσον ένα factstar σε σχέση με ένα ολοκληρωμένο δεδομένο είναι διαφορετικά μεταξύ τους είναι υποχρεωτικό να εξεταστούν όλοι οι κόμβοι όλων των δέντρων. Σχετικά με την επέκταση των λειτουργιών, με την πάροδο του χρόνου μπορούν να προστεθούν και άλλοι τύποι δεδομένων. Λειτουργίες που θα αποτελούσαν επέκταση της βιβλιοθήκης θα μπορούσαν να ήταν τέτοιου τύπου όπως αυτές του υποκεφαλαίου εξαγωγής δεδομένων. Όσον αφορά την δομή factstar θα μπορούσε να υλοποιηθεί συνάρτηση όπου ο χρήστης θα μπορεί να παραλείπει και να διαλέγει στοιχεία ανεξαρτήτως της φοράς αρχής προς

τέλος. Άλλες δευτερεύων λειτουργιές που θα μπορούσαν να προστεθούν είναι όπως η λειτουργία concat που ενώνει δυο λίστες, δηλαδή λειτουργίες που δεν έχουν καμία επιρροή στη μνήμη. Οι επεκτάσεις που μπορεί να δεχτεί η βιβλιοθήκη βασίζονται στις επιθυμίες και στις ανάγκες του μελλοντικού χρήστη.

Βιβλιογραφία

[1] Herbert Schildt. "Μάθετε τη C++ από το μηδέν". In: Εκδόσεις Κλειδάριθμος, 2004. Chap. 1.

[2] Donald Knuth. *AVL Tree | Set 1 (Insertion).* URL: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/. (latest visit: 13.02.2022).

Κεφάλαιο 6

Οδηγίες για προγραμματιστές

Αφού συμπεριληφθεί το αρχείο AVLHashTable στο προοίμιο του κώδικα, με την προϋπόθεση ότι αυτό και τα άλλα τρία αρχεία interfacetxt, AVL και allFactStructure βρίσκονται στην ίδια διεύθυνση με τον πηγαίο κώδικα, η κάθε αλληλεπίδραση με την βάση δεδομένων γίνεται μέσω της οντότητας HT. Να σημειωθεί οτι το data_with* είναι το δεδομένο που δέχεται η συνάρτηση με διακεκριμένα χαρακτηριστικά δηλαδή στις συγκρίσεις θα ληφθούν υπόψιν τα χαρακτηριστικά πριν το "*" (π.χ. τύπος_δεδομένου(χαρακτηριστικό1,χαρακτηριστικό2,*)). Στην παρούσα κατάσταση η βιβλιοθήκη παρέχει τις εξής 11 λειτουργίες

- void assertz(string Line)
  - HT.assertz("data");
    Εισάγει στην βάση δεδομένων το δεδομένο που δόθηκε ως όρισμα. Η συνάρτηση δεν επιστρέφει τίποτα.
- void consult(string Line)
  - HT.consult("data.txt");
    Εισάγει στην βάση δεδομένων τα δεδομένα που περιέχονται στο αρχείο το οποίο δόθηκε ως όρισμα. Η συνάρτηση δεν επιστρέφει τίποτα.
- bool findfact("data")
  - HT.findfact("data");
  - HT.findfact("data_with*");
    Γίνεται αναζήτηση στην βάση δεδομένων για το δεδομένο που

δόθηκε ως παράμετρος. Η συνάρτηση επιστρέφει true αν βρέθηκε το δεδομένο διαφορετικά επιστρέφει false.

- void retractall(string Line)
  - HT.retractall("data");
  - HT.retractall("data_with*");
    Γίνεται κατάργηση όλων των δεδομένων που είναι όμοια με το δεδομένο που δόθηκε ως όρισμα.
  - HT.retractall("(*)");
    Γίνεται κατάργηση όλων των δεδομένων.
    Σε κάθε περίπτωση η συνάρτηση δεν επιστρέφει τίποτα.
- void save(string filename)
  - HT.save("save1.txt");
    Γίνεται εξαγωγή των δεδομένων σε αρχείο με όνομα που δόθηκε ως όρισμα. Στην περίπτωση που υπάρχει ήδη το αρχείο, όλα τα περιεχόμενα αυτού του αρχείου θα αντικατασταθούν με αυτά της βάσης δεδομένων. Η συνάρτηση δεν επιστρέφει τίποτα.
- list<string> exportToList()
  - HT.exportToList();
    Η συνάρτηση επιστέφει μια διπλή συνδεδεμένη λίστα η οποία περιέχει όλα τα δεδομένα που είναι φορτωμένα στην βάση δεδομένων.
- forward_list<string> exportToFList()
  - HT.exportToFList();
    Η συνάρτηση επιστέφει μια απλή συνδεδεμένη λίστα η οποία περιέχει όλα τα δεδομένα που είναι φορτωμένα στην βάση δεδομένων.
- string exportToString()
  - HT.exportToString();
    Η συνάρτηση επιστέφει μια συμβολοσειρά η οποία περιέχει όλα τα δεδομένα που είναι φορτωμένα στην βάση δεδομένων χωρισμένα από κόμμα ",".
- list<string> matchedToList(string Line)

  - HT.matchedToList("data_with*");
    Η συνάρτηση επιστέφει μια διπλή συνδεδεμένη λίστα η οποία περιέχει δεδομένα τα οποία είναι όμοια με αυτό που δόθηκε ως όρισμα.

- forward_list<string> matchedToFList(string Line)

  - HT.matchedToFList("data_with*");
    Η συνάρτηση επιστέφει μια μονή συνδεδεμένη λίστα η οποία περιέχει δεδομένα τα οποία είναι όμοια με αυτό που δόθηκε ως όρισμα.

- list<T> concat(list<T> list1, list<T> list2, list<T> list3)

  - HT.concat(list1,list2,list3);
    Η συνάρτηση ωθεί στην αρχή τα περιεχόμενα της πρώτης λίστας στην τρίτη λίστα και στην συνέχεια ωθεί τα περιεχόμενα της δεύτερης λίστας στην τρίτη λίστα. Η τρίτη λίστα θα διατηρεί τα προηγούμενα περιεχόμενα της. Η συνάρτηση επιστρέφει την τρίτη λίστα.

Ακολουθεί ένα παράδειγμα ροής προγράμματος όπου παρουσιάζεται ο τρόπος με τον οποίο αξιοποιούνται οι δυνατότητες της βιβλιοθήκης.

```
1  #include "AVLHashTable.h"
2
3  using namespace std;
4
5  int main()
6  {
7   list<string> listString1, listString2,
8    listString3;
9   forward_list<string> listFString1, listFString2,
10   listFString3;
11  string justString;
12
13  HT.consult("AllSamplesSorted.txt");
14
15  if (HT.findfact("data_stmt(*)"))
16  {
17   //actions
18  }
19
```

```
20  if (HT.findfact("sequence(1,\"my_proc3\",5)"))
21  {
22   //actions
23  }
24
25  HT.assertz("combo(1,my_func1,1)");
26
27  HT.save("save1.txt");
28
29  HT.retractall("global_nils(*)");
30
31  HT.save("save2.txt");
32
33  HT.retractall("consecutive_106(\"false\")");
34
35  HT.save("save3.txt");
36
37  HT.retractall("data_stmt(\"all_proc\",\"a\",*)");
38
39  HT.save("save4.txt");
40
41  listFString1 = HT.exportToFList();
42
43  listString1 =  HT.exportToList();
44
45  justString = HT.exportToString();
46
47  listString2 = HT.matchedToList("data_stmt(*)");
48
49  listFString2 = HT.matchedToFList("global_nils(*)");
50
51  HT.retractall("(*)");
52
53  HT.consult("save5.txt");
54
55  listString3 =
56  HT.concat(listString1, listString2, listString3);
57 }
58 return 0;
```

Στην αρχή φορτώνεται στην μνήμη το πλήθος των δεδομένων που περιέχεται στο αρχείο AllSamplesSorted.txt(γραμμή 13). Στην συνέχεια γίνεται αναζήτηση για συγκεκριμένους τύπους δεδομένων και άμα βρεθούν στην μνήμη αυτά τα δεδομένα μπορεί να ακολουθήσει μια σειρά ενεργειών(γραμμή 15, 20). Ύστερα προστίθεται στην μνήμη ένα μεμονωμένο δεδομένο(γραμμή 25) και μετά αποθηκεύονται τα δεδομένα, που βρίσκονται στην μνήμη, σε ένα αρχείο με όνομα save1.txt(γραμμή 27). Αφού διαγραφούν όλα τα δεδομένα του συγκεκριμένου τύπου(γραμμή 29) θα αποθηκευτούν τα δεδομένα στο αρχείο με όνομα save2.txt(γραμμή 31). Άλλη μια αποθήκευση των δεδομένων στο αρχείο save3.txt(γραμμη 35) γίνεται μετά την διαγραφή όλων των διπλοτύπων δεδομένων(γραμμή 33). Η ίδια διαδικασία επαναλαμβάνεται(εως την γραμμή 40) με την διαφορά ότι στην παρούσα περίπτωση διαγραφής θα ληφθούν υπόψιν στην σύγκριση ο τύπος δεδομένου και τα δύο πρώτα χαρακτηριστικά. Στην συνέχεια η ροή του κώδικα έχει ως εξής. Εξάγονται τα δεδομένα και αποθηκεύονται σε μια απλή συνδεδεμένη λίστα(γραμμή 41), σε μια διπλά συνδεδεμένη λίστα(γραμμή 43), σε μια συμβολοσειρά(γραμμή 45). Σε μια διπλή συνδεδεμένη λίστα αποθηκεύονται τα δεδομένα του συγκεκριμένου τύπου δεδομένου(γραμμή 47) και σε μια απλή συνδεδεμένη λίστα θα αποθηκευτούν τα δεδομένα του συγκεκριμένου τύπου δεδομένου(γραμμή 49). Πριν το τέλος, θα αποτύχει να δημιουργηθεί το αρχείο save5(γραμμή 53) διότι καταργήθηκαν όλα τα δεδομένα από την μνήμη από την προηγούμενη ενέργεια(γραμμή 51). Τέλος θα αποθηκευτούν σε μια απλή λίστα το αποτέλεσμα της ώθησης των λιστών, που χρησιμοποιήθηκαν νωρίτερα, πρώτα της πρώτης στην τρίτη και μετά της δεύτερης στην τρίτη. Να σημειωθεί ότι το απόσπασμα αποτελεί παράδειγμα και η αποδοτικότητα της αξιοποίησης της βιβλιοθήκης βασίζεται στην δημιουργικότητα του ίδιου του προγραμματιστή.

Κεφάλαιο 7

Παράρτημα

7.1 AVL.h

```
#pragma once
#include "AVLHashTable.h"
#include "allFactStructure.h"
#include "interfacetxt.h"
#include "AVLHashTable.h"
#include <cstdio>
#include <sstream>
#include <forward_list>

using namespace std;

class AVLTree
{
private:

class Node
{
public:
size_t HashV = NULL;
GeneralFact* Data = nullptr;
Node* Left = nullptr;
Node* Right = nullptr;

Node()
```

```
: HashV(NULL), Data(nullptr), Left(nullptr), Right(nullptr) {}

//Default Constructor Node
Node(GeneralFact* NewData, size_t hashv)
: Data(NewData), Left(nullptr), Right(nullptr), HashV(hashv) {}

 //w/ string
Node(GeneralFact* NewData, string line)
: Data(NewData), Left(nullptr), Right(nullptr)
{
HashV = hash< string >{}(line);
}

//Constructor Node
Node(GeneralFact* NewData, Node* LeftPtr, Node* RightPtr, size_t HV)
: Data(NewData), Left(LeftPtr), Right(RightPtr), HashV(HV)
{}

~Node()
{
if (Data != nullptr)
{
delete Data;
Data = nullptr;
HashV = NULL;
Left = nullptr;
Right = nullptr;
}
}

//Copy Constructor Node
Node(Node& other)
{
if (this != &other)
{
if (this->Left != nullptr && other.Left != nullptr)
this->Left = &other;
if (this->Right != nullptr && other.Right != nullptr)
this->Right = &other;
this->Data = other.Data;
this->Left = other.Left;
```

```
this->Right = other.Right;
this->HashV = other.HashV;
}
}


//Copy operator Node
Node operator=(Node& other)
{
if (this != &other)
{
if (this->Left != nullptr && other.Left != nullptr)
this->Left = &other;
if (this->Right != nullptr && other.Right != nullptr)
this->Right = &other;
this->Data = other.Data;
this->Left = other.Left;
this->Right = other.Right;
this->HashV = other.HashV;
}
return *this;
}

// Move constructor Node
Node(Node&& other) noexcept
: Data(nullptr), Left(nullptr), Right(nullptr), HashV(NULL)
{
*this = move(other);
}

Node& operator=(Node&& other) noexcept
{
if (this != &other)
{
if (this->Left != nullptr && other.Left != nullptr)
this->Left = &other;
if (this->Right != nullptr && other.Right != nullptr)
this->Right = &other;
delete(Left);
delete(Right);
delete(Data);
```

```
HashV = NULL;

this->Data = other.Data;
this->Left = other.Left;
this->Right = other.Right;
this->HashV = other.HashV;

other.Data = nullptr;
other.Left = nullptr;
other.Right = nullptr;
other.HashV = NULL;
}
return *this;
}
};

Node* Root = nullptr;
size_t NumNodes = 0;

int height(Node* p);
int difference(Node* p);
Node* rr_rotat(Node* p);
Node* ll_rotat(Node* p);
Node* lr_rotat(Node* p);
Node* rl_rotat(Node* p);
Node* balance(Node* p);
Node* Insert(Node* r, GeneralFact* v,size_t hashv);
typename AVLTree::Node* Release(Node* r);
AVLTree::Node* delNode(Node* r, size_t Hash, bool* ptrdup);
AVLTree::Node* delNode(Node* r, factstar* fs, bool* ptrdup);
string show(Node* p);
void storeToVector(Node* node, vector<string>* ptr);

public:
AVLTree() noexcept
: Root(nullptr), NumNodes(0)
{} // Default constructor

AVLTree(Node* node1)
:Root(node1) {}
```

```
AVLTree(GeneralFact* fact,string line)
: Root(new Node(fact,line)) {}


~AVLTree() noexcept
{
Root = Release(Root); //delete all function
} // Destructor


// Copy constructor
AVLTree(const AVLTree& other)
{
if (this != &other)
{
this->Root = other.Root;
this->NumNodes = other.NumNodes;
}
}

// Move constructor
AVLTree(AVLTree&& other)noexcept
: Root(nullptr), NumNodes(0)
{
*this = other;
}

// Copy operator
AVLTree operator=(const AVLTree& other)
{
if (this != &other)
{
this->Root = other.Root;
this->NumNodes = other.NumNodes;
}
return *this;
}

// Move operator
AVLTree& operator=(AVLTree&& other)noexcept
{
```

```
if (this != &other)
{
Root = nullptr;
Root = other.Root;
other.Root = nullptr;
}
return *this;
}

int height()
{
height(this->Root);
}
int difference()
{
difference(this->Root);
}
Node* rr_rotat()
{
rr_rotat(this->Root);
}
Node* ll_rotat()
{
ll_rotat(this->Root);
}
Node* lr_rotat()
{
lr_rotat(this->Root);
}
Node* rl_rotat()
{
rl_rotat(this->Root);
}
Node* balance()
{
balance(this->Root);
}
Node* Insert(GeneralFact* v,size_t hashv)
{
return Insert(this->Root, v, hashv);
}
```

```
friend class HashTable;
friend typename AVLTree::Node* FindMin(AVLTree::Node* root);
friend void readnodes(Node* n, list<string>* ptr);
friend void readnodes(Node* n, forward_list<string>* ptr);
friend void readnodes(Node* n, string* ptr);
friend void readnodes(AVLTree::Node* n, factstar* ptr, list<string>* ptrL);
friend void readnodes(AVLTree::Node* n, factstar* ptr, forward_list<string>* ptrFL);
friend void readnodes(AVLTree::Node* n, factstar* ptr, bool* flag);
friend void readnodes(AVLTree::Node* n, size_t hashv, bool* flag);
};

//====================PRIVATE CODES OF FUNCTIONS

int AVLTree::height(Node* p) {
int h = 0;
if (p != NULL)
{
int l_height = height(p->Left);
int r_height = height(p->Right);
int max_height = max(l_height, r_height);
h = max_height + 1;
}
return h;
}

int AVLTree::difference(Node* p) {
int l_height = height(p->Left);
int r_height = height(p->Right);
int fb = l_height - r_height;
return fb;
}

typename AVLTree::Node* AVLTree::rr_rotat(Node* p)
{
Node* t;
t = p->Right;
p->Right = t->Left;
t->Left = p;
return t;
}
```

```
typename AVLTree::Node* AVLTree::ll_rotat(Node* p)
{
Node* t;
t = p->Left;
p->Left = t->Right;
t->Right = p;
return t;
}

typename AVLTree::Node* AVLTree::lr_rotat(Node* p)

{
Node* t;
t = p->Left;
p->Left = rr_rotat(t);
return ll_rotat(p);
}

typename AVLTree::Node* AVLTree::rl_rotat(Node* p)

{
Node* t;
t = p->Right;
p->Right = ll_rotat(t);
return rr_rotat(p);
}

typename AVLTree::Node* AVLTree::balance(Node* t)
{
if (t != nullptr)
{
int bal_factor = difference(t);
if (bal_factor > 1) {
if (difference(t->Left) > 0)
t = ll_rotat(t);
else
t = lr_rotat(t);
}
else if (bal_factor < -1) {
if (difference(t->Right) > 0)
t = rl_rotat(t);
```

```
else
t = rr_rotat(t);
}
}
return t;
}

typename AVLTree::Node* AVLTree::Insert(Node* r, GeneralFact* v,size_t hashv)
{
if (r == nullptr)
{
r = new Node(v,hashv);
NumNodes++;
return r;
}
else if (r->HashV < hashv)
{
r->Right = Insert(r->Right, v, hashv);
}
else
{
r->Left = Insert(r->Left, v, hashv);
}
return r;
}

typename AVLTree::Node* AVLTree::Release(Node* r)
{
if (r != nullptr)
{
Release(r->Left);
Release(r->Right);
delete(r);
this->NumNodes--;
r = nullptr;
}
return r;
}

typename AVLTree::Node* AVLTree::delNode(Node* r , size_t Hash, bool* ptrdup)
{
```

```
if (r == nullptr) return nullptr;
if (Hash < r->HashV) r->Left = delNode(r->Left, Hash, ptrdup);
else if (Hash > r->HashV) r->Right = delNode(r->Right, Hash, ptrdup);
else if (r->Left == nullptr && r->Right == nullptr)
{
*ptrdup = false;
delete r;
r = nullptr;
NumNodes--;
}
else if (r->Left == nullptr)
{
*ptrdup = false;
Node* n = r;
r = r->Right;
delete n;
NumNodes--;
}
else if (r->Right == nullptr)
{
*ptrdup = false;
Node* n = r;
r = r->Left;
delete n;
NumNodes--;
}
else
{
*ptrdup = false;
Node* n = FindMin(r->Right);
string tmp = makeStringOf(n->Data);
delete r->Data;
r->Data = makeInstanceOf(tmp);
r->HashV = n->HashV;
r->Right = delNode(r->Right, n->HashV, ptrdup);
}
return r;
}

typename AVLTree::Node* AVLTree::delNode(Node* r, factstar* fs, bool* ptrdup)
{
```

```
if (r == nullptr) return r;

if (r->Left != nullptr && r != nullptr) r->Left = delNode(r->Left, fs, ptrdup);
if (r->Right != nullptr && r != nullptr) r->Right = delNode(r->Right, fs, ptrdup);
if (matchfactsstar(r->Data, fs))
{
*ptrdup = false;
if (r->Left == nullptr && r->Right == nullptr)
{
delete r;
r = nullptr;
NumNodes--;
}
else if (r->Left == nullptr)
{
Node* n = r;
r = r->Right;
delete n;
NumNodes--;
}
else if (r->Right == nullptr)
{
Node* n = r;
r = r->Left;
delete n;
NumNodes--;
}
else
{
Node* n = FindMin(r->Right);
string tmp = makeStringOf(n->Data);
delete r->Data;
r->Data = makeInstanceOf(tmp);
r->HashV = n->HashV;
r->Right = delNode(r->Right, n->HashV, ptrdup);
}
}return r;
}

typename AVLTree::Node* FindMin(AVLTree::Node* root)
{
```

```
auto ptr = root;
while (ptr && ptr->Left != nullptr)
ptr = ptr->Left;
return ptr;
}

string AVLTree::show(Node* p)
{
string ss;
if (p->Right != nullptr)
ss += show(p->Right);
if (p->Left != nullptr)
ss += show(p->Left);
return makeStringOf(p->Data) + "\n";
}

void AVLTree::storeToVector(Node* node, vector<string>* ptr)
{
if (node == nullptr) return;
if (node->Left != nullptr)
{
storeToVector(node->Left, ptr);
}
if (node->Right != nullptr)
{
storeToVector(node->Right, ptr);
}
ptr->push_back(makeStringOf(node->Data));
}
```

7.2 allFactStructure.h

```
#pragma once
#include <vector>
#include <sstream>
#include <iostream>

using namespace std;

class GeneralFact
{
bool fordestr = 1;
public:
virtual void operator=(GeneralFact& other) = 0;
virtual ~GeneralFact() = 0;
};
GeneralFact::~GeneralFact()
{
fordestr = 1;
}

class factstar
{
GeneralFact* fact{};
int params{};
public:
factstar() { fact = nullptr; params = NULL; }
factstar(GeneralFact* q, int w)
: fact(q), params(w) {}

factstar(const factstar& other)
{
if (this != &other)
{
this->fact = other.fact;
this->params = other.params;
}
}
~factstar()
{
```

```
delete fact;
params = NULL;
}

void operator=(factstar& other)
{
if (this != &other)
{
this->fact = other.fact;
this->params = other.params;
}
}

factstar(factstar&& other) noexcept
: fact{ other.fact }, params{ other.params }{}

void operator=(factstar&& other) noexcept
{
swap(fact, other.fact);
swap(params, other.params);
}
friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class type_def : public GeneralFact
{
int q{}, e{}, t{}, u{}, i{}, o{};
string w{}, r{}, y{};
public:
type_def(const int q1, const int e1, const int t1, const int u1,
const int i1, const int o1,const string w1,const string r1, const string y1)
:q(q1), e(e1), t(t1), u(u1), i(i1), o(o1), w(w1), r(r1), y(y1){}

type_def(const type_def& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e; this->i = other.i;
```

```
this->o = other.o;
}
}
~type_def()
{
this->q = NULL; this->r.clear();
this->t = NULL; this->y.clear();
this->u = NULL; this->w.clear();
this->e = NULL; this->i = NULL;
this->o = NULL;

}
void operator=(GeneralFact& other) override
{
type_def* ptr = dynamic_cast<type_def*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->e = ptr->e; this->i = ptr->i;
this->o = ptr->o;
}
type_def(type_def&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u }, i{ other.i },
o{ other.o }, w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(type_def&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(i, other.i); std::swap(o, other.o);
std::swap(w, other.w); std::swap(r, other.r); std::swap(y, other.y);
}

friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);

};//page 5

class op_def : public GeneralFact
{
int q{}, r{}, t{}, y{}, u{};
string w{}, e{};
```

```
public:
op_def(const int q1, const int r1, const int t1, const int y1,
const int u1, const string w1, const string e1)
:q(q1), r(r1), t(t1), y(y1), u(u1), w(w1), e(e1){}
op_def(const op_def& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e;
}
}
~op_def()
{
this->q = NULL; this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL; this->w.clear();
this->e.clear();

}
void operator=(GeneralFact& other) override
{
op_def* ptr = dynamic_cast<op_def*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->e = ptr->e;
}
op_def(op_def&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u },
w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(op_def&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(w, other.w); std::swap(r, other.r);
std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);
```

```
friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 6

class hierarchy_part : public GeneralFact
{
int q{}, e{}, t{}, y{}, u{};
string w{}, r{};
public:
hierarchy_part(const int q1, const int e1, const int t1, const int y1,
const int u1, const string w1, const string r1)
:q(q1), e(e1), t(t1), y(y1), u(u1), w(w1), r(r1){}
hierarchy_part(const hierarchy_part& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e;
}
}
~hierarchy_part()
{
this->q = NULL; this->r.clear();
this->t = NULL; this->y = NULL;
this->u = NULL; this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
hierarchy_part* ptr = dynamic_cast<hierarchy_part*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->e = ptr->e;
}
hierarchy_part(hierarchy_part&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u },
w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(hierarchy_part&& other)  noexcept
```

```
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(w, other.w); std::swap(r, other.r);
std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 7

class data_stmt : public GeneralFact
{
int e{}, r{};
string q{}, w{}, t{}, y{};
public:
data_stmt(const int e1, const int r1, string q1, string w1,
string t1, const string y1)
:e(e1), r(r1), q(q1), w(w1), t(t1), y(y1) {}
data_stmt(const data_stmt& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->w = other.w; this->e = other.e;
}
}
~data_stmt()
{
this->q.clear(); this->r = NULL;
this->t.clear(); this->y.clear();
this->w.clear(); this->e = NULL;

}
void operator=(GeneralFact& other) override
{
data_stmt* ptr = dynamic_cast<data_stmt*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->w = ptr->w; this->e = ptr->e;
}
```

```
data_stmt(data_stmt&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t },
w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(data_stmt&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(w, other.w); std::swap(r, other.r); std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 8

class prog_stmt : public GeneralFact
{
int w{}, e{}, r{}, t{}, y{}, u{}, i{};
string q{};
public:
prog_stmt(const int w1, const int e1, const int r1, const int t1,
const int y1, const int u1, const int i1, const string q1)
:w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), q(q1){}
prog_stmt(const prog_stmt& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e; this->i = other.i;
}
}
~prog_stmt()
{
this->q.clear();this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL; this->w = NULL;
this->e = NULL; this->i = NULL;

}
void operator=(GeneralFact& other) override
{
```

```
prog_stmt* ptr = dynamic_cast<prog_stmt*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->e = ptr->e; this->i = ptr->i;
}
prog_stmt(prog_stmt&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u }, i{ other.i },
w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(prog_stmt&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(i, other.i); std::swap(w, other.w);
std::swap(r, other.r); std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 9

class joint_stmt : public GeneralFact
{
int w{}, e{}, r{}, t{}, y{};
string q{};
public:
joint_stmt(const int w1, const int e1, const int r1, const int t1,
const int y1, string q1)
:w(w1), e(e1), r(r1), t(t1), y(y1), q(q1){}
joint_stmt(const joint_stmt& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->w = other.w; this->e = other.e;
}
}

~joint_stmt()
{
this->q.clear();this->r = NULL;
```

```
this->t = NULL; this->y = NULL;
this->w = NULL; this->e = NULL;

}
void operator=(GeneralFact& other) override
{
joint_stmt* ptr = dynamic_cast<joint_stmt*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->w = ptr->w; this->e = ptr->e;
}
joint_stmt(joint_stmt&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t },
w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(joint_stmt&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(w, other.w); std::swap(r, other.r); std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 12

//vector
class call_stmt : public GeneralFact
{
int w{}, e{};
string q{};
vector<int>r{};
public:
call_stmt(const int w1, const int e1,const string q1, const vector<int> r1)
:w(w1), e(e1), q(q1), r(r1) {
r.shrink_to_fit();
}
call_stmt(const call_stmt& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
```

```
}
}
~call_stmt()
{
this->q.clear();this->r.clear();
this->w = NULL; this->e = NULL;

}
void operator=(GeneralFact& other) override
{
call_stmt* ptr = dynamic_cast<call_stmt*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
}
call_stmt(call_stmt&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r } {}
void operator=(call_stmt&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 13

//vector
class compo_stmt : public GeneralFact
{
int w{};
string q{};
vector<int>r{};
public:
compo_stmt(const int w1, string q1, const vector<int> r1)
:w(w1), q(q1), r(r1) {
r.shrink_to_fit();
}
compo_stmt(const compo_stmt& other)
{
if (this != &other)
{
```

```
this->q = other.q; this->r = other.r;
this->w = other.w;
}
}
~compo_stmt()
{
this->q.clear();this->r.clear();
this->w = NULL;

}
void operator=(GeneralFact& other) override
{
compo_stmt* ptr = dynamic_cast<compo_stmt*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w;
}
compo_stmt(compo_stmt&& other)  noexcept
: q{ other.q }, w{ other.w }, r{ other.r } {}
void operator=(compo_stmt&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 14

//vector
class rec_stmt : public GeneralFact
{
int w{};
string q{};
vector<int>e{};
public:
rec_stmt(const int w1,const string q1, const vector<int> e1)
:w(w1), q(q1), e(e1) {
e.shrink_to_fit();
}
rec_stmt(const rec_stmt& other)
{
if (this != &other)
```

```
{
this->q = other.q;
this->w = other.w;
this->e = other.e;
}
}
~rec_stmt()
{
this->q.clear();
this->w = NULL;
this->e.clear();

}
void operator=(GeneralFact& other) override
{
rec_stmt* ptr = dynamic_cast<rec_stmt*>(&other);
this->q = ptr->q;
this->w = ptr->w;
this->e = ptr->e;
}
rec_stmt(rec_stmt&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w } {}

void operator=(rec_stmt&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 16

class special_op : public GeneralFact
{
int w{}, r{}, t{}, y{}, i{}, o{}, p{};
string q{}, e{}, u{};
public:
special_op(const int w1, const int r1, const int t1, const int y1,
const int i1, const int o1, const int p1,const string q1,
const string e1, const string u1)
:w(w1), r(r1), t(t1), y(y1), i(i1), o(o1), p(p1), q(q1), e(e1), u(u1){}
```

```
special_op(const special_op& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e; this->i = other.i;
this->o = other.o; this->p = other.p;
}
}
~special_op()
{
this->q.clear();this->r = NULL;
this->t = NULL; this->y = NULL;
this->u.clear();this->w = NULL;
this->e.clear();this->i = NULL;
this->o = NULL; this->p = NULL;

}
void operator=(GeneralFact& other) override
{
special_op* ptr = dynamic_cast<special_op*>(&other);
this->q = ptr->q; r = ptr->r;
this->t = ptr->t; y = ptr->y;
this->u = ptr->u; w = ptr->w;
this->e = ptr->e; i = ptr->i;
this->o = ptr->o; p = ptr->p;
}
special_op(special_op&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u }, i{ other.i },
o{ other.o }, w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(special_op&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(i, other.i); std::swap(o, other.o);
std::swap(w, other.w); std::swap(r, other.r); std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
```

```
};//page 23

class special_dt : public GeneralFact
{
int w{}, r{};
string q{}, e{}, t{}, y{}, u{};
public:
special_dt(const int w1, const int r1, const string q1, const string e1,
const string t1, const string y1,const string u1)
:w(w1), r(r1), q(q1), e(e1), t(t1), y(y1), u(u1){}
special_dt(const special_dt& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e;
}
}
~special_dt()
{
this->q.clear(); this->r = NULL;
this->t.clear(); this->y.clear();
this->u.clear(); this->w = NULL;
this->e.clear();

}
void operator=(GeneralFact& other) override
{
special_dt* ptr = dynamic_cast<special_dt*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->e = ptr->e;
}
special_dt(special_dt&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u },
w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(special_dt&& other)  noexcept
{
```

```
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(w, other.w); std::swap(r, other.r);
std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 24

class local_object : public GeneralFact
{
int w{}, t{}, i{};
string q{}, e{}, r{}, y{}, u{}, o{};
public:
local_object(const int t1, const int w1, const int i1, const string q1,
const string e1, const string y1, const string u1, const string r1,
const string o1)
:t(t1), w(w1), i(i1), q(q1), e(e1), y(y1), u(u1), r(r1), o(o1){}
local_object()
:w(0),i(0),r(""),t(0),q(""),e(""),y(""),u(""),o(""){}
local_object(const local_object& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->i = other.i; this->o = other.o;
this->e = other.e;
}
}
~local_object()
{
this->q.clear();this->r.clear();
this->t = NULL; this->y.clear();
this->u.clear();this->w = NULL;
this->i = NULL; this->o.clear();
this->e.clear();

}
void operator=(GeneralFact& other) override
```

```
{
local_object* ptr = dynamic_cast<local_object*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->i = ptr->i; this->o = ptr->o;
this->e = ptr->e;
}

void operator=(local_object& other)
{
local_object* ptr = new local_object();
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->i = ptr->i; this->o = ptr->o;
this->e = ptr->e;
}

local_object(local_object&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u }, i{ other.i },
o{ other.o }, w{ other.w }, r{ other.r }, y{ other.y } {}
void operator=(local_object&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(i, other.i); std::swap(o, other.o);
std::swap(w, other.w); std::swap(r, other.r); std::swap(y, other.y);
}
bool operator==(local_object other)
{
if (this->q == other.q && this->w == other.w &&
this->e == other.e && this->r == other.r &&
this->t == other.t && this->y == other.y &&
this->u == other.u && this->i == other.i &&
this->o == other.o) return true;
else return false;
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 25
```

```
class state_node : public GeneralFact
{
int w{};
string q{};
public:
state_node(const int w1, const string q1)
:w(w1), q(q1) {}
state_node()
:w(0),q(""){}
~state_node()
{
this->q.clear();this->w = NULL;

}
void operator=(GeneralFact& other) override
{
state_node* ptr = dynamic_cast<state_node*>(&other);
this->q = ptr->q; this->w = ptr->w;
}

state_node(state_node&& other)  noexcept
: q{ other.q }, w{ other.w }  {}

void operator=(state_node&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 26

//vector
class subprogram_call : public state_node
{
int w{};
vector<int>q{};
public:
subprogram_call(const int w1, const vector<int>q1, const int e1, const string r1)
:state_node(e1, r1), w(w1), q(q1) {
```

```
q.shrink_to_fit();
}
subprogram_call(const subprogram_call& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~subprogram_call()
{
state_node* ptr = dynamic_cast<state_node*>(this);
this->q.clear(); this->w = NULL;
}
void operator=(const subprogram_call& other)
{
q = other.q; w = other.w;
}
subprogram_call(subprogram_call&& other)  noexcept
: q{ other.q }, w{ other.w } {}
void operator=(subprogram_call&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class dataflow : public state_node
{
int w{};
vector<int>q{};
public:
dataflow(const int w1, const vector<int> q1, const int e1, const string r1)
:state_node(e1, r1), w(w1), q(q1) {
q.shrink_to_fit();
}
dataflow(const dataflow& other)
{
```

```
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~dataflow()
{
state_node* ptr = dynamic_cast<state_node*>(this);
this->q.clear(); this->w = NULL;
}
void operator=(const dataflow& other)
{
q = other.q; w = other.w;
}
dataflow(dataflow&& other)  noexcept
: q{ other.q }, w{ other.w } {}
void operator=(dataflow&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class ifthen : public state_node
{
int r{}, t{};
vector<int> q{}, w{}, e{};
public:
ifthen(const int r1, const int t1, const vector<int> q1, const vector<int> w1,
const vector<int> e1, const int y1, const string u1)
:state_node(y1, u1), r(r1), t(t1), q(q1), w(w1), e(e1) {
q.shrink_to_fit();
w.shrink_to_fit();
e.shrink_to_fit();
}
ifthen(const ifthen& other)
{
if (this != &other)
```

```
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t;
}
}
~ifthen()
{
state_node* ptr = dynamic_cast<state_node*>(this);
this->q.clear(); this->w.clear();
this->e.clear(); this->r = NULL;
this->t = NULL;
}
void operator=(const ifthen& other)
{
q = other.q; w = other.w;
e = other.e; r = other.r;
t = other.t;
}
ifthen(ifthen&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, w{ other.w }, r{ other.r } {}
void operator=(ifthen&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(w, other.w); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class jump : public state_node
{
int w{};
vector<int> q{};
public:
jump(const int w1, const vector<int> q1, const int e1, const string r1)
:state_node(e1, r1), w(w1), q(q1) {
q.shrink_to_fit();
}
```

```
jump(const jump& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~jump()
{
state_node* ptr = dynamic_cast<state_node*>(this);
this->q.clear(); this->w = NULL;
}
void operator=(const jump& other)
{
q = other.q; w = other.w;
}
jump(jump&& other)  noexcept
: q{ other.q }, w{ other.w } {}
void operator=(jump&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class return_cos : public state_node
{
vector<int>q;
public:
return_cos(const vector<int> q1, const int e1, const string r1)
:state_node(e1, r1), q(q1) {
q.shrink_to_fit();
}
return_cos(const return_cos& other)
{
if (this != &other)
{
this->q = other.q;
```

```
}
}
~return_cos()
{
state_node* ptr = dynamic_cast<state_node*>(this);
this->q.clear();
}
void operator=(const return_cos& other)
{
q = other.q;
}
return_cos(return_cos&& other)  noexcept
: q{ other.q } {}
void operator=(return_cos&& other)  noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class change_op_number : public GeneralFact
{
int w{}, e{};
string q{};
public:
change_op_number(const int e1, const int w1, const string q1)
:e(e1), w(w1), q(q1){}
change_op_number(const change_op_number& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~change_op_number()
{
this->q.clear();this->w = NULL;
this->e = NULL;
```

```
}
void operator=(GeneralFact& other) override
{
change_op_number* ptr = dynamic_cast<change_op_number*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}
change_op_number(change_op_number&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w } {}
void operator=(change_op_number&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 28

class last_change_op_number : public GeneralFact
{
int w{}, e{};
string q{};
public:
last_change_op_number(const int e1, const int w1, const string q1)
:e(e1), w(w1), q(q1){}
last_change_op_number(const last_change_op_number& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~last_change_op_number()
{
this->q.clear(); this->w = NULL;
this->e = NULL;

}
void operator=(GeneralFact& other) override
```

```
{
last_change_op_number* ptr = dynamic_cast<last_change_op_number*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}
last_change_op_number(last_change_op_number&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w } {}
void operator=(last_change_op_number&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 28

//vector
class op_guards : public GeneralFact
{
int w{};
string q{};
vector<int>e{}, r{};
public:
op_guards(const int w1, string q1, const vector<int>e1, const vector<int> r1)
:w(w1), q(q1), e(e1), r(r1) {
e.shrink_to_fit();
r.shrink_to_fit();
}
op_guards(const op_guards& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
}
}
~op_guards()
{
this->q.clear();this->r.clear();
this->w = NULL; this->e.clear();
```

```
}
void operator=(GeneralFact& other) override
{
op_guards* ptr = dynamic_cast<op_guards*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
}
op_guards(op_guards&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r } {}
void operator=(op_guards&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 29

//vector
class var_guards : public GeneralFact
{
int w{};
string q{};
vector<int>e{}, r{};
public:
var_guards(const int w1, string q1, const vector<int> e1, const vector<int> r1)
:w(w1), q(q1), e(e1), r(r1) {
e.shrink_to_fit();
r.shrink_to_fit();
}
var_guards(const var_guards& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
}
}
~var_guards()
{
```

```
this->q.clear(); this->r.clear();
this->w = NULL; this->e.clear();

}
void operator=(GeneralFact& other) override
{
var_guards* ptr = dynamic_cast<var_guards*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
}
var_guards(var_guards&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r } {}
void operator=(var_guards&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 30

class guard_pair : public GeneralFact
{
int w{}, e{};
string q{}, r{};
public:
guard_pair(const int w1, const int e1, string q1, string r1)
:w(w1), e(e1), q(q1), r(r1){}
guard_pair(const guard_pair& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
}
}
~guard_pair()
{
this->q.clear();this->r.clear();
this->w = NULL; this->e = NULL;
```

```
}
void operator=(GeneralFact& other) override
{
guard_pair* ptr = dynamic_cast<guard_pair*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
}
guard_pair(guard_pair&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r } {}
void operator=(guard_pair&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 30

class guard_cond : public GeneralFact
{
int w{}, e{};
string q{}, r{};
public:
guard_cond(const int w1, const int e1, const string q1, const string r1)
:w(w1), e(e1), q(q1), r(r1){}
guard_cond(const guard_cond& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
}
}
~guard_cond()
{
this->q.clear(); this->r.clear();
this->w = NULL; this->e = NULL;

}
```

```
void operator=(GeneralFact& other) override
{
guard_cond* ptr = dynamic_cast<guard_cond*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
}
guard_cond(guard_cond&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r } {}
void operator=(guard_cond&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 31

//vector
class predecessors : public GeneralFact
{
int w{};
string q{};
vector<int>e{};
public:
predecessors(const int w1, string q1, const vector<int> e1)
:w(w1), q(q1), e(e1) {
e.shrink_to_fit();
}
predecessors(const predecessors& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~predecessors()
{
this->q.clear(); this->w = NULL;
this->e.clear();
```

```
}
void operator=(GeneralFact& other) override
{
predecessors* ptr = dynamic_cast<predecessors*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}
predecessors(predecessors&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w } {}
void operator=(predecessors&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 32

class cessor : public GeneralFact
{
int w{}, e{};
string q{};
public:
cessor(const int w1, const int e1, const string q1)
:w(w1), e(e1), q(q1){}
cessor(const cessor& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~cessor()
{
this->q.clear(); this->w = NULL;
this->e = NULL;

}
void operator=(GeneralFact& other) override
```

```
{
cessor* ptr = dynamic_cast<cessor*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}
cessor(cessor&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w } {}
void operator=(cessor&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 33

//vector
class cessor_kind : public GeneralFact
{
int e{}, r{};
string q{}, w{}, t{}, y{};
vector<int>u{};
public:
cessor_kind(const int e1, const int r1, const string q1, const string w1,
const string t1, const string y1, const vector<int> u1)
:e(e1), r(r1), q(q1), w(w1), t(t1), y(y1), u(u1) {
u.shrink_to_fit();
}
cessor_kind(const cessor_kind& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->w = other.w;
this->e = other.e;
}
}
~cessor_kind()
{
this->q.clear(); this->r = NULL;
```

```
this->t.clear(); this->y.clear();
this->u.clear(); this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
cessor_kind* ptr = dynamic_cast<cessor_kind*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->w = ptr->w;
this->e = ptr->e;
}
cessor_kind(cessor_kind&& other)  noexcept
: q{ other.q }, e{ other.e }, t{ other.t }, u{ other.u }, w{ other.w },
r{ other.r }, y{ other.y } {}
void operator=(cessor_kind&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(t, other.t);
std::swap(u, other.u); std::swap(w, other.w); std::swap(r, other.r);
std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};//page 34

class old_schedule : public GeneralFact
{
string q{};
public:
old_schedule(const string q1)
: q(q1){}

old_schedule(const old_schedule& other)
{
if (this != &other)
{
this->q = other.q;
}
}
```

```
~old_schedule()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
old_schedule* ptr = dynamic_cast<old_schedule*>(&other);
this->q = ptr->q;
}

old_schedule(old_schedule&& other)  noexcept
: q{ other.q } {}

void operator=(old_schedule&& other)  noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class new_schedule : public GeneralFact
{
string q{};
public:
new_schedule(const string q1)
: q(q1) {}

new_schedule(const new_schedule& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~new_schedule()
{
this->q.clear();
}
```

```
void operator=(GeneralFact& other) override
{
new_schedule* ptr = dynamic_cast<new_schedule*>(&other);
this->q = ptr->q;
}

new_schedule(new_schedule&& other)  noexcept
: q{ other.q } {}

void operator=(new_schedule&& other)  noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class  local_ifthen_chain_end_operations_were_written : public GeneralFact
{
int q{};
public:
local_ifthen_chain_end_operations_were_written(const int q1)
: q(q1) {}
local_ifthen_chain_end_operations_were_written(
const local_ifthen_chain_end_operations_were_written& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~local_ifthen_chain_end_operations_were_written()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
local_ifthen_chain_end_operations_were_written* ptr =
dynamic_cast<local_ifthen_chain_end_operations_were_written*>(&other);
this->q = ptr->q;
```

```
}

local_ifthen_chain_end_operations_were_written
(local_ifthen_chain_end_operations_were_written&& other)  noexcept
: q{ other.q } {}

void operator=(local_ifthen_chain_end_operations_were_written&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class calls_list : public GeneralFact
{
int w{}, e{};
string q{};
vector<int>r{};
public:
calls_list(const string q1,const int w1, const int e1, const vector<int> r1)
: q(q1), w(w1), e(e1), r(r1) {
r.shrink_to_fit();
}

calls_list(const calls_list& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~calls_list()
{
this->q .clear();this->w = NULL;
this->e = NULL;  this->r.clear();
}
void operator=(GeneralFact& other) override
```

```
{
calls_list* ptr = dynamic_cast<calls_list*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

calls_list(calls_list&& other)  noexcept
: q{ other.q }, w{ other.w },
  e{ other.e }, r{ other.r } {}

void operator=(calls_list&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class composites_list : public GeneralFact
{
int w{};
string q{};
vector<int>e{};
public:
composites_list(const string q1, const int w1, const vector<int> e1)
: q(q1), w(w1), e(e1) {
e.shrink_to_fit();
}

composites_list(const composites_list& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~composites_list()
```

```
{
this->q.clear();this->w = NULL;
this->e.clear();
}
void operator=(GeneralFact& other) override
{
composites_list* ptr = dynamic_cast<composites_list*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

composites_list(composites_list&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(composites_list&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class record_aggregates_list : public GeneralFact
{
int w{};
string q{};
vector<int>e{};
public:
record_aggregates_list(const string q1, const int w1, const vector<int> e1)
: q(q1), w(w1), e(e1) {
e.shrink_to_fit();
}

record_aggregates_list(const record_aggregates_list& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
```

```
this->e = other.e;
}
}
~record_aggregates_list()
{
this->q.clear(); this->w = NULL;
this->e.clear();
}
void operator=(GeneralFact& other) override
{
record_aggregates_list* ptr = dynamic_cast<record_aggregates_list*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

record_aggregates_list(record_aggregates_list&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(record_aggregates_list&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class mem_port : public GeneralFact
{
int q{}, y{}, i{}, s{};
string w{}, e{}, r{}, t{}, u{}, o{}, p{}, a{}, d{};
public:
mem_port(const int q1, const string w1, const string e1, const string r1,
const string t1, const int y1, const string u1, const int i1, const string o1,
const string p1, const string a1, const int s1, const string d1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1),
a(a1), s(s1), d(d1) {}

mem_port(const mem_port& other)
{
```

```
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
this->a = other.a; this->s = other.s;
this->d = other.d;
}
}
~mem_port()
{
this->q = NULL; this->w.clear();
this->e.clear();this->r.clear();
this->t.clear();this->y = NULL;
this->u.clear();this->i = NULL;
this->o.clear();this->p.clear();
this->a.clear();this->s = NULL;
this->d.clear();

}
void operator=(GeneralFact& other) override
{
mem_port* ptr = dynamic_cast<mem_port*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
this->a = ptr->a; this->s = ptr->s;
this->d = ptr->d;
}

mem_port(mem_port&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p }, a{ other.a }, s{ other.s },
d{ other.d } {}

void operator=(mem_port&& other)  noexcept
```

```
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
std::swap(a, other.a); std::swap(s, other.s);
std::swap(d, other.d);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class global_declarations :public GeneralFact
{
int w{};
vector<local_object> q{};
public:
global_declarations(const vector<local_object> q1, const int w1)
: q(q1), w(w1) {
q.shrink_to_fit();
}

global_declarations(const global_declarations& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~global_declarations()
{
this->q.clear();this->w = NULL;

}
void operator=(GeneralFact& other) override
{
global_declarations* ptr = dynamic_cast<global_declarations*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
}

global_declarations(global_declarations&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(global_declarations&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}

friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class source_is_normal_dt : public GeneralFact
{
int w{}, e{};
string q{};
public:
source_is_normal_dt(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

source_is_normal_dt(const source_is_normal_dt& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~source_is_normal_dt()
{
this->q.clear();this->w = NULL;
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
source_is_normal_dt* ptr = dynamic_cast<source_is_normal_dt*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
this->e = ptr->e;
}

source_is_normal_dt(source_is_normal_dt&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(source_is_normal_dt&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class combo : public GeneralFact
{
int q{}, e{};
string w{};
public:
combo(const int q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}

combo(const combo& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~combo()
{
this->q = NULL; this->w.clear();
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
combo* ptr = dynamic_cast<combo*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
this->e = ptr->e;
}

combo(combo&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(combo&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class sequence : public GeneralFact
{
int q{}, e{};
string w{};
public:
sequence(const int q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}

sequence(const sequence& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~sequence()
{
this->q = NULL; this->w.clear();
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
sequence* ptr = dynamic_cast<sequence*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
this->e = ptr->e;
}

sequence(sequence&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(sequence&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class for_loop : public GeneralFact
{
int q{}, e{}, r{}, t{}, y{}, u{}, i{}, o{}, p{}, a{}, s{}, d{};
string w{};
public:
for_loop(const int q1, const string w1, const int e1, const int r1,
const int t1, const int y1, const int u1, const int i1, const int o1,
const int p1, const int a1, const int s1, const int d1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1),
p(p1), a(a1), s(s1), d(d1) {}

for_loop(const for_loop& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
this->a = other.a; this->s = other.s;
this->d = other.d;
}
}
~for_loop()
```

```
{
this->q = NULL; this->w.clear();
this->e = NULL; this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL; this->i = NULL;
this->o = NULL; this->p = NULL;
this->a = NULL; this->s = NULL;
this->d = NULL;

}
void operator=(GeneralFact& other) override
{
for_loop* ptr = dynamic_cast<for_loop*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
this->a = ptr->a; this->s = ptr->s;
this->d = ptr->d;
}

for_loop(for_loop&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p }, a{ other.a }, s{ other.s },
d{ other.d } {}

void operator=(for_loop&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
std::swap(a, other.a); std::swap(s, other.s);
std::swap(d, other.d);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
```

```
};

class last_for_loop_entry : public GeneralFact
{
int q{};
public:
last_for_loop_entry(const int q1)
: q(q1) {}

last_for_loop_entry(const last_for_loop_entry& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~last_for_loop_entry()
{
this->q = NULL;

}
void operator=(GeneralFact& other) override
{
last_for_loop_entry* ptr = dynamic_cast<last_for_loop_entry*>(&other);
this->q = ptr->q;
}

last_for_loop_entry(last_for_loop_entry&& other)  noexcept
: q{ other.q } {}

void operator=(last_for_loop_entry&& other)  noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class while_loop : public GeneralFact
{
```

```
int q{}, e{}, r{}, t{}, y{}, u{};
string w{};
public:
while_loop(const int q1, const string w1, const int e1, const int r1,
const int t1, const int y1,
const int u1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1) {}

while_loop(const while_loop& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u;
}
}
~while_loop()
{
this->q = NULL; this->w.clear();
this->e = NULL; this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL;
}
void operator=(GeneralFact& other) override
{
while_loop* ptr = dynamic_cast<while_loop*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u;
}

while_loop(while_loop&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u } {}

void operator=(while_loop&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
```

```
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_while_loop_entry : public GeneralFact
{
int q{};
public:
last_while_loop_entry(const int q1)
: q(q1) {}

last_while_loop_entry(const last_while_loop_entry& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~last_while_loop_entry()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
last_while_loop_entry* ptr = dynamic_cast<last_while_loop_entry*>(&other);
this->q = ptr->q;
}

last_while_loop_entry(last_while_loop_entry&& other)  noexcept
: q{ other.q } {}

void operator=(last_while_loop_entry&& other)  noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);
```

```
friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class possible_end_if : public GeneralFact
{
string q{};
vector<int>w{};
public:
possible_end_if(const string q1, const vector<int> w1)
: q(q1), w(w1) {
w.shrink_to_fit();
}

possible_end_if(const possible_end_if& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~possible_end_if()
{
this->q.clear();this->w.clear();
}
void operator=(GeneralFact& other) override
{
possible_end_if* ptr = dynamic_cast<possible_end_if*>(&other);
this->q = ptr->q; this->w = ptr->w;
}

possible_end_if(possible_end_if&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(possible_end_if&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);
```

```
friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class end_if : public GeneralFact
{
string q{};
vector<int>w{};
public:
end_if(const string q1, const vector<int> w1)
: q(q1), w(w1) {
w.shrink_to_fit();
}
end_if(const end_if& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~end_if()
{
this->q.clear(); this->w.clear();
}
void operator=(GeneralFact& other) override
{
end_if* ptr = dynamic_cast<end_if*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

end_if(end_if&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(end_if&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
```

```
};

//vector
class nested_cond_fact : public GeneralFact
{
string q{};
vector<int>w{};
public:
nested_cond_fact(const string q1, const vector<int> w1)
: q(q1), w(w1) {
w.shrink_to_fit();
}
nested_cond_fact(const nested_cond_fact& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~nested_cond_fact()
{
this->q.clear(); this->w.clear();
}
void operator=(GeneralFact& other) override
{
nested_cond_fact* ptr = dynamic_cast<nested_cond_fact*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

nested_cond_fact(nested_cond_fact&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(nested_cond_fact&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

```
class top_level_call : public GeneralFact
{
int q{}, w{}, r{}, y{}, i{}, p{};
string e{}, t{}, u{}, o{}, a{};
public:
top_level_call(const int q1, const int w1, const string e1, const int r1,
const string t1, const int y1,const string u1, const int i1, const string o1,
const int p1, const string a1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1), a(a1) {}

top_level_call(const top_level_call& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
this->a = other.a;
}
}
~top_level_call()
{
this->q = NULL; this->w = NULL;
this->e.clear();this->r = NULL;
this->t.clear();this->y = NULL;
this->u.clear();this->i = NULL;
this->o.clear();this->p = NULL;
this->a.clear();

}
void operator=(GeneralFact& other) override
{
top_level_call* ptr = dynamic_cast<top_level_call*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
```

```
this->a = ptr->a;
}

top_level_call(top_level_call&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p }, a{ other.a } {}

void operator=(top_level_call&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
std::swap(a, other.a);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class top_level_call_parcs : public GeneralFact
{
int q{}, w{}, r{}, y{}, i{}, p{};
string e{}, t{}, u{}, o{}, a{};
public:
top_level_call_parcs(const int q1, const int w1, const string e1,
const int r1, const string t1, const int y1, const string u1,
const int i1, const string o1, const int p1 , const string a1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1), a(a1) {}

top_level_call_parcs(const top_level_call_parcs& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
```

```
this->a = other.a;
}
}
~top_level_call_parcs()
{
this->q = NULL; this->w = NULL;
this->e.clear(); this->r = NULL;
this->t.clear(); this->y = NULL;
this->u.clear(); this->i = NULL;
this->o.clear(); this->p = NULL;
this->a.clear();
}
void operator=(GeneralFact& other) override
{
top_level_call_parcs* ptr = dynamic_cast<top_level_call_parcs*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
}

top_level_call_parcs(top_level_call_parcs&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p }, a{ other.a } {}

void operator=(top_level_call_parcs&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
std::swap(a, other.a);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

```
class added_aux_call_ios : public GeneralFact
{
int w{};
string q{};
public:
added_aux_call_ios(const string q1, const int w1)
: q(q1), w(w1) {}

added_aux_call_ios(const added_aux_call_ios& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~added_aux_call_ios()
{
this->q.clear();this->w = NULL;
}
void operator=(GeneralFact& other) override
{
added_aux_call_ios* ptr = dynamic_cast<added_aux_call_ios*>(&other);
this->q = ptr->q; this->w = ptr->w;
}

added_aux_call_ios(added_aux_call_ios&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(added_aux_call_ios&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class added_aux_call_ios1 : public GeneralFact
{
int w{}, e{};
string q{};
```

```
public:
added_aux_call_ios1(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}
added_aux_call_ios1(const added_aux_call_ios1& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~added_aux_call_ios1()
{
this->q.clear();this->w = NULL;
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
added_aux_call_ios1* ptr = dynamic_cast<added_aux_call_ios1*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;

}
added_aux_call_ios1(added_aux_call_ios1&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(added_aux_call_ios1&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class added_aux_call_signals : public GeneralFact
{
int w{};
string q{};
public:
```

```
added_aux_call_signals(const string q1, const int w1)
: q(q1), w(w1) {}

added_aux_call_signals(const added_aux_call_signals& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~added_aux_call_signals()
{
this->q.clear();this->w = NULL;
}
void operator=(GeneralFact& other) override
{
added_aux_call_signals* ptr = dynamic_cast<added_aux_call_signals*>(&other);
this->q = ptr->q; this->w = ptr->w;

}
added_aux_call_signals(added_aux_call_signals&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(added_aux_call_signals&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class found_call_operator : public GeneralFact
{
int w{};
string q{};
public:
found_call_operator(const string q1, const int w1)
: q(q1), w(w1) {}

found_call_operator(const found_call_operator& other)
```

```
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~found_call_operator()
{
this->q.clear(); this->w = NULL;
}
void operator=(GeneralFact& other) override
{
found_call_operator* ptr = dynamic_cast<found_call_operator*>(&other);
this->q = ptr->q; this->w = ptr->w;

}
found_call_operator(found_call_operator&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(found_call_operator&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class added_verilog_aux_call_outputs : public GeneralFact
{
int w{};
string q{}, e{};
public:
added_verilog_aux_call_outputs(const string q1, const int w1, const string e1)
: q(q1), w(w1), e(e1) {}

added_verilog_aux_call_outputs(const added_verilog_aux_call_outputs& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
```

```
this->e = other.e;
}
}
~added_verilog_aux_call_outputs()
{
this->q.clear(); this->w = NULL;
this->e.clear();
}
void operator=(GeneralFact& other) override
{
added_verilog_aux_call_outputs* ptr =
dynamic_cast<added_verilog_aux_call_outputs*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

added_verilog_aux_call_outputs(added_verilog_aux_call_outputs&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(added_verilog_aux_call_outputs&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class raw_dependencies : public GeneralFact
{
int w{};
string q{};
vector<int>e{}, r{};
public:
raw_dependencies(const string q1, const int w1, const vector<int> e1,
const vector<int> r1)
: q(q1), w(w1), e(e1), r(r1) {
e.shrink_to_fit();
r.shrink_to_fit();
```

```
}
raw_dependencies(const raw_dependencies& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~raw_dependencies()
{
this->q.clear();this->w = NULL;
this->e.clear();this->r.clear();
}
void operator=(GeneralFact& other) override
{
raw_dependencies* ptr = dynamic_cast<raw_dependencies*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

raw_dependencies(raw_dependencies&& other)  noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(raw_dependencies&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class war_dependencies : public GeneralFact
{
int w{};
string q{};
vector<int>e{}, r{};
```

```
public:
war_dependencies(const string q1, const int w1, const vector<int> e1,
const vector<int> r1)
: q(q1), w(w1), e(e1), r(r1) {
e.shrink_to_fit();
r.shrink_to_fit();
}
war_dependencies(const war_dependencies& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~war_dependencies()
{
this->q.clear(); this->w = NULL;
this->e.clear(); this->r.clear();
}
void operator=(GeneralFact& other) override
{
war_dependencies* ptr = dynamic_cast<war_dependencies*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

war_dependencies(war_dependencies&& other)  noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(war_dependencies&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

```
//vector
class waw_dependencies : public GeneralFact
{
int w{};
string q{};
vector<int>e{}, r{};
public:
waw_dependencies(const string q1, const int w1, const vector<int> e1,
const vector<int> r1)
: q(q1), w(w1), e(e1), r(r1) {
e.shrink_to_fit();
r.shrink_to_fit();
}
waw_dependencies(const waw_dependencies& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~waw_dependencies()
{
this->q.clear(); this->w = NULL;
this->e.clear(); this->r.clear();
}
void operator=(GeneralFact& other) override
{
waw_dependencies* ptr = dynamic_cast<waw_dependencies*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

waw_dependencies(waw_dependencies&& other)  noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(waw_dependencies&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
```

```
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class schedule : public GeneralFact
{
int w{}, r{};
string q{}, e{};
public:
schedule(const string q1, const int w1, const string e1, const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

schedule(const schedule& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~schedule()
{
this->q.clear();this->w = NULL;
this->e.clear();this->r = NULL;

}
void operator=(GeneralFact& other) override
{
schedule* ptr = dynamic_cast<schedule*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;  this->r = ptr->r;
}

schedule(schedule&& other)  noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(schedule&& other)  noexcept
{
```

```
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_conditional_execution : public GeneralFact
{
int w{};
string q{};
public:
last_conditional_execution(const string q1, const int w1)
: q(q1), w(w1) {}

last_conditional_execution(const last_conditional_execution& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~last_conditional_execution()
{
this->q.clear();this->w = NULL;

}
void operator=(GeneralFact& other) override
{
last_conditional_execution* ptr =
dynamic_cast<last_conditional_execution*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

last_conditional_execution(last_conditional_execution&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(last_conditional_execution&& other)  noexcept
{
```

```
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class conditional_operations : public GeneralFact
{
int e{}, r{};
string q{}, w{};
vector<int>t{}, y{}, u{}, i{};
public:
conditional_operations(const string q1, const string w1, const int e1,
const int r1, const vector<int> t1, const vector<int> y1,const vector<int> u1,
const vector<int> i1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1) {
t.shrink_to_fit();
y.shrink_to_fit();
u.shrink_to_fit();
i.shrink_to_fit();
}
conditional_operations(const conditional_operations& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
}
}
~conditional_operations()
{
this->q.clear(); this->w.clear();
this->e = NULL; this->r = NULL;
this->t.clear(); this->y.clear();
this->u.clear(); this->i.clear();

}
```

```
void operator=(GeneralFact& other) override
{
conditional_operations* ptr = dynamic_cast<conditional_operations*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
}

conditional_operations(conditional_operations&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i }{}

void operator=(conditional_operations&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_conditional_transition_of_schedule : public GeneralFact
{
int e{};
string q{}, w{};
public:
last_conditional_transition_of_schedule(const string q1, const string w1,
const int e1)
: q(q1), w(w1), e(e1) {}

last_conditional_transition_of_schedule(
const last_conditional_transition_of_schedule& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
```

```
}
}
~last_conditional_transition_of_schedule()
{
this->q.clear();this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
last_conditional_transition_of_schedule* ptr =
dynamic_cast<last_conditional_transition_of_schedule*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

last_conditional_transition_of_schedule(
last_conditional_transition_of_schedule&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(last_conditional_transition_of_schedule&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class transition_to_be_rescheduled : public GeneralFact
{
int e{}, t{};
string q{}, w{}, r{};
public:
transition_to_be_rescheduled(const string w1, const int e1, const string q1,
const string r1, const int t1)
:w(w1), e(e1), q(q1), r(r1), t(t1) {}

transition_to_be_rescheduled(const transition_to_be_rescheduled& other)
{
```

```
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;
}
}
~transition_to_be_rescheduled()
{
this->q.clear(); this->r.clear();
this->w.clear(); this->e = NULL;
this->t = NULL;

}
void operator=(GeneralFact& other) override
{
transition_to_be_rescheduled* ptr =
dynamic_cast<transition_to_be_rescheduled*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;
}
transition_to_be_rescheduled(transition_to_be_rescheduled&& other) noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{other.t} {}
void operator=(transition_to_be_rescheduled&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_conditional_transition : public GeneralFact
{
int w{};
string q{};
public:
last_conditional_transition(const string q1, const int w1)
: q(q1), w(w1) {}
```

```
last_conditional_transition(const last_conditional_transition& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~last_conditional_transition()
{
this->q.clear();this->w = NULL;

}
void operator=(GeneralFact& other) override
{
last_conditional_transition* ptr =
dynamic_cast<last_conditional_transition*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

last_conditional_transition(last_conditional_transition&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(last_conditional_transition&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class conditional_transitions : public GeneralFact
{
int e{}, r{}, t{}, y{}, u{}, i{};
string q{}, w{};
public:
conditional_transitions(const string q1, const string w1, const int e1,
const int r1, const int t1, const int y1,
const int u1, const int i1)
```

```
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1) {}

conditional_transitions(const conditional_transitions& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
}
}
~conditional_transitions()
{
this->q.clear(); this->w.clear();
this->e = NULL; this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL; this->i = NULL;

}
void operator=(GeneralFact& other) override
{
conditional_transitions* ptr = dynamic_cast<conditional_transitions*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
}

conditional_transitions(conditional_transitions&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i } {}

void operator=(conditional_transitions&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
}
friend string makeStringOf(GeneralFact* obj);
```

```
friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class state : public GeneralFact
{
int e{}, t{}, y{};
string q{}, w{}, r{};
vector<int>u{}, i{};
public:
state(const string q1, const string w1, const int e1, const string r1, const int t1,
const int y1, const vector<int> u1, const vector<int> i1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1) {
u.shrink_to_fit();
i.shrink_to_fit();
}

state(const state& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
}
}
~state()
{
this->q.clear(); this->w.clear();
this->e = NULL; this->r.clear();
this->t = NULL; this->y = NULL;
this->u.clear(); this->i.clear();

}
void operator=(GeneralFact& other) override
{
state* ptr = dynamic_cast<state*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
```

```
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
}

state(state&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i } {}

void operator=(state&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class rescheduled : public GeneralFact
{
int e{}, t{};
string q{}, w{}, r{};
public:
rescheduled(const string w1, const int e1, const string q1, const string r1,
const int t1)
:w(w1), e(e1), q(q1), r(r1), t(t1) {}
rescheduled(const rescheduled& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;
}
}
~rescheduled()
{
this->q.clear(); this->r.clear();
this->w.clear(); this->e = NULL;
```

```
this->t = NULL;

}
void operator=(GeneralFact& other) override
{
rescheduled* ptr = dynamic_cast<rescheduled*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;
}
rescheduled(rescheduled&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{ other.t } {}
void operator=(rescheduled&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_rescheduled : public GeneralFact
{
int e{}, t{};
string q{}, w{}, r{};
public:
last_rescheduled(const string w1, const int e1, const string q1, const string r1,
const int t1)
:w(w1), e(e1), q(q1), r(r1), t(t1) {}
last_rescheduled(const last_rescheduled& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;
}
}
~last_rescheduled()
{
```

```
this->q.clear(); this->r.clear();
this->w.clear(); this->e = NULL;
this->t = NULL;

}
void operator=(GeneralFact& other) override
{
last_rescheduled* ptr = dynamic_cast<last_rescheduled*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;
}
last_rescheduled(last_rescheduled&& other)  noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{ other.t } {}
void operator=(last_rescheduled&& other)  noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class raw_cessor : public GeneralFact
{
int w{}, e{};
string q{};
public:
raw_cessor(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

raw_cessor(const raw_cessor& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~raw_cessor()
```

```
{
this->q.clear(); this->w = NULL;
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
raw_cessor* ptr = dynamic_cast<raw_cessor*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

raw_cessor(raw_cessor&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(raw_cessor&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class war_cessor : public GeneralFact
{
int w{}, e{};
string q{};
public:
war_cessor(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

war_cessor(const war_cessor& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~war_cessor()
```

```
{
this->q.clear(); this->w = NULL;
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
war_cessor* ptr = dynamic_cast<war_cessor*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

war_cessor(war_cessor&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(war_cessor&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class waw_cessor : public GeneralFact
{
int w{}, e{};
string q{};
public:
waw_cessor(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

waw_cessor(const waw_cessor& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~waw_cessor()
```

```
{
this->q.clear(); this->w = NULL;
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
waw_cessor* ptr = dynamic_cast<waw_cessor*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

waw_cessor(waw_cessor&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(waw_cessor&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class op_resource : public GeneralFact
{
int e{};
string q{}, w{};
public:
op_resource(const string q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}
op_resource(const op_resource& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~op_resource()
{
```

```
this->q.clear(); this->w.clear();
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
op_resource* ptr = dynamic_cast<op_resource*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

op_resource(op_resource&& other)  noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(op_resource&& other)  noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class global_resource : public GeneralFact
{
int q{};
public:
global_resource(const int q1)
: q(q1) {}

global_resource(const global_resource& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~global_resource()
{
this->q = NULL;
}
```

```
void operator=(GeneralFact& other) override
{
global_resource* ptr = dynamic_cast<global_resource*>(&other);
this->q = ptr->q;
}

global_resource(global_resource&& other)  noexcept
: q{ other.q } {}

void operator=(global_resource&& other)  noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class module_g_resource : public GeneralFact
{
int w{};
string q{};
public:
module_g_resource(const string q1, const int w1)
: q(q1), w(w1) {}

module_g_resource(const module_g_resource& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~module_g_resource()
{
this->q.clear(); this->w = NULL;
}
void operator=(GeneralFact& other) override
{
module_g_resource* ptr = dynamic_cast<module_g_resource*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
}

module_g_resource(module_g_resource&& other)  noexcept
: q{ other.q }, w{ other.w } {}

void operator=(module_g_resource&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class cf_previous_op : public GeneralFact
{
int w{}, e{}, r{};
string q{};
public:
cf_previous_op(const string q1, const int w1, const int e1, const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

cf_previous_op(const cf_previous_op& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~cf_previous_op()
{
this->q.clear();this->w = NULL;
this->e = NULL; this->r = NULL;

}
void operator=(GeneralFact& other) override
{
cf_previous_op* ptr = dynamic_cast<cf_previous_op*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
this->e = ptr->e; this->r = ptr->r;
}

cf_previous_op(cf_previous_op&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(cf_previous_op&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class cf_previous_state : public GeneralFact
{
int w{}, e{}, r{};
string q{};
public:
cf_previous_state(const string q1, const int w1, const int e1, const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

cf_previous_state(const cf_previous_state& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~cf_previous_state()
{
this->q.clear(); this->w = NULL;
this->e = NULL; this->r = NULL;
}
void operator=(GeneralFact& other) override
{
cf_previous_state* ptr = dynamic_cast<cf_previous_state*>(&other);
```

```
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;  this->r = ptr->r;
}

cf_previous_state(cf_previous_state&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(cf_previous_state&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class pred_candidate_examined : public GeneralFact
{
int w{}, e{};
string q{};
public:
pred_candidate_examined(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

pred_candidate_examined(const pred_candidate_examined& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~pred_candidate_examined()
{
this->q.clear();this->w = NULL;
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
```

```
pred_candidate_examined* ptr = dynamic_cast<pred_candidate_examined*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

pred_candidate_examined(pred_candidate_examined&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(pred_candidate_examined&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class reentrant_triangle : public GeneralFact
{
int w{}, e{}, r{}, a{}, s{};
string q{}, d{}, f{};
vector<int>t{}, y{}, u{}, i{}, o{}, p{};
public:
reentrant_triangle(const string q1, const int w1, const int e1, const int r1,
const vector<int> t1, const vector<int> y1,const vector<int> u1,
const vector<int> i1, const vector<int> o1, const vector<int> p1, const int a1,
const int s1, const string d1, const string f1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1), a(a1), s(s1),
d(d1), f(f1) {
t.shrink_to_fit();
y.shrink_to_fit();
u.shrink_to_fit();
i.shrink_to_fit();
o.shrink_to_fit();
p.shrink_to_fit();
}

reentrant_triangle(const reentrant_triangle& other)
{
```

```
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
this->a = other.a; this->s = other.s;
this->d = other.d; this->f = other.f;
}
}
~reentrant_triangle()
{
this->q.clear();this->w = NULL;
this->e = NULL; this->r = NULL;
this->t.clear();this->y.clear();
this->u.clear();this->i.clear();
this->o.clear();this->p.clear();
this->a = NULL; this->s = NULL;
this->d.clear();this->f.clear();

}
void operator=(GeneralFact& other) override
{
reentrant_triangle* ptr = dynamic_cast<reentrant_triangle*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
this->a = ptr->a; this->s = ptr->s;
this->d = ptr->d; this->f = ptr->f;
}

reentrant_triangle(reentrant_triangle&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p }, a{ other.a }, s{ other.s },
d{ other.d }, f{ other.f } {}

void operator=(reentrant_triangle&& other) noexcept
```

```
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
std::swap(a, other.a); std::swap(s, other.s);
std::swap(d, other.d);  std::swap(f, other.f);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_reentrant_triangle : public GeneralFact
{
int w{};
string q{};
public:
last_reentrant_triangle(const string q1, const int w1)
: q(q1), w(w1) {}

last_reentrant_triangle(const last_reentrant_triangle& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~last_reentrant_triangle()
{
this->q.clear();this->w = NULL;

}
void operator=(GeneralFact& other) override
{
last_reentrant_triangle* ptr = dynamic_cast<last_reentrant_triangle*>(&other);
this->q = ptr->q; this->w = ptr->w;

}
```

```
last_reentrant_triangle(last_reentrant_triangle&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(last_reentrant_triangle&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_schedule_state : public GeneralFact
{
int e{};
string q{}, w{};
public:
last_schedule_state(const string q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}

last_schedule_state(const last_schedule_state& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~last_schedule_state()
{
this->q.clear(); this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
last_schedule_state* ptr = dynamic_cast<last_schedule_state*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}
```

```
last_schedule_state(last_schedule_state&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(last_schedule_state&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class conditional_incomplete : public GeneralFact
{
int e{}, r{}, a{}, s{}, d{}, f{}, g{}, h{}, j{}, k{}, l{}, z{}, x{};
string q{}, w{};
vector<int>t{}, y{}, u{}, i{}, o{}, p{};
public:
conditional_incomplete(const string q1, const string w1, const int e1, const int r1,
const vector<int> t1, const vector<int> y1,const vector<int> u1,
const vector<int> i1, const vector<int> o1, const vector<int> p1, const int a1,
const int s1, const int d1, const int f1, const int g1, const int h1, const int j1,
const int k1, const int l1,const int z1, const int x1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1), a(a1), s(s1),
d(d1), f(f1), g(g1), h(h1),j(j1), k(k1), l(l1), z(z1), x(x1) {
t.shrink_to_fit();
y.shrink_to_fit();
u.shrink_to_fit();
i.shrink_to_fit();
o.shrink_to_fit();
p.shrink_to_fit();
}

conditional_incomplete(const conditional_incomplete& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
```

```
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
this->a = other.a; this->s = other.s;
this->d = other.d; this->f = other.f;
this->g = other.g; this->h = other.h;
this->j = other.j; this->k = other.k;
this->l = other.l; this->z = other.z;
this->x = other.x;
}
}
~conditional_incomplete()
{
this->q.clear();this->w.clear();
this->e = NULL; this->r = NULL;
this->t.clear();this->y.clear();
this->u.clear();this->i.clear();
this->o.clear();this->p.clear();
this->a = NULL; this->s = NULL;
this->d = NULL; this->f = NULL;
this->g = NULL; this->h = NULL;
this->j = NULL; this->k = NULL;
this->l = NULL; this->z = NULL;
this->x = NULL;
}
void operator=(GeneralFact& other) override
{
conditional_incomplete* ptr = dynamic_cast<conditional_incomplete*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
this->a = ptr->a; this->s = ptr->s;
this->d = ptr->d; this->f = ptr->f;
this->g = ptr->g; this->h = ptr->h;
this->j = ptr->j; this->k = ptr->k;
this->l = ptr->l; this->z = ptr->z;
this->x = ptr->x;
}
```

```
conditional_incomplete(conditional_incomplete&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p }, a{ other.a }, s{ other.s },
d{ other.d }, f{ other.f }, g{ other.g }, h{ other.h },
j{ other.j }, k{ other.k }, l{ other.l }, z{ other.z },
x{ other.x }{}

void operator=(conditional_incomplete&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
std::swap(a, other.a); std::swap(s, other.s);
std::swap(d, other.d); std::swap(f, other.f);
std::swap(g, other.g); std::swap(h, other.h);
std::swap(j, other.j); std::swap(k, other.k);
std::swap(l, other.l); std::swap(z, other.z);
std::swap(x, other.x);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class mixed_incomplete_state_lists : public GeneralFact
{
int e{};
string q{}, w{};
vector<int>r{}, t{};
public:
mixed_incomplete_state_lists(const string w1, const int e1, const string q1,
const vector<int> r1,
const vector<int> t1)
:w(w1), e(e1), q(q1), r(r1), t(t1) {
r.shrink_to_fit();
t.shrink_to_fit();
}
```

```
mixed_incomplete_state_lists(const mixed_incomplete_state_lists& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;
}
}
~mixed_incomplete_state_lists()
{
this->q.clear(); this->r.clear();
this->w.clear(); this->e = NULL;
this->t.clear();
}
void operator=(GeneralFact& other) override
{
mixed_incomplete_state_lists* ptr =
dynamic_cast<mixed_incomplete_state_lists*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;
}
mixed_incomplete_state_lists(mixed_incomplete_state_lists&& other) noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{ other.t } {}
void operator=(mixed_incomplete_state_lists&& other) noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class linear_incomplete_node : public GeneralFact
{
int e{}, y{};
string q{}, w{};
vector<int>r{}, t{};
```

```
public:
linear_incomplete_node(const string w1, const int e1, const string q1, const int y1,
const vector<int> r1, const vector<int> t1)
:w(w1), e(e1), q(q1), y(y1), r(r1), t(t1) {
r.shrink_to_fit();
t.shrink_to_fit();
}
linear_incomplete_node(const linear_incomplete_node& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;  this->y = other.y;
}
}
~linear_incomplete_node()
{
this->q.clear(); this->r.clear();
this->w.clear(); this->e = NULL;
this->t.clear(); this->y = NULL;

}
void operator=(GeneralFact& other) override
{
linear_incomplete_node* ptr = dynamic_cast<linear_incomplete_node*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;   this->y = ptr->y;
}
linear_incomplete_node(linear_incomplete_node&& other) noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{ other.t } {}
void operator=(linear_incomplete_node&& other) noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

```
class incomplete_links : public GeneralFact
{
int q{}, w{}, e{}, r{}, t{}, y{}, u{}, i{}, o{}, p{};
public:
incomplete_links(const int q1, const int w1, const int e1, const int r1,
const int t1, const int y1, const int u1, const int i1, const int o1, const int p1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1) {}

incomplete_links(const incomplete_links& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
}
}
~incomplete_links()
{
this->q = NULL; this->w = NULL;
this->e = NULL; this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL; this->i = NULL;
this->o = NULL; this->p = NULL;

}
void operator=(GeneralFact& other) override
{
incomplete_links* ptr = dynamic_cast<incomplete_links*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
}

incomplete_links(incomplete_links&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
```

```
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p } {}

void operator=(incomplete_links&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_incomplete : public GeneralFact
{
int q{}, w{}, e{};
string r{};
public:
last_incomplete(const int q1, const int w1, const int e1, const string r1)
: q(q1), w(w1), e(e1), r(r1) {}
last_incomplete(const last_incomplete& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~last_incomplete()
{
this->q = NULL; this->w = NULL;
this->e = NULL;  this->r.clear();

}
void operator=(GeneralFact& other) override
{
last_incomplete* ptr = dynamic_cast<last_incomplete*>(&other);
this->q = ptr->q; this->w = ptr->w;
```

```
this->e = ptr->e;  this->r = ptr->r;
}

last_incomplete(last_incomplete&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(last_incomplete&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class nil_node
{
public:
int q{};
string w{};

nil_node(const int q1, const string w1)
: q(q1), w(w1) {}
nil_node(const nil_node& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~nil_node()
{
this->q = NULL; this->w.clear();
}
void operator=(nil_node& other)
{
this->q = other.q; this->w = other.w;
}
```

```
nil_node(nil_node&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(nil_node&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
bool operator==(nil_node other)
{
if (this->q == other.q && this->w == other.w)return true;
else return false;
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class global_nils : public GeneralFact
{
vector<nil_node>q{};
public:
global_nils(const vector<nil_node> q1)
: q(q1) {
q.shrink_to_fit();
}

global_nils(const global_nils& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~global_nils()
{
this->q.clear();

}
void operator=(GeneralFact& other) override
{
```

```
global_nils* ptr = dynamic_cast<global_nils*>(&other);
this->q = ptr->q;
}

global_nils(global_nils&& other) noexcept
: q{ other.q } {}

void operator=(global_nils&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class current_module : public GeneralFact
{
string q{};
public:
current_module(const string q1)
: q(q1) {}

current_module(const current_module& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~current_module()
{
this->q.clear();

}
void operator=(GeneralFact& other) override
{
current_module* ptr = dynamic_cast<current_module*>(&other);
this->q = ptr->q;
}
```

```
current_module(current_module&& other) noexcept
: q{ other.q } {}

void operator=(current_module&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_linear_incomplete_node : public GeneralFact
{
int q{};
public:
last_linear_incomplete_node(const int q1)
: q(q1) {}

last_linear_incomplete_node(const last_linear_incomplete_node& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~last_linear_incomplete_node()
{
this->q = NULL;

}
void operator=(GeneralFact& other) override
{
last_linear_incomplete_node* ptr =
dynamic_cast<last_linear_incomplete_node*>(&other);
this->q = ptr->q;
}

last_linear_incomplete_node(last_linear_incomplete_node&& other) noexcept
: q{ other.q } {}
```

```
void operator=(last_linear_incomplete_node&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class operator_instances : public GeneralFact
{
int w{};
string q{};
public:
operator_instances(const string q1, const int w1)
: q(q1), w(w1) {}

operator_instances(const operator_instances& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~operator_instances()
{
this->q.clear(); this->w = NULL;

}
void operator=(GeneralFact& other) override
{
operator_instances* ptr = dynamic_cast<operator_instances*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

operator_instances(operator_instances&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(operator_instances&& other) noexcept
{
```

```
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class massively_parallel_style : public GeneralFact
{
int q{};
public:
massively_parallel_style(const int q1)
: q(q1) {}

massively_parallel_style(const massively_parallel_style& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~massively_parallel_style()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
massively_parallel_style* ptr = dynamic_cast<massively_parallel_style*>(&other);
this->q = ptr->q;
}

massively_parallel_style(massively_parallel_style&& other) noexcept
: q{ other.q } {}

void operator=(massively_parallel_style&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
```

```
};

class hdl_style : public GeneralFact
{
string q{};
public:
hdl_style(const string q1)
: q(q1) {}

hdl_style(const hdl_style& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~hdl_style()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
hdl_style* ptr = dynamic_cast<hdl_style*>(&other);
this->q = ptr->q;
}

hdl_style(hdl_style&& other) noexcept
: q{ other.q } {}

void operator=(hdl_style&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class op_instance : public GeneralFact
{
int q{}, e{}, r{};
```

```
string w{}, t{};
public:
op_instance(const string w1, const int e1, const int q1, const int r1,
const string t1)
:w(w1), e(e1), q(q1), r(r1), t(t1) {}
op_instance(const op_instance& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;
}
}
~op_instance()
{
this->q = NULL; this->r = NULL;
this->w.clear();this->e = NULL;
this->t.clear();

}
void operator=(GeneralFact& other) override
{
op_instance* ptr = dynamic_cast<op_instance*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;
}
op_instance(op_instance&& other) noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{ other.t } {}
void operator=(op_instance&& other) noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_op_instance : public GeneralFact
```

```
{
int q{}, e{};
string w{};
public:
last_op_instance(const int q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}

last_op_instance(const last_op_instance& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~last_op_instance()
{
this->q = NULL; this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
last_op_instance* ptr = dynamic_cast<last_op_instance*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

last_op_instance(last_op_instance&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(last_op_instance&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

```
class op_in_a_state : public GeneralFact
{
int q{}, r{};
string w{}, e{}, t{};
public:
op_in_a_state(const string w1, const string e1, const int q1, const int r1,
const string t1)
:w(w1), e(e1), q(q1), r(r1), t(t1) {}
op_in_a_state(const op_in_a_state& other)
{
if (this != &other)
{
this->q = other.q; this->r = other.r;
this->w = other.w; this->e = other.e;
this->t = other.t;
}
}
~op_in_a_state()
{
this->q = NULL; this->r = NULL;
this->w.clear();this->e.clear();
this->t.clear();
}
void operator=(GeneralFact& other) override
{
op_in_a_state* ptr = dynamic_cast<op_in_a_state*>(&other);
this->q = ptr->q; this->r = ptr->r;
this->w = ptr->w; this->e = ptr->e;
this->t = ptr->t;
}
op_in_a_state(op_in_a_state&& other) noexcept
: q{ other.q }, e{ other.e }, w{ other.w }, r{ other.r }, t{ other.t } {}
void operator=(op_in_a_state&& other) noexcept
{
std::swap(q, other.q); std::swap(e, other.e); std::swap(w, other.w);
std::swap(r, other.r), std::swap(t, other.t);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

```
class last_op_in_a_state : public GeneralFact
{
int q{}, r{};
string w{}, e{};
public:
last_op_in_a_state(const int q1, const string w1, const string e1, const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

last_op_in_a_state(const last_op_in_a_state& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~last_op_in_a_state()
{
this->q = NULL; this->w.clear();
this->e.clear();  this->r = NULL;

}
void operator=(GeneralFact& other) override
{
last_op_in_a_state* ptr = dynamic_cast<last_op_in_a_state*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;  this->r = ptr->r;
}

last_op_in_a_state(last_op_in_a_state&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(last_op_in_a_state&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);
```

```
friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class signal_instance : public GeneralFact
{
int q{}, t{}, y{}, u{};
string w{}, e{}, r{};
public:
signal_instance(const int q1, const string w1, const string e1, const string r1,
const int t1, const int y1, const int u1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1) {}

signal_instance(const signal_instance& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u;
}
}
~signal_instance()
{
this->q = NULL; this->w.clear();
this->e.clear();this->r.clear();
this->t = NULL; this->y = NULL;
this->u = NULL;

}
void operator=(GeneralFact& other) override
{
signal_instance* ptr = dynamic_cast<signal_instance*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u;
}

signal_instance(signal_instance&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
```

```
t{ other.t }, y{ other.y }, u{ other.u } {}

void operator=(signal_instance&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_signal_instance : public GeneralFact
{
int q{};
string w{};
public:
last_signal_instance(const int q1, const string w1)
: q(q1), w(w1) {}

last_signal_instance(const last_signal_instance& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~last_signal_instance()
{
this->q = NULL; this->w.clear();

}
void operator=(GeneralFact& other) override
{
last_signal_instance* ptr = dynamic_cast<last_signal_instance*>(&other);
this->q = ptr->q; this->w = ptr->w;

}
```

```
last_signal_instance(last_signal_instance&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(last_signal_instance&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class output_instance : public GeneralFact
{
int q{}, t{}, y{};
string w{}, e{}, r{};
public:
output_instance(const int q1, const string w1, const string e1, const string r1,
const int t1, const int y1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1) {}

output_instance(const output_instance& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
}
}
~output_instance()
{
this->q = NULL; this->w.clear();
this->e.clear();this->r.clear();
this->t = NULL; this->y = NULL;

}
void operator=(GeneralFact& other) override
{
output_instance* ptr = dynamic_cast<output_instance*> (&other);
this->q = ptr->q; this->w = ptr->w;
```

```
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
}

output_instance(output_instance&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y } {}

void operator=(output_instance&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_output_instance : public GeneralFact
{
int q{};
string w{};
public:
last_output_instance(const int q1, const string w1)
: q(q1), w(w1) {}

last_output_instance(const last_output_instance& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~last_output_instance()
{
this->q = NULL; this->w.clear();

}
void operator=(GeneralFact& other) override
{
```

```
last_output_instance* ptr = dynamic_cast<last_output_instance*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

last_output_instance(last_output_instance&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(last_output_instance&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class operator_instance_stats : public GeneralFact
{
int w{}, e{}, r{};
string q{};
public:
operator_instance_stats(const string q1, const int w1, const int e1, const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

operator_instance_stats(const operator_instance_stats& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~operator_instance_stats()
{
this->q.clear();this->w = NULL;
this->e = NULL; this->r = NULL;
}
void operator=(GeneralFact& other) override
{
operator_instance_stats* ptr = dynamic_cast<operator_instance_stats*>(&other);
```

```
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

operator_instance_stats(operator_instance_stats&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(operator_instance_stats&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class consecutive_106 : public GeneralFact
{
string q{};
public:
consecutive_106(const string q1)
: q(q1) {}

consecutive_106(const consecutive_106& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~consecutive_106()
{
this->q.clear();

}
void operator=(GeneralFact& other) override
{
consecutive_106* ptr = dynamic_cast<consecutive_106*>(&other);
this->q = ptr->q;
```

```
}

consecutive_106(consecutive_106&& other) noexcept
: q{ other.q } {}

void operator=(consecutive_106&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class operation_order : public GeneralFact
{
int e{}, r{}, t{}, y{}, u{}, i{}, o{}, p{};
string q{}, w{};
public:
operation_order(const string q1, const string w1, const int e1, const int r1,
const int t1, const int y1,const int u1, const int i1, const int o1, const int p1)
: q(q1), w(w1), e(e1), r(r1), t(t1), y(y1), u(u1), i(i1), o(o1), p(p1) {}

operation_order(const operation_order& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e; this->r = other.r;
this->t = other.t; this->y = other.y;
this->u = other.u; this->i = other.i;
this->o = other.o; this->p = other.p;
}
}
~operation_order()
{
this->q.clear();this->w.clear();
this->e = NULL; this->r = NULL;
this->t = NULL; this->y = NULL;
this->u = NULL; this->i = NULL;
this->o = NULL; this->p = NULL;
```

```
}
void operator=(GeneralFact& other) override
{
operation_order* ptr = dynamic_cast<operation_order*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
this->t = ptr->t; this->y = ptr->y;
this->u = ptr->u; this->i = ptr->i;
this->o = ptr->o; this->p = ptr->p;
}

operation_order(operation_order&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e }, r{ other.r },
t{ other.t }, y{ other.y }, u{ other.u }, i{ other.i },
o{ other.o }, p{ other.p } {}

void operator=(operation_order&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
std::swap(t, other.t); std::swap(y, other.y);
std::swap(u, other.u); std::swap(i, other.i);
std::swap(o, other.o); std::swap(p, other.p);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class max_parallel_call_order : public GeneralFact
{
int e{}, r{};
string q{}, w{};
public:
max_parallel_call_order(const string q1, const string w1, const int e1, const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

max_parallel_call_order(const max_parallel_call_order& other)
{
if (this != &other)
```

```
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~max_parallel_call_order()
{
this->q.clear(); this->w.clear();
this->e = NULL;  this->r = NULL;

}
void operator=(GeneralFact& other) override
{
max_parallel_call_order* ptr = dynamic_cast<max_parallel_call_order*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

max_parallel_call_order(max_parallel_call_order&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(max_parallel_call_order&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class max_op_order : public GeneralFact
{
int e{};
string q{}, w{};
public:
max_op_order(const string q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}

max_op_order(const max_op_order& other)
```

```
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~max_op_order()
{
this->q.clear(); this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
max_op_order* ptr = dynamic_cast<max_op_order*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

max_op_order(max_op_order&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(max_op_order&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class totalmax_call_order : public GeneralFact
{
int e{};
string q{}, w{};
public:
totalmax_call_order(const string q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}
```

```
totalmax_call_order(const totalmax_call_order& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~totalmax_call_order()
{
this->q.clear(); this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
totalmax_call_order* ptr = dynamic_cast<totalmax_call_order*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

totalmax_call_order(totalmax_call_order&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(totalmax_call_order&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class totalmax_gross_depth : public GeneralFact
{
int e{};
string q{}, w{};
public:
totalmax_gross_depth(const string q1, const string w1, const int e1)
: q(q1), w(w1), e(e1) {}
```

```
totalmax_gross_depth(const totalmax_gross_depth& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~totalmax_gross_depth()
{
this->q.clear(); this->w.clear();
this->e = NULL;

}
void operator=(GeneralFact& other) override
{
totalmax_gross_depth* ptr = dynamic_cast<totalmax_gross_depth*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

totalmax_gross_depth(totalmax_gross_depth&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(totalmax_gross_depth&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class current_total_max_order_entry : public GeneralFact
{
int q{};
public:
current_total_max_order_entry(const int q1)
: q(q1) {}
```

```
current_total_max_order_entry(const current_total_max_order_entry& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~current_total_max_order_entry()
{
this->q = NULL;

}
void operator=(GeneralFact& other) override
{
current_total_max_order_entry* ptr =
dynamic_cast<current_total_max_order_entry*>(&other);
this->q = ptr->q;
}

current_total_max_order_entry(current_total_max_order_entry&& other) noexcept
: q{ other.q } {}

void operator=(current_total_max_order_entry&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class module_last_state : public GeneralFact
{
int q{};
public:
module_last_state(const int q1)
: q(q1) {}

module_last_state(const module_last_state& other)
{
```

```
if (this != &other)
{
this->q = other.q;
}
}
~module_last_state()
{
this->q = NULL;

}
void operator=(GeneralFact& other) override
{
module_last_state* ptr = dynamic_cast<module_last_state*>(&other);
this->q = ptr->q;
}
module_last_state(module_last_state&& other) noexcept
: q{ other.q } {}
void operator=(module_last_state&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class module_local_list : public GeneralFact
{
vector<local_object>q{};
public:
module_local_list(vector<local_object> q1)
{
q = q1;
q.shrink_to_fit();
}

module_local_list(local_object& other)
{
module_local_list* ptr = dynamic_cast<module_local_list*>(&other);
if (this != ptr)
```

```
{
this->q = ptr->q;
}
}
~module_local_list()
{
this->q.clear();

}
void operator=(GeneralFact& other) override
{
module_local_list* ptr = dynamic_cast<module_local_list*>(&other);
this->q = ptr->q;
}

module_local_list(module_local_list&& other) noexcept
: q{ other.q }  {}

void operator=(module_local_list&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

//vector
class module_local_list_parcs : public GeneralFact
{
vector<local_object>q{};
public:
module_local_list_parcs(const vector<local_object> q1)
: q(q1) {
q.shrink_to_fit();
}

module_local_list_parcs(const module_local_list_parcs& other)
{
if (this != &other)
{
```

```
this->q = other.q;
}
}
~module_local_list_parcs()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
module_local_list_parcs* ptr = dynamic_cast<module_local_list_parcs*>(&other);
this->q = ptr->q;
}

module_local_list_parcs(module_local_list_parcs&& other) noexcept
: q{ other.q } {}

void operator=(module_local_list_parcs&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_non_io_found : public GeneralFact
{
int q{};
public:
last_non_io_found(const int q1)
: q(q1) {}

last_non_io_found(const last_non_io_found& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~last_non_io_found()
{
```

```
this->q = NULL;

}
void operator=(GeneralFact& other) override
{
last_non_io_found* ptr = dynamic_cast<last_non_io_found*>(&other);
this->q = ptr->q;
}

last_non_io_found(last_non_io_found&& other) noexcept
: q{ other.q } {}

void operator=(last_non_io_found&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class last_local_number : public GeneralFact
{
int q{};
public:
last_local_number(const int q1)
: q(q1) {}

last_local_number(const last_local_number& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~last_local_number()
{
this->q = NULL;

}
void operator=(GeneralFact& other) override
```

```
{
last_local_number* ptr = dynamic_cast<last_local_number*>(&other);
this->q = ptr->q;
}

last_local_number(last_local_number&& other) noexcept
: q{ other.q } {}

void operator=(last_local_number&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class printed_formal_ios_of_called_module : public GeneralFact
{
int w{}, e{};
string q{}, r{};
public:
printed_formal_ios_of_called_module(const string q1, const int w1,
const int e1, const string r1)
: q(q1), w(w1), e(e1), r(r1) {}

printed_formal_ios_of_called_module(
const printed_formal_ios_of_called_module& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~printed_formal_ios_of_called_module()
{
this->q.clear();this->w = NULL;
this->e = NULL; this->r.clear();
}
void operator=(GeneralFact& other) override
```

```
{
printed_formal_ios_of_called_module* ptr =
dynamic_cast<printed_formal_ios_of_called_module*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;  this->r = ptr->r;
}

printed_formal_ios_of_called_module(
printed_formal_ios_of_called_module&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(printed_formal_ios_of_called_module&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class it_includes_ifthen : public GeneralFact
{
int w{}, e{};
string q{};
public:
it_includes_ifthen(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

it_includes_ifthen(const it_includes_ifthen& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~it_includes_ifthen()
{
this->q.clear(); this->w = NULL;
```

```
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
it_includes_ifthen* ptr = dynamic_cast<it_includes_ifthen*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

it_includes_ifthen(it_includes_ifthen&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(it_includes_ifthen&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class it_includes_conditional_targeting : public GeneralFact
{
int w{}, e{};
string q{};
public:
it_includes_conditional_targeting(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

it_includes_conditional_targeting(const it_includes_conditional_targeting& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~it_includes_conditional_targeting()
{
this->q.clear(); this->w = NULL;
```

```
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
it_includes_conditional_targeting* ptr =
dynamic_cast<it_includes_conditional_targeting*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}

it_includes_conditional_targeting(
it_includes_conditional_targeting&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(it_includes_conditional_targeting&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class targets_conditional_variable : public GeneralFact
{
int w{}, e{}, r{};
string q{};
public:
targets_conditional_variable(const string q1, const int w1, const int e1,
const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

targets_conditional_variable(const targets_conditional_variable& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
```

```
~targets_conditional_variable()
{
this->q.clear();this->w = NULL;
this->e = NULL; this->r = NULL;
}
void operator=(GeneralFact& other) override
{
targets_conditional_variable* ptr =
dynamic_cast<targets_conditional_variable*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;  this->r = ptr->r;
}

targets_conditional_variable(targets_conditional_variable&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}

void operator=(targets_conditional_variable&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class variable_has_been_listed : public GeneralFact
{
string q{}, w{};
public:
variable_has_been_listed(const string q1, const string w1)
: q(q1), w(w1) {}

variable_has_been_listed(const variable_has_been_listed& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
```

```
~variable_has_been_listed()
{
this->q.clear(); this->w.clear();
}
void operator=(GeneralFact& other) override
{
variable_has_been_listed* ptr =
dynamic_cast<variable_has_been_listed*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

variable_has_been_listed(variable_has_been_listed&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(variable_has_been_listed&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class resetstyle : public GeneralFact
{
string q{};
public:
resetstyle(const string q1)
: q(q1) {}

resetstyle(const resetstyle& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~resetstyle()
{
this->q.clear();
```

```
}
void operator=(GeneralFact& other) override
{
resetstyle* ptr = dynamic_cast<resetstyle*>(&other);
this->q = ptr->q;
}

resetstyle(resetstyle&& other) noexcept
: q{ other.q } {}

void operator=(resetstyle&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class checkstyle : public GeneralFact
{
string q{};
public:
checkstyle(const string q1)
: q(q1) {}

checkstyle(const checkstyle& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~checkstyle()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
checkstyle* ptr = dynamic_cast<checkstyle*>(&other);
this->q = ptr->q;
```

```
}

checkstyle(checkstyle&& other) noexcept
: q{ other.q } {}

void operator=(checkstyle&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class total_local_entry : public GeneralFact
{
int q{};
public:
total_local_entry(const int q1)
: q(q1) {}

total_local_entry(const total_local_entry& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~total_local_entry()
{
this->q = NULL;

}
void operator=(GeneralFact& other) override
{
total_local_entry* ptr = dynamic_cast<total_local_entry*>(&other);
this->q = ptr->q;
}

total_local_entry(total_local_entry&& other) noexcept
: q{ other.q } {}
```

```
void operator=(total_local_entry&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class complex_next_state_operation_depth : public GeneralFact
{
int q{};
public:
complex_next_state_operation_depth(const int q1)
: q(q1) {}

complex_next_state_operation_depth(const complex_next_state_operation_depth& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~complex_next_state_operation_depth()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
complex_next_state_operation_depth* ptr =
dynamic_cast<complex_next_state_operation_depth*>(&other);
this->q = ptr->q;
}

complex_next_state_operation_depth(
complex_next_state_operation_depth&& other) noexcept
: q{ other.q } {}

void operator=(complex_next_state_operation_depth&& other) noexcept
{
```

```
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class output_filename : public GeneralFact
{
string q{};
public:
output_filename(const string q1)
: q(q1) {}

output_filename(const output_filename& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~output_filename()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
output_filename* ptr = dynamic_cast<output_filename*>(&other);
this->q = ptr->q;
}

output_filename(output_filename&& other) noexcept
: q{ other.q } {}

void operator=(output_filename&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
```

```
};

class hdl_io_pass : public GeneralFact
{
int q{};
public:
hdl_io_pass(const int q1)
: q(q1) {}

hdl_io_pass(const hdl_io_pass& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~hdl_io_pass()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
hdl_io_pass* ptr = dynamic_cast<hdl_io_pass*>(&other);
this->q = ptr->q;
}

hdl_io_pass(hdl_io_pass&& other) noexcept
: q{ other.q } {}

void operator=(hdl_io_pass&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class current_hdl_style : public GeneralFact
{
string q{};
```

```
public:
current_hdl_style(const string q1)
: q(q1) {}

current_hdl_style(const current_hdl_style& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~current_hdl_style()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
current_hdl_style* ptr = dynamic_cast<current_hdl_style*>(&other);
this->q = ptr->q;
}

current_hdl_style(current_hdl_style&& other) noexcept
: q{ other.q } {}

void operator=(current_hdl_style&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class call_ios_have_been_reset : public GeneralFact
{
string q{};
public:
call_ios_have_been_reset(const string q1)
: q(q1) {}

call_ios_have_been_reset(const call_ios_have_been_reset& other)
```

```
{
if (this != &other)
{
this->q = other.q;
}
}
~call_ios_have_been_reset()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
call_ios_have_been_reset* ptr =
dynamic_cast<call_ios_have_been_reset*>(&other);
this->q = ptr->q;
}

call_ios_have_been_reset(call_ios_have_been_reset&& other) noexcept
: q{ other.q } {}

void operator=(call_ios_have_been_reset&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class debug_mode : public GeneralFact
{
int q{};
public:
debug_mode(const int q1)
: q(q1) {}

debug_mode(const debug_mode& other)
{
if (this != &other)
{
this->q = other.q;
```

```
}
}
~debug_mode()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
debug_mode* ptr = dynamic_cast<debug_mode*>(&other);
this->q = ptr->q;
}

debug_mode(debug_mode&& other) noexcept
: q{ other.q } {}

void operator=(debug_mode&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class print_C_main_body : public GeneralFact
{
int q{};
public:
print_C_main_body(const int q1)
: q(q1) {}

print_C_main_body(const print_C_main_body& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~print_C_main_body()
{
this->q = NULL;
```

```
}
void operator=(GeneralFact& other) override
{
print_C_main_body* ptr = dynamic_cast<print_C_main_body*>(&other);
this->q = ptr->q;
}

print_C_main_body(print_C_main_body&& other) noexcept
: q{ other.q } {}

void operator=(print_C_main_body&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class cac_mode : public GeneralFact
{
int q{};
public:
cac_mode(const int q1)
: q(q1) {}

cac_mode(const cac_mode& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~cac_mode()
{
this->q = NULL;
}
void operator=(GeneralFact& other) override
{
cac_mode* ptr = dynamic_cast<cac_mode*>(&other);
this->q = ptr->q;
```

```
}

cac_mode(cac_mode&& other) noexcept
: q{ other.q } {}

void operator=(cac_mode&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class path : public GeneralFact
{
int w{}, e{};
string q{};
public:
path(const string q1, const int w1, const int e1)
: q(q1), w(w1), e(e1) {}

path(const path& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;
}
}
~path()
{
this->q.clear();this->w = NULL;
this->e = NULL;
}
void operator=(GeneralFact& other) override
{
path* ptr = dynamic_cast<path*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e;
}
```

```
path(path&& other) noexcept
: q{ other.q }, w{ other.w }, e{ other.e } {}

void operator=(path&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class max_path : public GeneralFact
{
int w{};
string q{};
public:
max_path(const string q1, const int w1)
: q(q1), w(w1) {}

max_path(const max_path& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~max_path()
{
this->q.clear(); this->w = NULL;
}
void operator=(GeneralFact& other) override
{
max_path* ptr = dynamic_cast<max_path*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

max_path(max_path&& other) noexcept
```

```
: q{ other.q }, w{ other.w } {}

void operator=(max_path&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class min_path : public GeneralFact
{
int w{};
string q{};
public:
min_path(const string q1, const int w1)
: q(q1), w(w1) {}

min_path(const min_path& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
}
}
~min_path()
{
this->q.clear(); this->w = NULL;
}
void operator=(GeneralFact& other) override
{
min_path* ptr = dynamic_cast<min_path*>(&other);
this->q = ptr->q; this->w = ptr->w;

}

min_path(min_path&& other) noexcept
: q{ other.q }, w{ other.w } {}

void operator=(min_path&& other) noexcept
```

```
{
std::swap(q, other.q); std::swap(w, other.w);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class op_belongs_to_state : public GeneralFact
{
int w{}, e{}, r{};
string q{};
public:
op_belongs_to_state(const string q1, const int w1, const int e1,
const int r1)
: q(q1), w(w1), e(e1), r(r1) {}

op_belongs_to_state(const op_belongs_to_state& other)
{
if (this != &other)
{
this->q = other.q; this->w = other.w;
this->e = other.e;  this->r = other.r;
}
}
~op_belongs_to_state()
{
this->q.clear();this->w = NULL;
this->e = NULL; this->r = NULL;

}
void operator=(GeneralFact& other) override
{
op_belongs_to_state* ptr = dynamic_cast<op_belongs_to_state*>(&other);
this->q = ptr->q; this->w = ptr->w;
this->e = ptr->e; this->r = ptr->r;
}

op_belongs_to_state(op_belongs_to_state&& other) noexcept
: q{ other.q }, w{ other.w },
e{ other.e }, r{ other.r } {}
```

```
void operator=(op_belongs_to_state&& other) noexcept
{
std::swap(q, other.q); std::swap(w, other.w);
std::swap(e, other.e); std::swap(r, other.r);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class top_module : public GeneralFact
{
string q{};
public:
top_module(const string q1)
: q(q1) {}

top_module(const top_module& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~top_module()
{
this->q.clear();

}
void operator=(GeneralFact& other) override
{
top_module* ptr = dynamic_cast<top_module*>(&other);
this->q = ptr->q;
}

top_module(top_module&& other) noexcept
: q{ other.q } {}

void operator=(top_module&& other) noexcept
{
```

```
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};

class package_name : public GeneralFact
{
string q{};
public:
package_name(const string q1)
: q(q1) {}

package_name(const package_name& other)
{
if (this != &other)
{
this->q = other.q;
}
}
~package_name()
{
this->q.clear();
}
void operator=(GeneralFact& other) override
{
package_name* ptr = dynamic_cast<package_name*>(&other);
this->q = ptr->q;
}

package_name(package_name&& other) noexcept
: q{ other.q } {}

void operator=(package_name&& other) noexcept
{
std::swap(q, other.q);
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
```

```
};

class GenfactError : public GeneralFact
{
string saying{};
public:
GenfactError(string pepe)
:saying(pepe) {}
void operator=(GeneralFact& other) override
{
GenfactError* ptr = dynamic_cast<GenfactError*>(&other);
if (this != &other)
this->saying = ptr->saying;
}
~GenfactError()
{
this->saying.clear();
}
friend string makeStringOf(GeneralFact* obj);

friend size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj);
};
```

7.3 interfacetxt.h

```
    #pragma once
#include "AVL.h"
#include "allFactStructure.h"
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <fstream>
#include <stdio.h>
#include <typeinfo>
#include <functional>

using namespace std;

const char pa = '('; // parenthesis open

const char pacl = ')'; // parenthesis close

const char co = ','; // coma

const char br = '['; // bracket open

const char brcl = ']'; // bracket close

const char us = '_'; // underscore

const char str  =  '*'; // star

/// @brief makes string of data
/// @param obj <- the data
string makeStringOf(GeneralFact* obj)
{
size_t prlst; // pre-last element of a vector
stringstream ss;
string ALine;
bool flag = 1;

if (typeid(type_def) == typeid(*obj))
```

```
{
flag = 0;
type_def* ptr = dynamic_cast<type_def*>(obj);
ss << "type_def(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i << co << ptr->o;
}
else if (typeid(op_def) == typeid(*obj))
{
flag = 0;
op_def* ptr = dynamic_cast<op_def*>(obj);
ss << "op_def(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u;
}
else if (typeid(hierarchy_part) == typeid(*obj ))
{
flag = 0;
hierarchy_part* ptr = dynamic_cast<hierarchy_part*>(obj);
ss << "hierarchy_part(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u;
}
else if (typeid(data_stmt) == typeid(*obj))
{
flag = 0;
data_stmt* ptr = dynamic_cast<data_stmt*>(obj);
ss << "data_stmt(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y;
}
else if (typeid(prog_stmt) == typeid(*obj))
{
flag = 0;
prog_stmt* ptr = dynamic_cast<prog_stmt*>(obj);
ss << "prog_stmt(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i;
}
else if (typeid(joint_stmt) == typeid(*obj))
{
flag = 0;
joint_stmt* ptr = dynamic_cast<joint_stmt*>(obj);
ss << "joint_stmt(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y;
}
```

```
else if (typeid(call_stmt) == typeid(*obj))
{
flag = 0;
call_stmt* ptr = dynamic_cast<call_stmt*>(obj);
ss << "call_stmt(" << ptr->q << co << ptr->w << co << ptr->e << co << br;
//vector
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst;++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;
}
else if (typeid(compo_stmt) == typeid(*obj))
{
flag = 0;
compo_stmt* ptr = dynamic_cast<compo_stmt*>(obj);
ss << "compo_stmt(" << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii)
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;
}
else if (typeid(rec_stmt) == typeid(*obj))
{
flag = 0;
rec_stmt* ptr = dynamic_cast<rec_stmt*>(obj);
ss << "rec_stmt(" << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
```

```
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl;
}
else if (typeid(special_op) == typeid(*obj))
{
flag = 0;
special_op* ptr = dynamic_cast<special_op*>(obj);
ss << "special_op(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i << co << ptr->o
<< co << ptr->p;
}
else if (typeid(special_dt) == typeid(*obj))
{
flag = 0;
special_dt* ptr = dynamic_cast<special_dt*>(obj);
ss << "special_dt(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u;

}
else if (typeid(local_object) == typeid(*obj))
{
flag = 0;
local_object* ptr = dynamic_cast<local_object*>(obj);
ss << "local_object(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i << co << ptr->o;
}
else if (typeid(subprogram_call) == typeid(*obj))
{
flag = 0;
state_node* ptr = dynamic_cast<state_node*>(obj);
ss << "state_node(" << ptr->q << co << ptr->w << co;
subprogram_call* ptr2 = dynamic_cast<subprogram_call*>(obj);
ss << "subprogram_call(" << br;
//vector
```

```
if (!ptr2->q.empty())
{
prlst = ptr2->q.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->q[ii] << co;
}
ss << ptr2->q.back();
}
ss << brcl << co << ptr2->w << pacl;
}
else if (typeid(dataflow) == typeid(*obj))
{
flag = 0;
state_node* ptr = dynamic_cast<state_node*>(obj);
ss << "state_node(" << ptr->q << co << ptr->w << co;
dataflow* ptr2 = dynamic_cast<dataflow*>(obj);
ss << "dataflow(" << br;
//vector
if (!ptr2->q.empty())
{
prlst = ptr2->q.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->q[ii] << co;
}
ss << ptr2->q.back();
}
ss << brcl << co << ptr2->w << pacl;
}
else if (typeid(ifthen) == typeid(*obj))
{
flag = 0;
state_node* ptr = dynamic_cast<state_node*>(obj);
ss << "state_node(" << ptr->q << co << ptr->w << co;
ifthen* ptr2 = dynamic_cast<ifthen*>(obj);
ss << "ifthen(" << br;
//vector
if (!ptr2->q.empty())
{
prlst = ptr2->q.size() - 1;
```

```
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->q[ii] << co;
}
ss << ptr2->q.back();
}
ss << brcl << co << br;
//vector2
if (!ptr2->w.empty())
{
prlst = ptr2->w.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->w[ii] << co;
}
ss << ptr2->w.back();
}
ss << brcl << co << br;
//vector3
if (!ptr2->e.empty())
{
prlst = ptr2->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->e[ii] << co;
}
ss << ptr2->e.back();
}
ss << brcl << co << ptr2->r << co << ptr2->t << pacl;
}
else        if        (typeid(jump)        ==        typeid(*obj))
//losing align after 15 else if
{
flag = 0;
state_node* ptr = dynamic_cast<state_node*>(obj);
ss << "state_node(" << ptr->q << co << ptr->w << co;
jump* ptr2 = dynamic_cast<jump*>(obj);
ss << "jump" << pa << br;
//vector
if (!ptr2->q.empty())
{
```

```
prlst = ptr2->q.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->q[ii] << co;
}
ss << ptr2->q.back();
}
ss << brcl << co << ptr2->w << pacl;
}
else if (typeid(return_cos) == typeid(*obj))
{
flag = 0;
state_node* ptr = dynamic_cast<state_node*>(obj);
ss << "state_node(" << ptr->q << co << ptr->w << co;
return_cos* ptr2 = dynamic_cast<return_cos*>(obj);
ss << "return(" << br;
//vector
if (!ptr2->q.empty())
{
prlst = ptr2->q.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr2->q[ii] << co;
}
ss << ptr2->q.back();
}
ss << brcl << pacl;
}
else if (typeid(change_op_number) == typeid(*obj))
{
flag = 0;
change_op_number* ptr = dynamic_cast<change_op_number*>(obj);
ss << "change_op_number(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(last_change_op_number) == typeid(*obj))
{
flag = 0;
last_change_op_number* ptr = dynamic_cast<last_change_op_number*>(obj);
ss << "last_change_op_number(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(op_guards) == typeid(*obj))
```

```
{
flag = 0;
op_guards* ptr = dynamic_cast<op_guards*>(obj);
ss << "op_guards(" << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;
}
else if (typeid(var_guards) == typeid(*obj))
{
flag = 0;
var_guards* ptr = dynamic_cast<var_guards*>(obj);
ss << "var_guards("  << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
```

```
}
ss << brcl << co << br;
//vector2
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;
}
else if (typeid(guard_pair) == typeid(*obj))
{
flag = 0;
guard_pair* ptr = dynamic_cast<guard_pair*>(obj);
ss << "guard_pair(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r;
}
else if (typeid(guard_cond) == typeid(*obj))
{
flag = 0;
guard_cond* ptr = dynamic_cast<guard_cond*>(obj);
ss << "guard_cond("  << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r;
}
else if (typeid(predecessors) == typeid(*obj))
{
flag = 0;
predecessors* ptr = dynamic_cast<predecessors*>(obj);
ss << "predecessors("  << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
```

```
ss << brcl;


}
else if (typeid(cessor) == typeid(*obj))
{
flag = 0;
cessor* ptr = dynamic_cast<cessor*>(obj);
ss << "cessor("  << ptr->q << co << ptr->w << co << ptr->e ;
}
else if (typeid(cessor_kind) == typeid(*obj))
{
flag = 0;
cessor_kind* ptr = dynamic_cast<cessor_kind*>(obj);
ss << "cessor_kind("  << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << br;
//vector
if (!ptr->u.empty())
{
prlst = ptr->u.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->u[ii] << co;
}
ss << ptr->u.back();
}
ss << brcl;

}
else if (typeid(old_schedule) == typeid(*obj))
{
flag = 0;
old_schedule* ptr = dynamic_cast<old_schedule*>(obj);
ss << "old_schedule("  << ptr->q;
}
else if (typeid(new_schedule) == typeid(*obj))
{
flag = 0;
new_schedule* ptr = dynamic_cast<new_schedule*>(obj);
ss << "new_schedule("  << ptr->q;
}
```

```
else if (typeid(local_ifthen_chain_end_operations_were_written) == typeid(*obj))
{
flag = 0;
local_ifthen_chain_end_operations_were_written* ptr =
dynamic_cast<local_ifthen_chain_end_operations_were_written*>(obj);
ss << "local_ifthen_chain_end_operations_were_written(" << ptr->q;
}
else if (typeid(calls_list) == typeid(*obj))
{
flag = 0;
calls_list* ptr = dynamic_cast<calls_list*>(obj);
ss << "calls_list(" << ptr->q << co << ptr->w << co << ptr->e << co << br;
//vector
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;
}
else if (typeid(composites_list) == typeid(*obj))
{
flag = 0;
composites_list* ptr = dynamic_cast<composites_list*>(obj);
ss << "composites_list(" << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl;
}
```

```
else if (typeid(record_aggregates_list) == typeid(*obj))
{
flag = 0;
record_aggregates_list* ptr = dynamic_cast<record_aggregates_list*>(obj);
ss << "record_aggregates_list(" << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl;
}
else if (typeid(mem_port) == typeid(*obj))
{
flag = 0;
mem_port* ptr = dynamic_cast<mem_port*>(obj);
ss << "mem_port(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i << co
<< ptr->o << co << ptr->p << co << ptr->a << co << ptr->s << co << ptr->d;
}
else if (typeid(global_declarations) == typeid(*obj))
{
stringstream tt;
flag = 0;
int ii = 0;
size_t vs;
global_declarations* ptr = dynamic_cast<global_declarations*>(obj);
ss << "global_declarations(" << br;
if (!ptr->q.empty())
{
vs = ptr->q.size();
for (int ii = 0; ii < vs; ii++)
{
ss << "local_object(" << ptr->q[ii].q << co << ptr->q[ii].w << co
<< ptr->q[ii].e << co << ptr->q[ii].r << co << ptr->q[ii].t << co
<< ptr->q[ii].y << co << ptr->q[ii].u << co << ptr->q[ii].i << co
```

```
<< ptr->q[ii].o << pacl << co;
}
string tmp = ss.str();
tmp.resize(tmp.size() - 1);  // getting rid of the last coma
swap(ss, tt);
ss << tmp;
}
ss << brcl << co << ptr->w;
}
else if (typeid(source_is_normal_dt) == typeid(*obj))
{
flag = 0;
source_is_normal_dt* ptr = dynamic_cast<source_is_normal_dt*>(obj);
ss << "source_is_normal_dt("  << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(combo) == typeid(*obj))
{
flag = 0;
combo* ptr = dynamic_cast<combo*>(obj);
ss << "combo("  << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(sequence) == typeid(*obj))
{
flag = 0;
sequence* ptr = dynamic_cast<sequence*>(obj);
ss << "sequence("  << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(for_loop) == typeid(*obj))
{
flag = 0;
for_loop* ptr = dynamic_cast<for_loop*>(obj);
ss << "for_loop("  << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i << co
<< ptr->o << co << ptr->p << co << ptr->a << co << ptr->s << co << ptr->d;
}
else if (typeid(last_for_loop_entry) == typeid(*obj))
{
flag = 0;
last_for_loop_entry* ptr = dynamic_cast<last_for_loop_entry*>(obj);
ss << "last_for_loop_entry("  << ptr->q;
}
```

```
else if (typeid(while_loop) == typeid(*obj))
{
flag = 0;
while_loop* ptr = dynamic_cast<while_loop*>(obj);
ss << "while_loop("  << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << ptr->u;
}
else if (typeid(last_while_loop_entry) == typeid(*obj))
{
flag = 0;
last_while_loop_entry* ptr = dynamic_cast<last_while_loop_entry*>(obj);
ss << "last_while_loop_entry("  << ptr->q;
}
else if (typeid(possible_end_if) == typeid(*obj))
{
flag = 0;
possible_end_if* ptr = dynamic_cast<possible_end_if*>(obj);
ss << "possible_end_if("  << ptr->q << co << br;
//vector
if (!ptr->w.empty())
{
prlst = ptr->w.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->w[ii] << co;
}
ss << ptr->w.back();
}
ss << brcl;
}
else if (typeid(end_if) == typeid(*obj))
{
flag = 0;
end_if* ptr = dynamic_cast<end_if*>(obj);
ss << "end_if("  << ptr->q << co << br;
//vector
if (!ptr->w.empty())
{
prlst = ptr->w.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
```

```
ss << ptr->w[ii] << co;
}
ss << ptr->w.back();
}
ss << brcl;

}
else if (typeid(nested_cond_fact) == typeid(*obj))
{
flag = 0;
nested_cond_fact* ptr = dynamic_cast<nested_cond_fact*>(obj);
ss << "nested_cond_fact("  << ptr->q << co << br;
//vector
if (!ptr->w.empty())
{
prlst = ptr->w.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->w[ii] << co;
}
ss << ptr->w.back();
}
ss << brcl;
}
else if (typeid(top_level_call) == typeid(*obj))
{
flag = 0;
top_level_call* ptr = dynamic_cast<top_level_call*>(obj);
ss << "top_level_call("  << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->r << co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i
<< co << ptr->o << co << ptr->p << co << ptr->a;
}
else if (typeid(top_level_call_parcs) == typeid(*obj))
{
flag = 0;
top_level_call_parcs* ptr = dynamic_cast<top_level_call_parcs*>(obj);
ss << "top_level_call_parcs("  << ptr->q << co << ptr->w << co << ptr->e
<< co << ptr->r << co << ptr->t << co << ptr->y << co << ptr->u << co
<< ptr->i << co << ptr->o << co << ptr->p << co << ptr->a;
}
else if (typeid(added_aux_call_ios) == typeid(*obj))
```

```
{
flag = 0;
added_aux_call_ios* ptr = dynamic_cast<added_aux_call_ios*>(obj);
ss << "added_aux_call_ios("  << ptr->q << co << ptr->w ;
}
else if (typeid(added_aux_call_ios1) == typeid(*obj))
{
flag = 0;
added_aux_call_ios1* ptr = dynamic_cast<added_aux_call_ios1*>(obj);
ss << "added_aux_call_ios1("  << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(added_aux_call_signals) == typeid(*obj))
{
flag = 0;
added_aux_call_signals* ptr = dynamic_cast<added_aux_call_signals*>(obj);
ss << "added_aux_call_signals("  << ptr->q << co << ptr->w;
}
else if (typeid(found_call_operator) == typeid(*obj))
{
flag = 0;
found_call_operator* ptr = dynamic_cast<found_call_operator*>(obj);
ss << "found_call_operator("  << ptr->q << co << ptr->w;
}
else if (typeid(added_verilog_aux_call_outputs) == typeid(*obj))
{
flag = 0;
added_verilog_aux_call_outputs* ptr =
dynamic_cast<added_verilog_aux_call_outputs*>(obj);
ss << "added_verilog_aux_call_outputs("  << ptr->q << co << ptr->w
<< co << ptr->e;
}
else if (typeid(raw_dependencies) == typeid(*obj))
{
flag = 0;
raw_dependencies* ptr = dynamic_cast<raw_dependencies*>(obj);
ss << "raw_dependencies("  << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
```

```
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;

}
else if (typeid(war_dependencies) == typeid(*obj))
{
flag = 0;
war_dependencies* ptr = dynamic_cast<war_dependencies*>(obj);
ss << "war_dependencies(" << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
```

```
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;
}
else if (typeid(waw_dependencies) == typeid(*obj))
{
flag = 0;
waw_dependencies* ptr = dynamic_cast<waw_dependencies*>(obj);
ss << "waw_dependencies("  << ptr->q << co << ptr->w << co << br;
//vector
if (!ptr->e.empty())
{
prlst = ptr->e.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->e[ii] << co;
}
ss << ptr->e.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl;

}
else if (typeid(schedule) == typeid(*obj))
{
flag = 0;
schedule* ptr = dynamic_cast<schedule*>(obj);
ss << "schedule("  << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r;
}
```

```
else if (typeid(last_conditional_execution) == typeid(*obj))
{
flag = 0;
last_conditional_execution* ptr = dynamic_cast<last_conditional_execution*>(obj);
ss << "last_conditional_execution("  << ptr->q << co << ptr->w;
}
else if (typeid(conditional_operations) == typeid(*obj))
{
flag = 0;
conditional_operations* ptr = dynamic_cast<conditional_operations*>(obj);
ss << "conditional_operations("  << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->r << co << br;
//vector
if (!ptr->t.empty())
{
prlst = ptr->t.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->t[ii] << co;
}
ss << ptr->t.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->y.empty())
{
prlst = ptr->y.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->y[ii] << co;
}
ss << ptr->y.back();
}
ss << brcl << co << br;
//vector3
if (!ptr->u.empty())
{
prlst = ptr->u.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->u[ii] << co;
```

```
}
ss << ptr->u.back();
}
ss << brcl << co << br;
//vector4
if (!ptr->i.empty())
{
prlst = ptr->i.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->i[ii] << co;
}
ss << ptr->i.back();
}
ss << brcl;
}
else if (typeid(last_conditional_transition_of_schedule) == typeid(*obj))
{
flag = 0;
last_conditional_transition_of_schedule* ptr =
dynamic_cast<last_conditional_transition_of_schedule*>(obj);
ss << "last_conditional_transition_of_schedule(" << ptr->q << co << ptr->w
<< co << ptr->e;
}
else if (typeid(transition_to_be_rescheduled) == typeid(*obj))
{
flag = 0;
transition_to_be_rescheduled* ptr =
dynamic_cast<transition_to_be_rescheduled*>(obj);
ss << "transition_to_be_rescheduled(" << ptr->q << co << ptr->w << co << ptr->e
<< co << ptr->r << co << ptr->t;
}
else if (typeid(last_conditional_transition) == typeid(*obj))
{
flag = 0;
last_conditional_transition* ptr = dynamic_cast<last_conditional_transition*>(obj);
ss << "last_conditional_transition(" << ptr->q << co << ptr->w;
}
else if (typeid(conditional_transitions) == typeid(*obj))
{
flag = 0;
```

```
conditional_transitions* ptr = dynamic_cast<conditional_transitions*>(obj);
ss << "conditional_transitions(" << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->r << co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i;
}
else if (typeid(state) == typeid(*obj))
{
flag = 0;
state* ptr = dynamic_cast<state*>(obj);
ss << "state(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t << co << ptr->y << co << br;
//vector
if (!ptr->u.empty())
{
prlst = ptr->u.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->u[ii] << co;
}
ss << ptr->u.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->i.empty())
{
prlst = ptr->i.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->i[ii] << co;
}
ss << ptr->i.back();
}
ss << brcl;
}
else if (typeid(rescheduled) == typeid(*obj))
{
flag = 0;
rescheduled* ptr = dynamic_cast<rescheduled*>(obj);
ss << "rescheduled(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r
<< co << ptr->t;
}
else if (typeid(last_rescheduled) == typeid(*obj))
```

```
{
flag = 0;
last_rescheduled* ptr = dynamic_cast<last_rescheduled*>(obj);
ss << "last_rescheduled(" << ptr->q << co << ptr->w << co << ptr->e << co <<
ptr->r << co << ptr->t;
}
else if (typeid(raw_cessor) == typeid(*obj))
{
flag = 0;
raw_cessor* ptr = dynamic_cast<raw_cessor*>(obj);
ss << "raw_cessor(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(war_cessor) == typeid(*obj))
{
flag = 0;
war_cessor* ptr = dynamic_cast<war_cessor*>(obj);
ss << "war_cessor(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(waw_cessor) == typeid(*obj))
{
flag = 0;
waw_cessor* ptr = dynamic_cast<waw_cessor*>(obj);
ss << "waw_cessor(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(op_resource) == typeid(*obj))
{
flag = 0;
op_resource* ptr = dynamic_cast<op_resource*>(obj);
ss << "op_resource(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(global_resource) == typeid(*obj))
{
flag = 0;
global_resource* ptr = dynamic_cast<global_resource*>(obj);
ss << "global_resource(" << ptr->q;
}
else if (typeid(module_g_resource) == typeid(*obj))
{
flag = 0;
module_g_resource* ptr = dynamic_cast<module_g_resource*>(obj);
ss << "module_g_resource(" << ptr->q << co << ptr->w;
```

```
}
else if (typeid(cf_previous_op) == typeid(*obj))
{
flag = 0;
cf_previous_op* ptr = dynamic_cast<cf_previous_op*>(obj);
ss << "cf_previous_op(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r;
}
else if (typeid(cf_previous_state) == typeid(*obj))
{
flag = 0;
cf_previous_state* ptr = dynamic_cast<cf_previous_state*>(obj);
ss << "cf_previous_state(" << ptr->q << co << ptr->w << co << ptr->e << co << ptr->r;
}
else if (typeid(pred_candidate_examined) == typeid(*obj))
{
flag = 0;
pred_candidate_examined* ptr = dynamic_cast<pred_candidate_examined*>(obj);
ss << "pred_candidate_examined(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(reentrant_triangle) == typeid(*obj))
{
flag = 0;
reentrant_triangle* ptr = dynamic_cast<reentrant_triangle*>(obj);
ss << "reentrant_triangle(" << ptr->q << co << ptr->w << co << ptr->e << co <<
ptr->r << co << br;
//vector
if (!ptr->t.empty())
{
prlst = ptr->t.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->t[ii] << co;
}
ss << ptr->t.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->y.empty())
{
prlst = ptr->y.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
```

```
{
ss << ptr->y[ii] << co;
}
ss << ptr->y.back();
}
ss << brcl << co << br;
//vector3
if (!ptr->u.empty())
{
prlst = ptr->u.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->u[ii] << co;
}
ss << ptr->u.back();
}
ss << brcl << co << br;
//vector4
if (!ptr->i.empty())
{
prlst = ptr->i.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->i[ii] << co;
}
ss << ptr->i.back();
}
ss << brcl << co << br;
//vector5
if (!ptr->o.empty())
{
prlst = ptr->o.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->o[ii] << co;
}
ss << ptr->o.back();
}
ss << brcl << co << br;
//vector6
if (!ptr->p.empty())
```

```
{
prlst = ptr->p.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->p[ii] << co;
}
ss << ptr->p.back();
}
ss << brcl << co << ptr->a << co << ptr->s << co << ptr->d << co << ptr->f;
}
else if (typeid(last_reentrant_triangle) == typeid(*obj))
{
flag = 0;
last_reentrant_triangle* ptr = dynamic_cast<last_reentrant_triangle*>(obj);
ss << "last_reentrant_triangle(" << ptr->q << co << ptr->w;
}
else if (typeid(last_schedule_state) == typeid(*obj))
{
flag = 0;
last_schedule_state* ptr = dynamic_cast<last_schedule_state*>(obj);
ss << "last_schedule_state(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(conditional_incomplete) == typeid(*obj))
{
flag = 0;
conditional_incomplete* ptr = dynamic_cast<conditional_incomplete*>(obj);
ss << "conditional_incomplete(" << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->r << co << br;
//vector
if (!ptr->t.empty())
{
prlst = ptr->t.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->t[ii] << co;
}
ss << ptr->t.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->y.empty())
```

```
{
prlst = ptr->y.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->y[ii] << co;
}
ss << ptr->y.back();
}
ss << brcl << co << br;
//vector3
if (!ptr->u.empty())
{
prlst = ptr->u.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->u[ii] << co;
}
ss << ptr->u.back();
}
ss << brcl << co << br;
//vector4
if (!ptr->i.empty())
{
prlst = ptr->i.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->i[ii] << co;
}
ss << ptr->i.back();
}
ss << brcl << co << br;
//vector5
if (!ptr->o.empty())
{
prlst = ptr->o.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->o[ii] << co;
}
ss << ptr->o.back();
}
```

```
ss << brcl << co << br;
//vector6
if (!ptr->p.empty())
{
prlst = ptr->p.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->p[ii] << co;
}
ss << ptr->p.back();
}
ss << brcl << co << ptr->a << co << ptr->s << co << ptr->d << co << ptr->f << co
<< ptr->g << co << ptr->h << co << ptr->j << co << ptr->k << co << ptr->l << co
<< ptr->z << co << ptr->x;
}
else if (typeid(mixed_incomplete_state_lists) == typeid(*obj))
{
flag = 0;
mixed_incomplete_state_lists* ptr =
dynamic_cast<mixed_incomplete_state_lists*>(obj);
ss << "mixed_incomplete_state_lists(" << ptr->q << co << ptr->w << co << ptr->e
<< co << br;
//vector
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->t.empty())
{
prlst = ptr->t.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->t[ii] << co;
}
```

```
ss << ptr->t.back();
}
ss << brcl;
}
else if (typeid(linear_incomplete_node) == typeid(*obj))
{
flag = 0;
linear_incomplete_node* ptr = dynamic_cast<linear_incomplete_node*>(obj);
ss << "linear_incomplete_node(" << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->y << co << br;
//vector
if (!ptr->r.empty())
{
prlst = ptr->r.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->r[ii] << co;
}
ss << ptr->r.back();
}
ss << brcl << co << br;
//vector2
if (!ptr->t.empty())
{
prlst = ptr->t.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << ptr->t[ii] << co;
}
ss << ptr->t.back();
}
ss << brcl;
}
else if (typeid(incomplete_links) == typeid(*obj))
{
flag = 0;
incomplete_links* ptr = dynamic_cast<incomplete_links*>(obj);
ss << "incomplete_links(" << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->r << co << ptr->t << co << ptr->y << co << ptr->u << co << ptr->i
<< co << ptr->o << co << ptr->p;
}
```

```
else if (typeid(last_incomplete) == typeid(*obj))
{
flag = 0;
last_incomplete* ptr = dynamic_cast<last_incomplete*>(obj);
ss << "last_incomplete(" << ptr->q << co << ptr->w << co << ptr->e << co
<< ptr->r;
}
else if (typeid(global_nils) == typeid(*obj))
{
flag = 0;
global_nils* ptr = dynamic_cast<global_nils*>(obj);
ss << "global_nils(" << br ;
//vector
if (!ptr->q.empty())
{
prlst = ptr->q.size() - 1;
for (int ii = 0; ii < prlst; ++ii) // loop until pre-last element.
{
ss << "nil_node" << pa;

ss << ptr->q[ii].q << co << ptr->q[ii].w << pacl << co;
}
ss << "nil_node" << pa << ptr->q.back().q << co << ptr->q.back().w;
ss << pacl;
}
ss << brcl;
}
else if (typeid(current_module) == typeid(*obj))
{
flag = 0;
current_module* ptr = dynamic_cast<current_module*>(obj);
ss << "current_module(" << ptr->q;
}
else if (typeid(last_linear_incomplete_node) == typeid(*obj))
{
flag = 0;
last_linear_incomplete_node* ptr =
dynamic_cast<last_linear_incomplete_node*>(obj);
ss << "last_linear_incomplete_node(" << ptr->q;
}
else if (typeid(operator_instances) == typeid(*obj))
```

```
{
flag = 0;
operator_instances* ptr = dynamic_cast<operator_instances*>(obj);
ss << "operator_instances(" << ptr->q << co << ptr->w;
}
else if (typeid(massively_parallel_style) == typeid(*obj))
{
flag = 0;
massively_parallel_style* ptr =
dynamic_cast<massively_parallel_style*>(obj);
ss << "massively_parallel_style(" << ptr->q;
}
else if (typeid(hdl_style) == typeid(*obj))
{
flag = 0;
hdl_style* ptr = dynamic_cast<hdl_style*>(obj);
ss << "hdl_style(" << ptr->q;
}
else if (typeid(op_instance) == typeid(*obj))
{
flag = 0;
op_instance* ptr = dynamic_cast<op_instance*>(obj);
ss << "op_instance(" << ptr->q << co << ptr->w << co << ptr->e <<
co << ptr->r << co << ptr->t;
}
else if (typeid(last_op_instance) == typeid(*obj))
{
flag = 0;
last_op_instance* ptr = dynamic_cast<last_op_instance*>(obj);
ss << "last_op_instance(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(op_in_a_state) == typeid(*obj))
{
flag = 0;
op_in_a_state* ptr = dynamic_cast<op_in_a_state*>(obj);
ss << "op_in_a_state(" << ptr->q << co << ptr->w << co << ptr->e <<
co << ptr->r << co << ptr->t;
}
else if (typeid(last_op_in_a_state) == typeid(*obj))
{
flag = 0;
```

```
last_op_in_a_state* ptr = dynamic_cast<last_op_in_a_state*>(obj);
ss << "last_op_in_a_state(" << ptr->q << co << ptr->w << co << ptr->e
<< co << ptr->r;
}
else if (typeid(signal_instance) == typeid(*obj))
{
flag = 0;
signal_instance* ptr = dynamic_cast<signal_instance*>(obj);
ss << "signal_instance(" << ptr->q << co << ptr->w << co << ptr->e <<
co << ptr->r << co << ptr->t << co << ptr->y << co << ptr->u;
}
else if (typeid(last_signal_instance) == typeid(*obj))
{
flag = 0;
last_signal_instance* ptr = dynamic_cast<last_signal_instance*>(obj);
ss << "last_signal_instance(" << ptr->q << co << ptr->w;
}
else if (typeid(output_instance) == typeid(*obj))
{
flag = 0;
output_instance* ptr = dynamic_cast<output_instance*>(obj);
ss << "output_instance(" << ptr->q << co << ptr->w << co << ptr->e <<
co << ptr->r << co << ptr->t << co << ptr->y;
}
else if (typeid(last_output_instance) == typeid(*obj))
{
flag = 0;
last_output_instance* ptr = dynamic_cast<last_output_instance*>(obj);
ss << "last_output_instance(" << ptr->q << co << ptr->w;
}
else if (typeid(operator_instance_stats) == typeid(*obj))
{
flag = 0;
operator_instance_stats* ptr = dynamic_cast<operator_instance_stats*>(obj);
ss << "operator_instance_stats(" << ptr->q << co << ptr->w << co <<
ptr->e << co << ptr->r;
}
else if (typeid(consecutive_106) == typeid(*obj))
{
flag = 0;
consecutive_106* ptr = dynamic_cast<consecutive_106*>(obj);
```

```
ss << "consecutive_106(" << ptr->q;
}
else if (typeid(operation_order) == typeid(*obj))
{
flag = 0;
operation_order* ptr = dynamic_cast<operation_order*>(obj);
ss << "operation_order(" << ptr->q << co << ptr->w << co << ptr->e <<
co << ptr->r << co << ptr->t << co << ptr->y << co << ptr->u << co <<
ptr->i << co << ptr->o << co << ptr->p;
}
else if (typeid(max_parallel_call_order) == typeid(*obj))
{
flag = 0;
max_parallel_call_order* ptr =
dynamic_cast<max_parallel_call_order*>(obj);
ss << "max_parallel_call_order(" << ptr->q << co << ptr->w << co <<
ptr->e << co << ptr->r;
}
else if (typeid(max_op_order) == typeid(*obj))
{
flag = 0;
max_op_order* ptr = dynamic_cast<max_op_order*>(obj);
ss << "max_op_order(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(totalmax_call_order) == typeid(*obj))
{
flag = 0;
totalmax_call_order* ptr = dynamic_cast<totalmax_call_order*>(obj);
ss << "totalmax_call_order(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(totalmax_gross_depth) == typeid(*obj))
{
flag = 0;
totalmax_gross_depth* ptr = dynamic_cast<totalmax_gross_depth*>(obj);
ss << "totalmax_gross_depth(" << ptr->q << co << ptr->w << co << ptr->e;
}
else if (typeid(current_total_max_order_entry) == typeid(*obj))
{
flag = 0;
current_total_max_order_entry* ptr =
dynamic_cast<current_total_max_order_entry*>(obj);
```

```
ss << "current_total_max_order_entry(" << ptr->q;
}
else if (typeid(module_last_state) == typeid(*obj))
{
flag = 0;
module_last_state* ptr = dynamic_cast<module_last_state*>(obj);
ss << "module_last_state(" << ptr->q;
}
else if (typeid(module_local_list) == typeid(*obj))
{
stringstream tt;
flag = 0;
module_local_list* ptr = dynamic_cast<module_local_list*>(obj);
int ii = 0;
size_t vs;
ss << "module_local_list(" << br;
if (!ptr->q.empty())
{
vs = ptr->q.size();
for (int ii = 0; ii < vs; ii++)
{
ss << "local_object(" << ptr->q[ii].q << co << ptr->q[ii].w <<
co << ptr->q[ii].e << co << ptr->q[ii].r << co << ptr->q[ii].t
<< co << ptr->q[ii].y << co << ptr->q[ii].u << co << ptr->q[ii].i
<< co << ptr->q[ii].o << pacl << co;
}
string tmp = ss.str();
tmp.resize(tmp.size() - 1);  // getting rid of the last
swap(ss, tt);
ss << tmp;
ss << brcl;
}
}
if (flag) //exeeded the limit of nested else if
{
if (typeid(module_local_list_parcs) == typeid(*obj) )
{
stringstream tt;
module_local_list_parcs* ptr =
dynamic_cast<module_local_list_parcs*>(obj);
int ii = 0;
```

```
size_t vs;
ss << "module_local_list_parcs(" << br;
if (!ptr->q.empty())
{
vs = ptr->q.size();
for (int ii = 0; ii < vs; ii++)
{
ss << "local_object(" << ptr->q[ii].q << co << ptr->q[ii].w
<< co << ptr->q[ii].e << co << ptr->q[ii].r << co << ptr->q[ii].t
<< co << ptr->q[ii].y << co << ptr->q[ii].u << co << ptr->q[ii].i
<< co << ptr->q[ii].o << pacl << co;
}
string tmp = ss.str();
tmp.resize(tmp.size() - 1);  // getting rid of the last
swap(ss, tt);
ss << tmp;
ss << brcl;
}
}
else if (typeid(last_non_io_found) == typeid(*obj))
{
last_non_io_found* ptr = dynamic_cast<last_non_io_found*>(obj);
ss << "last_non_io_found(" << ptr->q;
}
else if (typeid(last_local_number) == typeid(*obj))
{
last_local_number* ptr = dynamic_cast<last_local_number*>(obj);
ss << "last_local_number(" << ptr->q;
}
else if (typeid(printed_formal_ios_of_called_module) == typeid(*obj))
{
printed_formal_ios_of_called_module* ptr =
dynamic_cast<printed_formal_ios_of_called_module*>(obj);
ss << "printed_formal_ios_of_called_module(" << ptr->q << co << ptr->w
<< co << ptr->e << co << ptr->r;
}
else if (typeid(it_includes_ifthen) == typeid(*obj))
{
it_includes_ifthen* ptr = dynamic_cast<it_includes_ifthen*>(obj);
ss << "it_includes_ifthen(" << ptr->q << co << ptr->w << co << ptr->e;
}
```

```
else if (typeid(it_includes_conditional_targeting) == typeid(*obj))
{
it_includes_conditional_targeting* ptr =
dynamic_cast<it_includes_conditional_targeting*>(obj);
ss << "it_includes_conditional_targeting(" << ptr->q << co << ptr->w
<< co << ptr->e;
}
else if (typeid(targets_conditional_variable) == typeid(*obj))
{
targets_conditional_variable* ptr =
dynamic_cast<targets_conditional_variable*>(obj);
ss << "targets_conditional_variable(" << ptr->q << co << ptr->w << co
<< ptr->e << co << ptr->r;
}
else if (typeid(variable_has_been_listed) == typeid(*obj))
{
variable_has_been_listed* ptr =
dynamic_cast<variable_has_been_listed*>(obj);
ss << "variable_has_been_listed(" << ptr->q << co << ptr->w;
}
else if (typeid(resetstyle) == typeid(*obj))
{
resetstyle* ptr = dynamic_cast<resetstyle*>(obj);
ss << "resetstyle(" << ptr->q;
}
else if (typeid(checkstyle) == typeid(*obj))
{
checkstyle* ptr = dynamic_cast<checkstyle*>(obj);
ss << "checkstyle(" << ptr->q;
}
else if (typeid(total_local_entry) == typeid(*obj) )
{
total_local_entry* ptr = dynamic_cast<total_local_entry*>(obj);
ss << "total_local_entry(" << ptr->q;
}
else if (typeid(complex_next_state_operation_depth) == typeid(*obj))
{
complex_next_state_operation_depth* ptr =
dynamic_cast<complex_next_state_operation_depth*>(obj);
ss << "complex_next_state_operation_depth(" << ptr->q;
}
```

```
else if (typeid(output_filename) == typeid(*obj))
{
output_filename* ptr = dynamic_cast<output_filename*>(obj);
ss << "output_filename(" << ptr->q;
}
else if (typeid(hdl_io_pass) == typeid(*obj))
{
hdl_io_pass* ptr = dynamic_cast<hdl_io_pass*>(obj);
ss << "hdl_io_pass(" << ptr->q;
}
else if (typeid(current_hdl_style) == typeid(*obj))
{
current_hdl_style* ptr = dynamic_cast<current_hdl_style*>(obj);
ss << "current_hdl_style(" << ptr->q;
}
else if (typeid(call_ios_have_been_reset) == typeid(*obj))
{
call_ios_have_been_reset* ptr =
dynamic_cast<call_ios_have_been_reset*>(obj);
ss << "call_ios_have_been_reset(" << ptr->q;
}
else if (typeid(debug_mode) == typeid(*obj))
{
debug_mode* ptr = dynamic_cast<debug_mode*>(obj);
ss << "debug_mode(" << ptr->q;
}
else if (typeid(print_C_main_body) == typeid(*obj))
{
print_C_main_body* ptr = dynamic_cast<print_C_main_body*>(obj);
ss << "print_C_main_body(" << ptr->q;
}
else if (typeid(cac_mode) == typeid(*obj))
{
cac_mode* ptr = dynamic_cast<cac_mode*>(obj);
ss << "cac_mode(" << ptr->q;
}
else if (typeid(path) == typeid(*obj))
{
path* ptr = dynamic_cast<path*>(obj);
ss << "path(" << ptr->q << co << ptr->w << co << ptr->e;
}
```

```
else if (typeid(max_path) == typeid(*obj))
{
max_path* ptr = dynamic_cast<max_path*>(obj);
ss << "max_path(" << ptr->q << co << ptr->w;
}
else if (typeid(min_path) == typeid(*obj))
{
min_path* ptr = dynamic_cast<min_path*>(obj);
ss << "min_path(" << ptr->q << co << ptr->w;
}
else if (typeid(op_belongs_to_state) == typeid(*obj))
{
op_belongs_to_state* ptr = dynamic_cast<op_belongs_to_state*>(obj);
ss << "op_belongs_to_state(" << ptr->q << co << ptr->w << co << ptr->e
<< co << ptr->r;
}
else if (typeid(top_module) == typeid(*obj))
{
top_module* ptr = dynamic_cast<top_module*>(obj);
ss << "top_module(" << ptr->q;
}
else if (typeid(package_name) == typeid(*obj))
{
package_name* ptr = dynamic_cast<package_name*>(obj);
ss << "package_name(" << ptr->q;
}
else
{
GenfactError* ptr = new GenfactError("error for ");
ptr->saying += typeid(*obj).name();
ss << "Error at fact " << ptr->saying;
}
} // if (flag)
ss << pacl;
ALine = ss.str();
return ALine;
}

/// @brief makes data of string
/// @param inputline <- the string
GeneralFact* makeInstanceOf(string inputline)
```

```
{
string          ALine, subl;
size_t          pos, brpos;
vector<int>vq, vw, ve, vr, vt, vy;
vector<local_object>vobj;

pos = 0;
ALine = inputline.substr(0, inputline.find(pa, 0));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

if (ALine == "type_def")
{
int q, e, t, u, i, o;
string w, r, y;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
o = stoi(ALine);
return new type_def(q, e, t, u, i, o, w, r, y);
}
else if (ALine == "op_def")
{
int q, r, t, y, u;
string w, e;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);
return new op_def(q, r, t, y, u, w, e);
}
else if (ALine == "hierarchy_part")
{
int q, e, t, y, u;
string w, r;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);
return new hierarchy_part(q, e, t, y, u, w, r);
}
else if (ALine == "data_stmt")
{
int e, r;
string q, w, t, y;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size()-1);
y = ALine;
return new data_stmt(e, r, q, w, t, y);
}
else if (ALine == "prog_stmt")
{
int w, e, r, t, y, u, i;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
i = stoi(ALine);
return new prog_stmt(w, e, r, t, y, u, i, q);
}
else if (ALine == "joint_stmt")
{
int w, e, r, t, y;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
y = stoi(ALine);
return new joint_stmt(w, e, r, t, y, q);
}
else if (ALine == "call_stmt")
{
vq.clear();
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, brpos);
```

```
vq.push_back(stoi(ALine));
}
return new call_stmt(w, e, q, vq);
//for (int i : vq)
// cout << i << co;
}
else if (ALine == "compo_stmt")
{
vq.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
}
return new compo_stmt(w, q, vq);
}
else if (ALine == "rec_stmt")
```

```
{
vq.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
}
return new rec_stmt(w, q, vq);
}
else if (ALine == "special_op")
{
int w, r, t, y, i, o, p;
string q, e, u;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);
return new special_op(w, r, t, y, i, o, p, q, e, u);
}
else if (ALine == "special_dt")
{
int w, r;
string q, e, t, y, u;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
ALine.resize(ALine.size() - 1);
u = ALine;
return new special_dt(w, r, q, e, t, y, u);
}
else if (ALine == "local_object")
{
int w, i, t;
string q, e, r, y, u, o;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
return new local_object(t, w, i, q, e, y, u, r, o);
}
else if (ALine == "state_node")
{
vq.clear();
int w, e, r;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(pa, 0));
pos += ALine.size() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);
if (ALine == "subprogram_call")
{
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
```

```
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl,0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(pacl, 0));
e = stoi(ALine);
return new subprogram_call(e, vq, w, q);
}
else if (ALine == "dataflow")
{
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(pacl, 0));
e = stoi(ALine);
```

```
return new dataflow(e, vq, w, q);
}
else if (ALine == "ifthen")
{
vw.clear();
ve.clear();

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
```

```
vw.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector3
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
ve.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++pos, inputline.length() - pos);

subl = inputline.substr(pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
pos += ALine.length();
return new ifthen(e, r, vq, vw, ve, w, q);
}
else if (ALine == "jump")
{
//vector
if ((subl.find(brcl, 0)) != 0)
```

```
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(pacl, 0));
e = stoi(ALine);
return new jump(e, vq, w, q);
}
else if (ALine == "return")
{
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}

return new return_cos(vq, w, q);
}
}
else if (ALine == "change_op_number")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new change_op_number(e, w, q);
}
else if (ALine == "last_change_op_number")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
e = stoi(ALine);

return new last_change_op_number(e, w, q);
}
else if (ALine == "op_guards")
{
vq.clear();
vw.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);
```

```
//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}

return new op_guards(w, q, vq, vw);
}
else if (ALine == "var_guards")
{
vq.clear();
vw.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
```

```
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}

return new var_guards(w, q, vq, vw);
}
else if (ALine == "guard_pair")
{
int w, e;
string q, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
r = ALine;

return new guard_pair(w, e, q, r);
}
else if (ALine == "guard_cond")
{
int w, e;
string q, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
```

```
r = ALine;

return new guard_cond(w, e, q, r);
}
else if (ALine == "predecessors")
{
vq.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
return new predecessors(w, q, vq);
}
else if (ALine == "cessor")
{
```

```
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
return new cessor(w, e, q);
}
else if (ALine == "cessor_kind")
{
vq.clear();
int e, r;
string q, w, t, y;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
return new cessor_kind(e, r, q, w, t, y, vq);

}
else if (ALine == "old_schedule")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
return new old_schedule(q);
}
else if (ALine == "new_schedule")
```

```
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new new_schedule(q);
}
else if (ALine == "local_ifthen_chain_end_operations_were_written")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new local_ifthen_chain_end_operations_were_written(q);
}
else if (ALine == "calls_list")
{
vq.clear();
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
```

```
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
return new calls_list(q, w, e, vq);
}
else if (ALine == "composites_list")
{
vq.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
return new composites_list(q, w, vq);
}
else if (ALine == "record_aggregates_list")
{
vq.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
```

```
return new record_aggregates_list(q, w, vq);
}
else if (ALine == "mem_port")
{
int q, y, i, s;
string w, e, r, t, u, o, p, a, d;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
o = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
p = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
a = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
d = ALine;

return new mem_port(q, w, e, r, t, y, u, i, o, p, a, s, d);
}
else if (ALine == "global_declarations")
{
vobj.clear();
bool flag2 = 1;
int w, t, i, p;
string q, r, e, y, u, o;

subl = inputline.substr(++pos, inputline.length() - pos);
```

```
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (flag2)
{
ALine = subl.substr(13, subl.find(co, 0) - 13);
q = ALine;
pos += ALine.length() + 13;
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

if (inputline.find(co, pos) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
vobj.push_back(local_object(t, w, i, q, e, y, u, r, o));
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);
}
else
{
flag2 = 0;
ALine = subl.substr(0, subl.find(brcl, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
vobj.push_back(local_object(t, w, i, q, e, y, u, r, o));
pos += ALine.length();
subl = inputline.substr(++++++pos, inputline.length() - pos);
}
}
}
else subl = inputline.substr(++++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);
vobj.shrink_to_fit();
return new global_declarations(vobj, p);
}
else if (ALine == "source_is_normal_dt")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new source_is_normal_dt(q, w, e);
}
else if (ALine == "combo")
{
int q, e;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new combo(q, w, e);
}
else if (ALine == "sequence")
{
int q, e;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new sequence(q, w, e);
}
else if (ALine == "for_loop")
{
int q, e, r, t, y, u, i, o, p, a, s, d;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
a = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
d = stoi(ALine);

return new for_loop(q, w, e, r, t, y, u, i, o, p, a, s, d);
}
else if (ALine == "last_for_loop_entry")
{
```

```
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new last_for_loop_entry(q);
}
else if (ALine == "while_loop")
{
int q, e, r, t, y, u;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);

return new while_loop(q, w, e, r, t, y, u);
}
else if (ALine == "last_while_loop_entry")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new last_for_loop_entry(q);
}
else if (ALine == "possible_end_if")
{
vq.clear();
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
```

```
return new possible_end_if(q, vq);
}
else if (ALine == "end_if")
{
vq.clear();
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
return new end_if(q, vq);
}
else if (ALine == "nested_cond_fact")
{
vq.clear();
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
}
return new nested_cond_fact(q, vq);
}
else if (ALine == "top_level_call")
{
int q, w, r, y, i, p;
string e, t, u, o, a;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
o = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(pacl, 0));
a = ALine;

return new top_level_call(q, w, e, r, t, y, u, i, o, p, a);
}
else if (ALine == "top_level_call_parcs")
{
int q, w, r, y, i, p;
string e, t, u, o, a;
```

```
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
o = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(pacl, 0));
a = ALine;

return new top_level_call_parcs(q, w, e, r, t, y, u, i, o, p, a);
}
else if (ALine == "added_aux_call_ios")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new added_aux_call_ios(q, w);
}
else if (ALine == "added_aux_call_ios1")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new added_aux_call_ios1(q, w, e);
}
else if (ALine == "added_aux_call_signals")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new added_aux_call_signals(q, w);
}
else if (ALine == "found_call_operator")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new found_call_operator(q, w);
}
else if (ALine == "added_verilog_aux_call_outputs")
{
```

```
int w;
string q, e;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
e = ALine;

return new added_verilog_aux_call_outputs(q, w, e);
}
else if (ALine == "raw_dependencies")
{
vq.clear();
vw.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
```

```
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}
return new raw_dependencies(q, w, vq, vw);
}
else if (ALine == "war_dependencies")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
```

```
vw.push_back(stoi(ALine));
}
return new war_dependencies(q, w, vq, vw);
}
else if (ALine == "waw_dependencies")
{
vq.clear();
vw.clear();
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);
```

```
//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}
return new waw_dependencies(q, w, vq, vw);
}
else if (ALine == "schedule")
{
int w, r;
string q, e;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
```

```
return new schedule(q, w, e, r);
}
else if (ALine == "last_conditional_execution")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
return new last_conditional_execution(q, w);
}
else if (ALine == "conditional_operations")
{
int e, r;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);
```

```
//vector3
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
ve.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);


//vector4
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vr.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vr.push_back(stoi(ALine));
}

return new conditional_operations(q, w, e, r, vq, vw, ve, vr);
}
else if (ALine == "last_conditional_transition_of_schedule")
```

```
{
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new last_conditional_transition_of_schedule(q, w, e);
}
else if (ALine == "transition_to_be_rescheduled")
{
int e, t;
string q, w, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.find(co, 0));
```

```
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
t = stoi(ALine);

return new transition_to_be_rescheduled(w, e, q, r, t);
}
else if (ALine == "last_conditional_transition")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new last_conditional_transition(q, w);
}
else if (ALine == "conditional_transitions")
{
int e, r, t, y, u, i;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
i = stoi(ALine);

return new conditional_transitions(q, w, e, r, t, y, u, i);
}
else if (ALine == "state")
{
vq.clear();
vw.clear();
int e, t, y;
string q, w, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}

return new state(q, w, e, r, t, y, vq, vw);
}
else if (ALine == "rescheduled")
{
int e, t;
string q, w, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
t = stoi(ALine);

return new rescheduled(w, e, q, r, t);
}
else if (ALine == "last_rescheduled")
{
int e, t;
string q, w, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
t = stoi(ALine);

return new last_rescheduled(w, e, q, r, t);
}
```

```
else if (ALine == "raw_cessor")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new raw_cessor(q, w, e);
}
else if (ALine == "war_cessor")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new war_cessor(q, w, e);
}
else if (ALine == "waw_cessor")
```

```
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new waw_cessor(q, w, e);
}
else if (ALine == "op_resource")
{
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new op_resource(q, w, e);
}
else if (ALine == "global_resource")
{
```

```
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new global_resource(q);
}
else if (ALine == "module_g_resource")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new module_g_resource(q, w);
}
else if (ALine == "cf_previous_op")
{
int w, e, r;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);

return new cf_previous_op(q, w, e, r);
}
else if (ALine == "cf_previous_state")
{
int w, e, r;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);

return new cf_previous_state(q, w, e, r);
}
else if (ALine == "pred_candidate_examined")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new pred_candidate_examined(q, w, e);
}
else if (ALine == "reentrant_triangle")
{
int w, e, r, a, s;
string q, d, f;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
```

```
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector3
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
```

```
ve.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
ve.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector4
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vr.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vr.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector5
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vt.push_back(stoi(ALine));
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vt.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector6
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vy.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vy.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
a = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
d = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 1);
f = ALine;

return new reentrant_triangle(q, w, e, r, vq, vw, ve, vr, vt,
vy, a, s, d, f);
}
else if (ALine == "last_reentrant_triangle")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
return new last_reentrant_triangle(q, w);
}
else if (ALine == "last_schedule_state")
{
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new last_schedule_state(q, w, e);
}
else if (ALine == "conditional_incomplete")
{
int e, r, a, s, d, f, g, h, j, k, l, z, x;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector3
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
```

```
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
ve.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector4
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vr.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vr.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector5
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vt.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
```

```
}
ALine = subl.substr(0, subl.find(brcl, 0));
vt.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector6
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vy.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vy.push_back(stoi(ALine));
pos += ALine.length() + 1; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
a = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
d = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
f = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
g = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
h = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
j = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
k = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
l = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
z = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
x = stoi(ALine);

return new conditional_incomplete(q, w, e, r, vq, vw, ve, vr, vt,
```

```
vy, a, s, d, f, g, h, j, k, l, z, x);
}
else if (ALine == "mixed_incomplete_state_lists")
{
vq.clear();
vw.clear();
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length()+1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}

return new mixed_incomplete_state_lists(w, e, q, vq, vw);
}
else if (ALine == "linear_incomplete_node")
{
vq.clear();
vw.clear();
int e, y;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
ALine = subl.substr(0, subl.find(brcl, 0));
vq.push_back(stoi(ALine));
pos += ALine.length() + 2; // getting after bracket
subl = inputline.substr(++pos, inputline.length() - pos);
}
else subl = inputline.substr(++++++pos, inputline.length() - pos);

//vector2
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
brpos -= ALine.length();
brpos--;
}
```

```
ALine = subl.substr(0, subl.find(brcl, 0));
vw.push_back(stoi(ALine));
}

return new linear_incomplete_node(w, e, q, y, vq, vw);
}
else if (ALine == "incomplete_links")
{
int q, w, e, r, t, y, u, i, o, p;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);

return new incomplete_links(q, w, e, r, t, y, u, i, o, p);
}
else if (ALine == "last_incomplete")
{
int q, w, e;
string r;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
```

```
r = ALine;

return new last_incomplete(q, w, e, r);
}
else if (ALine == "global_nils")
{
vector<nil_node>vnil;
nil_node nn(0,"");
bool flag2 = 1;
subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (flag2)
{

ALine = subl.substr(9, subl.find(co, 0) - 9);
nn.q = stoi(ALine);
pos += ALine.length() + 9;
subl = inputline.substr(++pos, inputline.length() - pos);

if (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 1);
nn.w = ALine;
vnil.push_back(nn);
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);
}
else
{
flag2 = 0;
subl.resize(subl.size() - 3);
nn.w = subl;
vnil.push_back(nn);
}
}
}
```

```
return new global_nils(vnil);
}
else if (ALine == "current_module")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new current_module(q);
}
else if (ALine == "last_linear_incomplete_node")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new last_linear_incomplete_node(q);
}
else if (ALine == "operator_instances")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new operator_instances(q, w);
}
else if (ALine == "massively_parallel_style")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
q = stoi(ALine);

return new massively_parallel_style(q);
}
else if (ALine == "hdl_style")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new hdl_style(q);
}
else if (ALine == "op_instance")
{
int q, e, r;
string w, t;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
ALine.resize(ALine.size() - 1);
t = ALine;

return new op_instance(w, e, q, r, t);
}
else if (ALine == "last_op_instance")
{
int q, e;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
return new last_op_instance(q, w, e);
}
else if (ALine == "op_in_a_state")
{
int q, r;
string w, e, t;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
t = ALine;

return new op_in_a_state(w, e, q, r, t);
}
else if (ALine == "last_op_in_a_state")
{
int q, r;
string w, e;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
return new last_op_in_a_state(q, w, e, r);
}
```

```
else if (ALine == "signal_instance")
{
int q, t, y, u;
string w, e, r;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);

return new signal_instance(q, w, e, r, t, y, u);
}
else if (ALine == "last_signal_instance")
```

```
{
int q;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
w = ALine;

return new last_signal_instance(q, w);
}
else if (ALine == "output_instance")
{
int q, t, y;
string w, e, r;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.rfind(pacl, 0));
y = stoi(ALine);

return new output_instance(q, w, e, r, t, y);
}
else if (ALine == "last_output_instance")
{
int q;
string w;

ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
w = ALine;

return new last_output_instance(q, w);
}
else if (ALine == "operator_instance_stats")
{
int w, e, r;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);


ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);

return new operator_instance_stats(q, w, e, r);
}
else if (ALine == "consecutive_106")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new consecutive_106(q);
}
else if (ALine == "operation_order")
{
int e, r, t, y, u, i, o, p;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);

return new operation_order(q, w, e, r, t, y, u, i, o, p);
}
else if (ALine == "max_parallel_call_order")
{
int e, r;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
```

```
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);

return new max_parallel_call_order(q, w, e, r);
}
else if (ALine == "max_op_order")
{
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new max_op_order(q, w, e);
}
else if (ALine == "totalmax_call_order")
{
```

```
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new totalmax_call_order(q, w, e);
}
else if (ALine == "totalmax_gross_depth")
{
int e;
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new totalmax_gross_depth(q, w, e);
}
else if (ALine == "current_total_max_order_entry")
{
int q;
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new current_total_max_order_entry(q);
}
else if (ALine == "module_last_state")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new module_last_state(q);
}
else if (ALine == "module_local_list")
{
vobj.clear();
bool flag2 = 1;
int w, t, i;
string q, r, e, y, u, o;

subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (flag2)
{
ALine = subl.substr(13, subl.find(co, 0) - 13);
q = ALine;
pos += ALine.length() + 13;
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
```

```
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

if (inputline.find(co, pos) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
vobj.push_back(local_object(t, w, i, q, e, y, u, r, o));
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);
}
else
{
flag2 = 0;
```

```
ALine = subl.substr(0, subl.find(brcl, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
vobj.push_back(local_object(t, w, i, q, e, y, u, r, o));
pos += ALine.length();
}
}
}
return new module_local_list(vobj);
}
//exeeded the limit of nested else if
if (ALine == "module_local_list_parcs")
{
vobj.clear();
bool flag2 = 1;
int w, t, i;
string q, r, e, y, u, o;

subl = inputline.substr(++pos, inputline.length() - pos);

//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (flag2)
{
ALine = subl.substr(13, subl.find(co, 0) - 13);
q = ALine;
pos += ALine.length() + 13;
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

if (inputline.find(co, pos) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
vobj.push_back(local_object(t, w, i, q, e, y, u, r, o));
pos += ALine.length() + 1;
subl = inputline.substr(++pos, inputline.length() - pos);
}
else
{
flag2 = 0;
ALine = subl.substr(0, subl.find(brcl, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
vobj.push_back(local_object(t, w, i, q, e, y, u, r, o));
```

```
pos += ALine.length();
}
}
}
return new module_local_list_parcs(vobj);
}
else if (ALine == "last_non_io_found")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new last_non_io_found(q);
}
else if (ALine == "last_local_number")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new last_local_number(q);
}
else if (ALine == "printed_formal_ios_of_called_module")
{
int w, e;
string q, r;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
r = ALine;

return new printed_formal_ios_of_called_module(q, w, e, r);
}
else if (ALine == "it_includes_ifthen")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new it_includes_ifthen(q, w, e);
}
else if (ALine == "it_includes_conditional_targeting")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new it_includes_conditional_targeting(q, w, e);
}
else if (ALine == "targets_conditional_variable")
{
int w, e, r;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);

return new targets_conditional_variable(q, w, e, r);
}
else if (ALine == "variable_has_been_listed")
{
string q, w;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
w = ALine;

return new variable_has_been_listed(q, w);
}
else if (ALine == "resetstyle")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new resetstyle(q);
}
else if (ALine == "checkstyle")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new checkstyle(q);
}
else if (ALine == "total_local_entry")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new total_local_entry(q);
}
else if (ALine == "complex_next_state_operation_depth")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
q = stoi(ALine);

return new complex_next_state_operation_depth(q);
}
else if (ALine == "output_filename")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new output_filename(q);
}
else if (ALine == "hdl_io_pass")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new hdl_io_pass(q);
}
else if (ALine == "current_hdl_style")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new current_hdl_style(q);
}
else if (ALine == "call_ios_have_been_reset")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
```

```
return new call_ios_have_been_reset(q);
}
else if (ALine == "debug_mode")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new debug_mode(q);
}
else if (ALine == "print_C_main_body")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new print_C_main_body(q);
}
else if (ALine == "cac_mode")
{
int q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);

return new cac_mode(q);
}
else if (ALine == "path")
{
int w, e;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);

return new path(q, w, e);
}
else if (ALine == "max_path")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new max_path(q, w);
}
else if (ALine == "min_path")
{
int w;
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);

return new min_path(q, w);
}
else if (ALine == "op_belongs_to_state")
{
int w, e, r;
```

```
string q;

ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);

ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);

return new op_belongs_to_state(q, w, e, r);
}
else if (ALine == "top_module")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;

return new top_module(q);
}
else if (ALine == "package_name")
{
string q;

ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
return new package_name(q);
}
```

```
return new GenfactError(inputline);
}

/// @brief makes data with parameters until "*" of string
/// @param inputline <- the string
factstar makefactstar (string inputline)
{
int params = 0;
string ALine, subl;
size_t pos, brpos;
bool flag = 1, stop = 1;

GeneralFact* ptr = nullptr;

pos = 0;
ALine = inputline.substr(0, inputline.find(pa, 0));
pos += ALine.length();

if (ALine == "type_def")
{
flag = 0;
int q{}, e{}, t{}, u{}, i{}, o{};
string w{}, r{}, y{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
o = stoi(ALine);
}
}
ptr = new type_def(q, e, t, u, i, o, w, r, y);
}
else if (ALine == "op_def")
{
flag = 0;
int q{}, r{}, t{}, y{}, u{};
string w{}, e{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);
}
}
ptr = new op_def(q, r, t, y, u, w, e);
}
else if (ALine == "hierarchy_part")
{
flag = 0;
int q{}, e{}, t{}, y{}, u{};
string w{}, r{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);
}
}
ptr = new hierarchy_part(q, e, t, y, u, w, r);
}
else if (ALine == "data_stmt")
{
flag = 0;
int e{}, r{};
string q{}, w{}, t{}, y{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
y = ALine;
}
}
ptr = new data_stmt(e, r, q, w, t, y);
}
else if (ALine == "prog_stmt")
{
flag = 0;
int w{}, e{}, r{}, t{}, y{}, u{}, i{};
string q{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
```

```
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
i = stoi(ALine);
}
}
ptr = new prog_stmt(w, e, r, t, y, u, i, q);
}
```

```
else if (ALine == "joint_stmt")
{
flag = 0;
int w{}, e{}, r{}, t{}, y{};
string q{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
```

```
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
y = stoi(ALine);
}
}
ptr = new joint_stmt(w, e, r, t, y, q);
}
else if (ALine == "call_stmt")
{
flag = 0;
int w{}, e{};
string q{};
vector<int>vq;
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
```

```
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length()
- pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new call_stmt(w, e, q, vq);
}
else if (ALine == "compo_stmt")
{
flag = 0;
int w{};
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length()
- pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new compo_stmt(w, q, vq);
}
```

```
else if (ALine == "rec_stmt")
{
flag = 0;
int w{};
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
```

```
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length()
- pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new rec_stmt(w, q, vq);
}
else if (ALine == "special_op")
{
flag = 0;
int w{}, r{}, t{}, y{}, i{}, o{}, p{};
string q{}, e{}, u{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
```

```
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);
}
}
ptr = new special_op(w, r, t, y, i, o, p, q, e, u);
}
else if (ALine == "special_dt")
{
flag = 0;
int w{}, r{};
string q{}, e{}, t{}, y{}, u{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
u = ALine;
}
}
ptr = new special_dt(w, r, q, e, t, y, u);
}
else if (ALine == "local_object")
{
flag = 0;
int w{}, i{}, t{};
string q{}, e{}, r{}, y{}, u{}, o{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
o = ALine;
}
}
ptr = new local_object(t, w, i, q, e, y, u, r, o);
}
else if (ALine == "state_node")
{
flag = 0; // use do while
int w{}, e{}, r{};
string q{};
vector<int>vq, vw, ve;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(pa, 0));
if (ALine == "subprogram_call")
{
pos += ALine.size();
subl = inputline.substr(++pos, inputline.length()
- pos);
//vector
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length()
- pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
pos += ALine.length();
vq.push_back(stoi(ALine));
}
subl = inputline.substr(++++pos, inputline.length()
- pos);
}
```

```
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(pacl, 0));
e = stoi(ALine);
}
}
ptr = new subprogram_call(e, vq, w, q);
}
else if (ALine == "dataflow")
{
pos += ALine.size();
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length()
- pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
pos += ALine.length();
vq.push_back(stoi(ALine));
}
subl = inputline.substr(++++pos,
```

```
inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(pacl, 0));
e = stoi(ALine);
}
}
ptr = new dataflow(e, vq, w, q);
}
else if (ALine == "ifthen")
{
pos += ALine.size();
subl = inputline.substr(++pos, inputline.length()
- pos);

//vector
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length()
- pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
```

```
pos += ALine.length();
vq.push_back(stoi(ALine));
}
subl = inputline.substr(++++pos,
inputline.length() - pos);
}
else stop = 0;

//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0,
subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos,
inputline.length() - pos);
}
else stop = 0;
}

//vector3
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0,
subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
ve.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos,
inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
subl = inputline.substr(pos,
inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
pos += ALine.length();
}
}
ptr = new ifthen(e, r, vq, vw, ve, w, q);
}
else if (ALine == "jump")
{
pos += ALine.size();
subl = inputline.substr(++pos,
inputline.length() - pos);
//vector
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0,
subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
```

```
}
ALine = subl.substr(0, brpos);
pos += ALine.length();
vq.push_back(stoi(ALine));
}
subl = inputline.substr(++++pos,
inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(pacl, 0));
e = stoi(ALine);
}
}
ptr = new jump(e, vq, w, q);
}
else if (ALine == "return")
{
pos += ALine.size();
subl = inputline.substr(++pos,
inputline.length() - pos);
//vector
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{

ALine = subl.substr(0,
subl.find(co, 0));
vq.push_back(stoi(ALine));
```

```
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
pos += ALine.length() + 1;
vq.push_back(stoi(ALine));
}
}
ptr = new return_cos(vq, w, q);
}
}
else ptr = new state_node(w, q);
}
else
{
ptr = new state_node(w, q);
}
}
else if (ALine == "change_op_number")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new change_op_number(e, w, q);
}
else if (ALine == "last_change_op_number")
{
flag = 0;
int w{}, e{};
string q{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new last_change_op_number(e, w, q);
}
else if (ALine == "op_guards")
{
flag = 0;
int w{};
string q{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos,
inputline.length() - pos);
}
else stop = 0;
}
```

```
//vector2
if(stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new op_guards(w, q, vq, vw);
}
else if (ALine == "var_guards")
{
flag = 0;
int w{};
string q{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
```

```
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos,
inputline.length() - pos);
}
else stop = 0;
}

//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0,
subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new var_guards(w, q, vq, vw);
}
```

```
else if (ALine == "guard_pair")
{
flag = 0;
int w{}, e{};
string q{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
```

```
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
r = ALine;
}
}
ptr = new guard_pair(w, e, q, r);
}
else if (ALine == "guard_cond")
{
flag = 0;
int w{}, e{};
string q{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
r = ALine;
}
}
ptr = new guard_cond(w, e, q, r);
}
else if (ALine == "predecessors")
{
flag = 0;
int w{};
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos,
inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos,
inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
```

```
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new predecessors(w, q, vq);
}
else if (ALine == "cessor")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
}
else stop = 0;
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new cessor(w, e, q);
}
else if (ALine == "cessor_kind")
{
flag = 0;
int e{}, r{};
string q{}, w{}, t{}, y{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new cessor_kind(e, r, q, w, t, y, vq);
}
else if (ALine == "old_schedule")
{
flag = 0;
string q{};
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new old_schedule(q);
}
else if (ALine == "new_schedule")
{
flag = 0;
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new new_schedule(q);
}
else if (ALine == "local_ifthen_chain_end_operations_were_written")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = stoi(ALine);
```

```
}
ptr = new local_ifthen_chain_end_operations_were_written(q);
}
else if (ALine == "calls_list")
{
flag = 0;
int w{}, e{};
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new calls_list(q, w, e, vq);
}
else if (ALine == "composites_list")
{
flag = 0;
int w{};
string q{};
vector<int>vq;
```

```
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
```

```
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new composites_list(q, w, vq);
}
else if (ALine == "record_aggregates_list")
{
flag = 0;
int w{};
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
```

```
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos,
inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new record_aggregates_list(q, w, vq);
}
else if (ALine == "mem_port")
{
flag = 0;
int q{}, y{}, i{}, s{};
string w{}, e{}, r{}, t{}, u{}, o{}, p{}, a{}, d{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
p = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
a = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
d = ALine;
}
}
ptr = new mem_port(q, w, e, r, t, y, u, i, o, p, a, s, d);
}
else if (ALine == "global_declarations")
{
flag = 0;
int w{}, t{}, i{}, p{};
string q{}, e{}, r{}, y{}, u{}, o{};
vector<local_object>vobj;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
```

```
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos && stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
pos += 13;
brpos -= 13;
subl = inputline.substr(pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
if (subl.find(co, 0) < brpos) //last parameter
{
ALine = subl.substr(0, subl.find(co, 0));
pos += ALine.length();
brpos -= ALine.length();
ALine.resize(ALine.size() - 1);
o = ALine;
pos++;
brpos--;
}
else //last
{
ALine = subl.substr(0, subl.find(co, 0));
pos += ALine.length();
ALine.resize(ALine.size() - 2);
brpos -= ALine.length();
o = ALine;
}
vobj.push_back(*(new local_object(t, w, i, q, e, y, u, r, o)));
}
else stop = 0;
}
brpos--;
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else
{
params++;
pos += 2;
subl = inputline.substr(pos, inputline.length() - pos);
}
}
else stop = 0;

if (stop)
{
if (subl.find(us, 0) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
}
}
ptr = new global_declarations(vobj, p);
}
else if (ALine == "source_is_normal_dt")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new source_is_normal_dt(q, w, e);
}
else if (ALine == "combo")
{
flag = 0;
int q{}, e{};
string w{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new combo(q, w, e);
}
else if (ALine == "sequence")
{
flag = 0;
int q{}, e{};
string w{};
subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new sequence(q, w, e);
}
else if (ALine == "for_loop")
{
flag = 0;
int q{}, e{}, r{}, t{}, y{}, u{}, i{}, o{}, p{}, a{}, s{}, d{};
string w{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
```

```
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
a = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
d = stoi(ALine);
}
}
ptr = new for_loop(q, w, e, r, t, y, u, i, o, p, a, s, d);
}
else if (ALine == "last_for_loop_entry")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new last_for_loop_entry(q);
}
else if (ALine == "while_loop")
{
flag = 0;
int q{}, e{}, r{}, t{}, y{}, u{};
string w{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);
}
}
ptr = new while_loop(q, w, e, r, t, y, u);
}
else if (ALine == "last_while_loop_entry")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new last_for_loop_entry(q);
}
else if (ALine == "possible_end_if")
{
flag = 0;
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
```

```
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new possible_end_if(q, vq);
}
else if (ALine == "end_if")
{
flag = 0;
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
```

```
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new end_if(q, vq);
}
else if (ALine == "nested_cond_fact")
{
flag = 0;
string q{};
vector<int>vq;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;

if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{

subl = inputline.substr(++pos, inputline.length() - pos);
params++;
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
pos++;
}
}
}
ptr = new nested_cond_fact(q, vq);
}
else if (ALine == "top_level_call")
{
flag = 0;
int q{}, w{}, r{}, y{}, i{}, p{};
string e{}, t{}, u{}, o{}, a{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
a = ALine;
}
}
ptr = new top_level_call(q, w, e, r, t, y, u, i, o, p, a);
}
```

```
else if (ALine == "top_level_call_parcs")
{
flag = 0;
int q{}, w{}, r{}, y{}, i{}, p{};
string e{}, t{}, u{}, o{}, a{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
p = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
a = ALine;
}
}
ptr = new top_level_call_parcs(q, w, e, r, t, y, u, i, o, p, a);
}
else if (ALine == "added_aux_call_ios")
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new added_aux_call_ios(q, w);
```

```
}
else if (ALine == "added_aux_call_ios1")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new added_aux_call_ios1(q, w, e);
}
```

```
else if (ALine == "added_aux_call_signals")
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new added_aux_call_signals(q, w);
}
else if (ALine == "found_call_operator")
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new found_call_operator(q, w);
}
else if (ALine == "added_verilog_aux_call_outputs")
{
flag = 0;
int w{};
string q{}, e{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
e = ALine;
}
}
ptr = new added_verilog_aux_call_outputs(q, w, e);
}
else if (ALine == "raw_dependencies")
{
flag = 0;
int w{};
string q{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}


//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
```

```
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new raw_dependencies(q, w, vq, vw);
}
else if (ALine == "war_dependencies")
{
flag = 0;
int w{};
string q{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector2
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new war_dependencies(q, w, vq, vw);
}
else if (ALine == "waw_dependencies")
{
flag = 0;
int w{};
string q{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new waw_dependencies(q, w, vq, vw);
}
else if (ALine == "schedule")
{
flag = 0;
int w{}, r{};
string q{}, e{};
vector<int>vq, vw, ve, vr;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
}
}
ptr = new schedule(q, w, e, r);
}
else if (ALine == "last_conditional_execution")
```

```
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new last_conditional_execution(q, w);
}
else if (ALine == "conditional_operations")
{
flag = 0;
int e{}, r{};
string q{}, w{};
vector<int>vq, vw, ve, vr;

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
//vector
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
pos += ALine.length();
vq.push_back(stoi(ALine));
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
```

```
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector3
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
ve.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
```

```
}

//vector4
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vr.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vr.push_back(stoi(ALine));
pos += ALine.size();
}
}
else stop = 0;
}
ptr = new conditional_operations(q, w, e, r, vq, vw, ve, vr);
}
else if (ALine == "last_conditional_transition_of_schedule")
{
flag = 0;
int e{};
string q{}, w{};


subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new last_conditional_transition_of_schedule(q, w, e);
}
else if (ALine == "transition_to_be_rescheduled")
{
flag = 0;
int e{}, t{};
string q{}, w{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
```

```
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
t = stoi(ALine);
}
}
ptr = new transition_to_be_rescheduled(w, e, q, r, t);
}
else if (ALine == "last_conditional_transition")
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);

if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new last_conditional_transition(q, w);
}
else if (ALine == "conditional_transitions")
```

```
{
flag = 0;
int e{}, r{}, t{}, y{}, u{}, i{};
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
i = stoi(ALine);
}
}
ptr = new conditional_transitions(q, w, e, r, t, y, u, i);
}
else if (ALine == "state")
{
flag = 0;
int e{}, t{}, y{};
string q{}, w{}, r{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
```

```
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector2
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new state(q, w, e, r, t, y, vq, vw);
}
else if (ALine == "rescheduled")
{
flag = 0;
int e{}, t{};
string q{}, w{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
t = stoi(ALine);
}
}
ptr = new rescheduled(w, e, q, r, t);
}
else if (ALine == "last_rescheduled")
{
flag = 0;
int e{}, t{};
string q{}, w{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
t = stoi(ALine);
}
}
ptr = new last_rescheduled(w, e, q, r, t);
}
else if (ALine == "raw_cessor")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new raw_cessor(q, w, e);
}
else if (ALine == "war_cessor")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new war_cessor(q, w, e);
}
else if (ALine == "waw_cessor")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new waw_cessor(q, w, e);
}
else if (ALine == "op_resource")
{
flag = 0;
int e{};
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new op_resource(q, w, e);
}
else if (ALine == "global_resource")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new global_resource(q);
}
else if (ALine == "module_g_resource")
{
flag = 0;
int w{};
string q{};
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new module_g_resource(q, w);
}
else if (ALine == "cf_previous_op")
{
flag = 0;
int w{}, e{}, r{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new cf_previous_op(q, w, e, r);
}
else if (ALine == "cf_previous_state")
{
flag = 0;
int w{}, e{}, r{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new cf_previous_state(q, w, e, r);
}
```

```
else if (ALine == "pred_candidate_examined")
{
flag = 0;
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new pred_candidate_examined(q, w, e);
}
else if (ALine == "reentrant_triangle")
{
```

```
flag = 0;
int w{}, e{}, r{}, a{}, s{};
string q{}, d{}, f{};
vector<int>vq, vw, ve, vr, vt, vy;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
pos += ALine.length();
vq.push_back(stoi(ALine));
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector2
if (stop)
```

```
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector3
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
```

```
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
ve.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector4
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vr.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vr.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
//vector5
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vt.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vt.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector6
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
```

```
vy.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vy.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
a = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
d = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 1);
f = ALine;
}
}
ptr = new reentrant_triangle(q, w, e, r, vq, vw, ve, vr, vt, vy, a, s, d, f);
}
else if (ALine == "last_reentrant_triangle")
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new last_reentrant_triangle(q, w);
}
else if (ALine == "last_schedule_state")
{
flag = 0;
int e{};
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new last_schedule_state(q, w, e);
}
else if (ALine == "conditional_incomplete")
{
flag = 0;
int e{}, r{}, a{}, s{}, d{}, f{}, g{}, h{}, j{}, k{}, l{}, z{}, x{};
string q{}, w{};
vector<int>vq, vw, ve, vr, vt, vy;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
pos += ALine.length();
vq.push_back(stoi(ALine));
```

```
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector3
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
```

```
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
ve.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
ve.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector4
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vr.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
```

```
vr.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector5
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vt.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vt.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}

//vector6
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vy.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vy.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
a = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
s = stoi(ALine);
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
d = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
f = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
g = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
h = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
j = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
k = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
l = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
z = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
x = stoi(ALine);
}
}
ptr = new conditional_incomplete(q, w, e, r, vq, vw, ve, vr, vt, vy, a,
s, d, f, g, h, j, k, l, z, x);
}
else if (ALine == "mixed_incomplete_state_lists")
{
flag = 0;
int e{};
string q{}, w{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
```

```
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}


//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
```

```
pos += ALine.size();
}
}
}
ptr = new mixed_incomplete_state_lists(w, e, q, vq, vw);
}
else if (ALine == "linear_incomplete_node")
{
flag = 0;
int e{}, y{};
string q{}, w{};
vector<int>vq, vw;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}

if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vq.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vq.push_back(stoi(ALine));
pos += ALine.size();
```

```
}
subl = inputline.substr(++++pos, inputline.length() - pos);
}
else stop = 0;
}


//vector2
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
params++;
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos)
{
ALine = subl.substr(0, subl.find(co, 0));
vw.push_back(stoi(ALine));
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
ALine = subl.substr(0, brpos);
vw.push_back(stoi(ALine));
pos += ALine.size();
}
}
}
ptr = new linear_incomplete_node(w, e, q, y, vq, vw);
}
else if (ALine == "incomplete_links")
{
flag = 0;
int q{}, w{}, e{}, r{}, t{}, y{}, u{}, i{}, o{}, p{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);
}
}
ptr = new incomplete_links(q, w, e, r, t, y, u, i, o, p);
}
else if (ALine == "last_incomplete")
{
flag = 0;
int q{}, w{}, e{};
string r{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
r = ALine;
}
```

```
}
ptr = new last_incomplete(q, w, e, r);
}
else if (ALine == "global_nils")
{
flag = 0;
vector<nil_node>vnil{};
nil_node nn(0, "");
int iv = 0;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos && stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
pos += 9;  // 9 because global_nils([>>nil_node(<<_, \"true\")
brpos -= 9;
subl = inputline.substr(pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
nn.q = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
if (subl.find(co, 0) < brpos) //last parameters
{
ALine = subl.substr(0, subl.find(co, 0));
pos += ALine.length();
brpos -= ALine.length();
ALine.resize(ALine.size() - 1);
nn.w = ALine;
}
else //last
```

```
{
subl.resize(subl.size() - 2);
nn.w = subl;
}
vnil.push_back(nn);
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
iv++;
}
else stop = 0;
}
}
else params++;
}
ptr = new global_nils(vnil);
}
else if (ALine == "current_module")
{
flag = 0;
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new current_module(q);
}
else if (ALine == "last_linear_incomplete_node")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
q = stoi(ALine);
}
ptr = new last_linear_incomplete_node(q);
}
else if (ALine == "operator_instances")
{
flag = 0;
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new operator_instances(q, w);
}
else if (ALine == "massively_parallel_style")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
q = stoi(ALine);
}
ptr = new massively_parallel_style(q);
}
else if (ALine == "hdl_style")
{
flag = 0;
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new hdl_style(q);
}
else if (ALine == "op_instance")
{
flag = 0;
int q{}, e{}, r{};
string w{}, t{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
t = ALine;
}
}
ptr = new op_instance(w, e, q, r, t);
```

```
}
else if (ALine == "last_op_instance")
{
flag = 0;
int q{}, e{};
string w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new last_op_instance(q, w, e);
}
else if (ALine == "op_in_a_state")
```

```
{
flag = 0;
int q{}, r{};
string w{}, e{}, t{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
t = ALine;
}
}
ptr = new op_in_a_state(w, e, q, r, t);
}
else if (ALine == "last_op_in_a_state")
{
flag = 0;
int q{}, r{};
string w{}, e{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
```

```
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new last_op_in_a_state(q, w, e, r);
}
else if (ALine == "signal_instance")
{
flag = 0;
int q{}, t{}, y{}, u{};
string w{}, e{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
```

```
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
u = stoi(ALine);
}
}
ptr = new signal_instance(q, w, e, r, t, y, u);
}
else if (ALine == "last_signal_instance")
{
flag = 0;
int q{};
```

```
string w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
w = ALine;
}
}
ptr = new last_signal_instance(q, w);
}
else if (ALine == "output_instance")
{
flag = 0;
int q{}, t{}, y{};
string w{}, e{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
```

```
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
y = stoi(ALine);
}
}
ptr = new output_instance(q, w, e, r, t, y);
}
else if (ALine == "last_output_instance")
{
flag = 0;
int q{};
string w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
w = ALine;
```

```
}
}
ptr = new last_output_instance(q, w);
}
else if (ALine == "operator_instance_stats")
{
flag = 0;
int w{}, e{}, r{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
```

```
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new operator_instance_stats(q, w, e, r);
}
else if (ALine == "consecutive_106")
{
flag = 0;
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new consecutive_106(q);
}
else if (ALine == "operation_order")
{
flag = 0;
int e{}, r{}, t{}, y{}, u{}, i{}, o{}, p{};
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
r = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
```

```
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
y = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
u = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
```

```
i = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
o = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
p = stoi(ALine);
}
}
ptr = new operation_order(q, w, e, r, t, y, u, i, o, p);
}
else if (ALine == "max_parallel_call_order")
{
flag = 0;
int e{}, r{};
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
```

```
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new max_parallel_call_order(q, w, e, r);
}
else if (ALine == "max_op_order")
{
```

```
flag = 0;
int e{};
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new max_op_order(q, w, e);
}
else if (ALine == "totalmax_call_order")
{
flag = 0;
int e{};
```

```
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new totalmax_call_order(q, w, e);
}
else if (ALine == "totalmax_gross_depth")
{
flag = 0;
int e{};
string q{}, w{};
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new totalmax_gross_depth(q, w, e);
}
else if (ALine == "current_total_max_order_entry")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new current_total_max_order_entry(q);
}
else if (ALine == "module_last_state")
{
flag = 0;
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new module_last_state(q);
}
else if (ALine == "module_local_list")
{
flag = 0;
int iv = 0;
int w{}, t{}, i{};
string q{}, r{}, e{}, y{}, u{}, o{};
vector<local_object>vobj;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos && stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
pos += 13;
brpos -= 13;
subl = inputline.substr(pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
```

```
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
if (subl.find(co, 0) < brpos) //last parameter
{
ALine = subl.substr(0, subl.find(co, 0));
pos += ALine.length();
brpos -= ALine.length();
ALine.resize(ALine.size() - 1);
o = ALine;
pos++;
brpos--;
}
else //last
{
ALine = subl.substr(0, subl.find(co, 0));
ALine.resize(ALine.size() - 3);
o = ALine;
}
vobj.push_back(*(new local_object(t, w, i, q, e, y, u, r, o)));
}
else stop = 0;
}
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
}
ptr = new module_local_list(vobj);
}
if (flag) //exeeded the limit of nested else if
{
if (ALine == "module_local_list_parcs")
{
flag = 0;
```

```
int iv = 0;
int w{}, t{}, i{};
string q{}, r{}, e{}, y{}, u{}, o{};
vector<local_object>vobj;

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
subl = inputline.substr(++pos, inputline.length() - pos);
//vector
if ((subl.find(brcl, 0)) != 0)
{
brpos = subl.find(brcl, 0);
while (subl.find(co, 0) < brpos && stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
pos += 13;
brpos -= 13;
subl = inputline.substr(pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
e = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
r = ALine;
```

```
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
t = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
y = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
u = ALine;
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
ALine = subl.substr(0, subl.find(co, 0));
i = stoi(ALine);
pos += ALine.length();
brpos -= ALine.length();
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
if (subl.find(co, 0) < brpos) //last parameter
{
ALine = subl.substr(0, subl.find(co, 0));
pos += ALine.length();
brpos -= ALine.length();
ALine.resize(ALine.size() - 1);
o = ALine;
pos++;
brpos--;
}
else //last
{
ALine = subl.substr(0, subl.find(co, 0));
```

```
ALine.resize(ALine.size() - 3);
o = ALine;
}
vobj.push_back(*(new local_object(t, w, i, q, e, y, u, r, o)));
}
else stop = 0;
}
brpos--;
subl = inputline.substr(++pos, inputline.length() - pos);
}
}
ptr = new module_local_list_parcs(vobj);
}
else if (ALine == "last_non_io_found")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new last_non_io_found(q);
}
else if (ALine == "last_local_number")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new last_local_number(q);
}
else if (ALine == "printed_formal_ios_of_called_module")
{
```

```
int w{}, e{};
string q{}, r{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
```

```
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
r = ALine;
}
}
ptr = new printed_formal_ios_of_called_module(q, w, e, r);
}
else if (ALine == "it_includes_ifthen")
{
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
```

```
e = stoi(ALine);
}
}
ptr = new it_includes_ifthen(q, w, e);
}
else if (ALine == "it_includes_conditional_targeting")
{
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
```

```
ptr = new it_includes_conditional_targeting(q, w, e);
}
else if (ALine == "targets_conditional_variable")
{
int w{}, e{}, r{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
```

```
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new targets_conditional_variable(q, w, e, r);
}
else if (ALine == "variable_has_been_listed")
{
string q{}, w{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
w = ALine;
}
}
ptr = new variable_has_been_listed(q, w);
}
else if (ALine == "resetstyle")
{
string q{};
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new resetstyle(q);
}
else if (ALine == "checkstyle")
{
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new checkstyle(q);
}
else if (ALine == "total_local_entry")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new total_local_entry(q);
}
else if (ALine == "complex_next_state_operation_depth")
{
int q{};
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new complex_next_state_operation_depth(q);
}
else if (ALine == "output_filename")
{
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new output_filename(q);
}
else if (ALine == "hdl_io_pass")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new hdl_io_pass(q);
}
else if (ALine == "current_hdl_style")
{
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new current_hdl_style(q);
}
else if (ALine == "call_ios_have_been_reset")
{
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new call_ios_have_been_reset(q);
}
else if (ALine == "debug_mode")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new debug_mode(q);
}
else if (ALine == "print_C_main_body")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
```

```
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new print_C_main_body(q);
}
else if (ALine == "cac_mode")
{
int q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
q = stoi(ALine);
}
ptr = new cac_mode(q);
}
else if (ALine == "path")
{
int w{}, e{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
```

```
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
e = stoi(ALine);
}
}
ptr = new path(q, w, e);
}
else if (ALine == "max_path")
{
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
```

```
}
ptr = new max_path(q, w);
}
else if (ALine == "min_path")
{
int w{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
w = stoi(ALine);
}
}
ptr = new min_path(q, w);
}
else if (ALine == "op_belongs_to_state")
{
int w{}, e{}, r{};
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
q = ALine;
pos += ALine.length();
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
w = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.find(co, 0));
e = stoi(ALine);
pos += ALine.length();
subl = inputline.substr(++pos, inputline.length() - pos);
}
else stop = 0;
}
if (stop)
{
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
r = stoi(ALine);
}
}
ptr = new op_belongs_to_state(q, w, e, r);
}
else if (ALine == "top_module")
{
string q{};
```

```
subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new top_module(q);
}
else if (ALine == "package_name")
{
string q{};

subl = inputline.substr(++pos, inputline.length() - pos);
if ((subl.find(str, 0)) != 0)
{
params++;
ALine = subl.substr(0, subl.rfind(pacl, 0));
ALine.resize(ALine.size() - 1);
q = ALine;
}
ptr = new package_name(q);
}
else
{
GenfactError* ptr = new GenfactError("error for " + inputline);
cout << "unable to make instance of: " << inputline << endl;
}
}
return factstar(ptr, params);
}

/// @brief compares 2 data
/// @param Treesfact <- loaded data
/// @param obj <- data to compare
/// @return if matched 1 else 0
size_t matchfactsstar(GeneralFact* Treesfact, factstar* obj)
{
bool flag = 1;
```

```
GeneralFact* fact = obj->fact;
size_t p = obj->params;

if (typeid(type_def) == typeid(*fact) && typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
type_def* ptr = dynamic_cast<type_def*>(fact);
type_def* ptr2 = dynamic_cast<type_def*>(Treesfact);
if (p)
{
switch (p)
{
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(op_def) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
op_def* ptr = dynamic_cast<op_def*>(fact);
op_def* ptr2 = dynamic_cast<op_def*>(Treesfact);

if (p)
{
```

```
switch (p)
{
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}

}
}
else if (typeid(hierarchy_part) == typeid(*fact) && typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
hierarchy_part* ptr = dynamic_cast<hierarchy_part*>(fact);
hierarchy_part* ptr2 = dynamic_cast<hierarchy_part*>(Treesfact);
if (p)
{
switch (p)
{
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
```

```
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}

}
}
else if (typeid(data_stmt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
data_stmt* ptr = dynamic_cast<data_stmt*>(fact);
data_stmt* ptr2 = dynamic_cast<data_stmt*>(Treesfact);
if (p)
{
switch (p)
{
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(prog_stmt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
prog_stmt* ptr = dynamic_cast<prog_stmt*>(fact);
prog_stmt* ptr2 = dynamic_cast<prog_stmt*>(Treesfact);
if (p)
{
switch (p)
```

```
{
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(joint_stmt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
joint_stmt* ptr = dynamic_cast<joint_stmt*>(fact);
joint_stmt* ptr2 = dynamic_cast<joint_stmt*>(Treesfact);
if (p)
{
switch (p)
{
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
```

```
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(call_stmt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
call_stmt* ptr = dynamic_cast<call_stmt*>(fact);
call_stmt* ptr2 = dynamic_cast<call_stmt*>(Treesfact);
size_t siz = ptr2->r.size();

if (p)
{
switch (p)
{
case 4: //vector
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(compo_stmt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
compo_stmt* ptr = dynamic_cast<compo_stmt*>(fact);
compo_stmt* ptr2 = dynamic_cast<compo_stmt*>(Treesfact);
size_t siz = ptr2->r.size();

if (p)
{
```

```
switch (p)
{
case 3: //vector
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(rec_stmt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
rec_stmt* ptr = dynamic_cast<rec_stmt*>(fact);
rec_stmt* ptr2 = dynamic_cast<rec_stmt*>(Treesfact);
size_t siz = ptr2->e.size();

if (p)
{
switch (p)
{
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(special_op) == typeid(*fact)&& typeid(*fact)
```

```
== typeid(*Treesfact))
{
flag = 0;
special_op* ptr = dynamic_cast<special_op*>(fact);
special_op* ptr2 = dynamic_cast<special_op*>(Treesfact);
if (p)
{
switch (p)
{
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(special_dt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
special_dt* ptr = dynamic_cast<special_dt*>(fact);
special_dt* ptr2 = dynamic_cast<special_dt*>(Treesfact);
if (p)
{
switch (p)
```

```
{
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(local_object) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
local_object* ptr = dynamic_cast<local_object*>(fact);
local_object* ptr2 = dynamic_cast<local_object*>(Treesfact);
if (p)
{
switch (p)
{
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
```

```
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(subprogram_call) == typeid(*fact) && typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
subprogram_call* ptr = dynamic_cast<subprogram_call*>(fact);
subprogram_call* ptr2 = dynamic_cast<subprogram_call*>(Treesfact);
state_node* ptr1 = dynamic_cast<state_node*>(fact);
state_node* ptr12 = dynamic_cast<state_node*>(Treesfact);
size_t siz = ptr->q.size();

if (p)
{
switch (p)
{
case 5:
if (ptr->w != ptr2->w) return 0;
case 4: //vector
if (ptr2->q.size() != ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->q[i] == ptr2->q[i])) return 0;
case 3:
case 2:
if (ptr1->w != ptr12->w) return 0;
case 1:
if (ptr1->q != ptr12->q) return 0;
}
}
}
else if (typeid(dataflow) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
```

```
flag = 0;
dataflow* ptr = dynamic_cast<dataflow*>(fact);
dataflow* ptr2 = dynamic_cast<dataflow*>(Treesfact);
state_node* ptr1 = dynamic_cast<state_node*>(fact);
state_node* ptr12 = dynamic_cast<state_node*>(Treesfact);
size_t siz = ptr->q.size();

if (p)
{
switch (p)
{
case 5:
if (ptr->w != ptr2->w) return 0;
case 4: //vector
if (ptr2->q.size() != ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->q[i] == ptr2->q[i])) return 0;
case 3:
case 2:
if (ptr1->w != ptr12->w) return 0;
case 1:
if (ptr1->q != ptr12->q) return 0;
}
}
}
else if (typeid(ifthen) == typeid(*fact) && typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
ifthen* ptr = dynamic_cast<ifthen*>(fact);
ifthen* ptr2 = dynamic_cast<ifthen*>(Treesfact);
state_node* ptr1 = dynamic_cast<state_node*>(fact);
state_node* ptr12 = dynamic_cast<state_node*>(Treesfact);
size_t siz = ptr->q.size();
size_t siz2 = ptr->w.size();
size_t siz3 = ptr->e.size();

if (p)
{
```

```
switch (p)
{
case 8:
if (ptr->t != ptr2->t) return 0;
case 7:
if (ptr->r != ptr2->r) return 0;
case 6: //vector3
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz3; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 5: //vector2
if (ptr2->w.size() != ptr->w.size()) return 0;
else if (ptr2->w.empty() != ptr->w.empty()) return 0;
else if (!(ptr2->w.empty() && ptr->w.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->w[i] == ptr2->w[i])) return 0;
case 4: //vector
if (ptr2->q.size() != ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->q[i] == ptr2->q[i])) return 0;
case 3:
case 2:
if (ptr1->w != ptr12->w) return 0;
case 1:
if (ptr1->q != ptr12->q) return 0;
}
}
}
else if (typeid(jump) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
jump* ptr = dynamic_cast<jump*>(fact);
jump* ptr2 = dynamic_cast<jump*>(Treesfact);
state_node* ptr1 = dynamic_cast<state_node*>(fact);
state_node* ptr12 = dynamic_cast<state_node*>(Treesfact);
size_t siz = ptr->q.size();
```

```
if (p)
{
switch (p)
{
case 5:
if (ptr->w != ptr2->w) return 0;
case 4: //vector
if (ptr2->q.size() != ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->q[i] == ptr2->q[i])) return 0;
case 3:
case 2:
if (ptr1->w != ptr12->w) return 0;
case 1:
if (ptr1->q != ptr12->q) return 0;
}
}
}
else if (typeid(return_cos) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
return_cos* ptr = dynamic_cast<return_cos*>(fact);
return_cos* ptr2 = dynamic_cast<return_cos*>(Treesfact);
state_node* ptr1 = dynamic_cast<state_node*>(fact);
state_node* ptr12 = dynamic_cast<state_node*>(Treesfact);

size_t siz = ptr->q.size();

if (p)
{
switch (p)
{
case 4: //vector
if (ptr2->q.size() != ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < siz; i++)
```

```
if (!(ptr->q[i] == ptr2->q[i])) return 0;
case 3:
case 2:
if (ptr1->w != ptr12->w) return 0;
case 1:
if (ptr1->q != ptr12->q) return 0;
}
}
}
else if (typeid(state_node) == typeid(*fact))
{
if (typeid(subprogram_call) == typeid(*Treesfact) || typeid(dataflow)
== typeid(*Treesfact) || typeid(ifthen) == typeid(*Treesfact)
|| typeid(jump) == typeid(*Treesfact) ||
typeid(return_cos) == typeid(*Treesfact))
{
flag = 0;
state_node* ptr = dynamic_cast<state_node*>(fact);
state_node* ptr2 = dynamic_cast<state_node*>(Treesfact);

if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
}
else if (typeid(change_op_number) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
change_op_number* ptr = dynamic_cast<change_op_number*>(fact);
change_op_number* ptr2 = dynamic_cast<change_op_number*>(Treesfact);
if (p)
{
switch (p)
```

```
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_change_op_number) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_change_op_number* ptr =
dynamic_cast<last_change_op_number*>(fact);
last_change_op_number* ptr2 =
dynamic_cast<last_change_op_number*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(op_guards) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
op_guards* ptr = dynamic_cast<op_guards*>(fact);
op_guards* ptr2 = dynamic_cast<op_guards*>(Treesfact);
size_t siz = ptr->e.size();
size_t siz2 = ptr->r.size();

if (p)
```

```
{
switch (p)
{
case 4: //vector2
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(var_guards) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
var_guards* ptr = dynamic_cast<var_guards*>(fact);
var_guards* ptr2 = dynamic_cast<var_guards*>(Treesfact);
size_t siz = ptr->e.size();
size_t siz2 = ptr->r.size();

if (p)
{
switch (p)
{
case 4: //vector2
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
```

```
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(guard_pair) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
guard_pair* ptr = dynamic_cast<guard_pair*>(fact);
guard_pair* ptr2 = dynamic_cast<guard_pair*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(guard_cond) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
guard_cond* ptr = dynamic_cast<guard_cond*>(fact);
guard_cond* ptr2 = dynamic_cast<guard_cond*>(Treesfact);
if (p)
```

```
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(predecessors) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
predecessors* ptr = dynamic_cast<predecessors*>(fact);
predecessors* ptr2 = dynamic_cast<predecessors*>(Treesfact);
size_t siz = ptr->e.size();

if (p)
{
switch (p)
{
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(cessor) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
```

```
{
flag = 0;
cessor* ptr = dynamic_cast<cessor*>(fact);
cessor* ptr2 = dynamic_cast<cessor*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(cessor_kind) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
cessor_kind* ptr = dynamic_cast<cessor_kind*>(fact);
cessor_kind* ptr2 = dynamic_cast<cessor_kind*>(Treesfact);
size_t siz = ptr->u.size();

if (p)
{
switch (p)
{
case 7:
if (ptr2->u.size() != ptr->u.size()) return 0;
for (int i = 0; i < siz; i++)
if (!(ptr->u[i] == ptr2->u[i])) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
```

```
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(old_schedule) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
old_schedule* ptr = dynamic_cast<old_schedule*>(fact);
old_schedule* ptr2 = dynamic_cast<old_schedule*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(new_schedule) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
new_schedule* ptr = dynamic_cast<new_schedule*>(fact);
new_schedule* ptr2 = dynamic_cast<new_schedule*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(local_ifthen_chain_end_operations_were_written)
== typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
local_ifthen_chain_end_operations_were_written* ptr =
dynamic_cast<local_ifthen_chain_end_operations_were_written*>(fact);
local_ifthen_chain_end_operations_were_written* ptr2 =
dynamic_cast<local_ifthen_chain_end_operations_were_written*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
```

```
}
else if (typeid(calls_list) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
calls_list* ptr = dynamic_cast<calls_list*>(fact);
calls_list* ptr2 = dynamic_cast<calls_list*>(Treesfact);
size_t siz = ptr->r.size();

if (p)
{
switch (p)
{
case 4: //vector
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(composites_list) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
composites_list* ptr = dynamic_cast<composites_list*>(fact);
composites_list* ptr2 = dynamic_cast<composites_list*>(Treesfact);
size_t siz = ptr->e.size();

if (p)
{
switch (p)
{
case 3: //vector
```

```
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(record_aggregates_list) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
record_aggregates_list* ptr =
dynamic_cast<record_aggregates_list*>(fact);
record_aggregates_list* ptr2 =
dynamic_cast<record_aggregates_list*>(Treesfact);
size_t siz = ptr->e.size();

if (p)
{
switch (p)
{
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(mem_port) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
```

```
{
flag = 0;
mem_port* ptr = dynamic_cast<mem_port*>(fact);
mem_port* ptr2 = dynamic_cast<mem_port*>(Treesfact);
if (p)
{
switch (p)
{
case 13:
if (ptr->d != ptr2->d) return 0;
case 12:
if (ptr->s != ptr2->s) return 0;
case 11:
if (ptr->a != ptr2->a) return 0;
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(global_declarations) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
```

```
global_declarations* ptr =
dynamic_cast<global_declarations*>(fact);
global_declarations* ptr2 =
dynamic_cast<global_declarations*>(Treesfact);

if (p)
{
if (ptr->w != NULL)
{
if (ptr->w != ptr2->w) return 0;
--p;
}
//vector
if (ptr2->q.size() < ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < p; i++)
{
if (ptr->q[i].q != ptr2->q[i].q) return 0;
if (ptr->q[i].w != ptr2->q[i].w) return 0;
if (ptr->q[i].e != ptr2->q[i].e) return 0;
if (ptr->q[i].r != ptr2->q[i].r) return 0;
if (ptr->q[i].t != ptr2->q[i].t) return 0;
if (ptr->q[i].y != ptr2->q[i].y) return 0;
if (ptr->q[i].u != ptr2->q[i].u) return 0;
if (ptr->q[i].i != ptr2->q[i].i) return 0;
if (ptr->q[i].o != ptr2->q[i].o) return 0;
}
}
}
else if (typeid(source_is_normal_dt) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
source_is_normal_dt* ptr = dynamic_cast<source_is_normal_dt*>(fact);
source_is_normal_dt* ptr2 = dynamic_cast<source_is_normal_dt*>(Treesfact);

if (p)
{
switch (p)
{
```

```
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(combo) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
combo* ptr = dynamic_cast<combo*>(fact);
combo* ptr2 = dynamic_cast<combo*>(Treesfact);

if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(sequence) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
sequence* ptr = dynamic_cast<sequence*>(fact);
sequence* ptr2 = dynamic_cast<sequence*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
```

```
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(for_loop) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
for_loop* ptr = dynamic_cast<for_loop*>(fact);
for_loop* ptr2 = dynamic_cast<for_loop*>(Treesfact);

if (p)
{
switch (p)
{
case 13:
if (ptr->d != ptr2->d) return 0;
case 12:
if (ptr->s != ptr2->s) return 0;
case 11:
if (ptr->a != ptr2->a) return 0;
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
```

```
}
}
}
else if (typeid(last_for_loop_entry) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_for_loop_entry* ptr = dynamic_cast<last_for_loop_entry*>(fact);
last_for_loop_entry* ptr2 = dynamic_cast<last_for_loop_entry*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(while_loop) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
while_loop* ptr = dynamic_cast<while_loop*>(fact);
while_loop* ptr2 = dynamic_cast<while_loop*>(Treesfact);

if (p)
{
switch (p)
{
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
```

```
}
else if (typeid(last_while_loop_entry) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_while_loop_entry* ptr = dynamic_cast<last_while_loop_entry*>(fact);
last_while_loop_entry* ptr2 = dynamic_cast<last_while_loop_entry*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(possible_end_if) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
possible_end_if* ptr = dynamic_cast<possible_end_if*>(fact);
possible_end_if* ptr2 = dynamic_cast<possible_end_if*>(Treesfact);
size_t siz = ptr2->w.size();

if (p)
{
switch (p)
{
case 2: //vector
if (ptr2->w.size() != ptr->w.size()) return 0;
else if (ptr2->w.empty() != ptr->w.empty()) return 0;
else if (!(ptr2->w.empty() && ptr->w.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->w[i] == ptr2->w[i])) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(end_if) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
end_if* ptr = dynamic_cast<end_if*>(fact);
end_if* ptr2 = dynamic_cast<end_if*>(Treesfact);
size_t siz = ptr2->w.size();
```

```
if (p)
{
switch (p)
{
case 2: //vector
if (ptr2->w.size() != ptr->w.size()) return 0;
else if (ptr2->w.empty() != ptr->w.empty()) return 0;
else if (!(ptr2->w.empty() && ptr->w.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->w[i] == ptr2->w[i])) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(nested_cond_fact) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
nested_cond_fact* ptr = dynamic_cast<nested_cond_fact*>(fact);
nested_cond_fact* ptr2 = dynamic_cast<nested_cond_fact*>(Treesfact);
size_t siz = ptr2->w.size();

if (p)
{
switch (p)
{
case 2: //vector
if (ptr2->w.size() != ptr->w.size()) return 0;
else if (ptr2->w.empty() != ptr->w.empty()) return 0;
else if (!(ptr2->w.empty() && ptr->w.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->w[i] == ptr2->w[i])) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(top_level_call) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
```

```
{
flag = 0;
top_level_call* ptr = dynamic_cast<top_level_call*>(fact);
top_level_call* ptr2 = dynamic_cast<top_level_call*>(Treesfact);
if (p)
{
switch (p)
{
case 11:
if (ptr->a != ptr2->a) return 0;
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(top_level_call_parcs) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
top_level_call_parcs* ptr = dynamic_cast<top_level_call_parcs*>(fact);
top_level_call_parcs* ptr2 = dynamic_cast<top_level_call_parcs*>(Treesfact);
if (p)
{
```

```
switch (p)
{
case 11:
if (ptr->a != ptr2->a) return 0;
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(added_aux_call_ios) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
added_aux_call_ios* ptr = dynamic_cast<added_aux_call_ios*>(fact);
added_aux_call_ios* ptr2 = dynamic_cast<added_aux_call_ios*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
```

```
}
}
}
else if (typeid(added_aux_call_ios1) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
added_aux_call_ios1* ptr = dynamic_cast<added_aux_call_ios1*>(fact);
added_aux_call_ios1* ptr2 = dynamic_cast<added_aux_call_ios1*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(added_aux_call_signals) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
added_aux_call_signals* ptr = dynamic_cast<added_aux_call_signals*>(fact);
added_aux_call_signals* ptr2 = dynamic_cast<added_aux_call_signals*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(found_call_operator) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
```

```
{
flag = 0;
found_call_operator* ptr = dynamic_cast<found_call_operator*>(fact);
found_call_operator* ptr2 = dynamic_cast<found_call_operator*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(added_verilog_aux_call_outputs) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
added_verilog_aux_call_outputs* ptr =
dynamic_cast<added_verilog_aux_call_outputs*>(fact);
added_verilog_aux_call_outputs* ptr2 =
dynamic_cast<added_verilog_aux_call_outputs*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(raw_dependencies) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
raw_dependencies* ptr = dynamic_cast<raw_dependencies*>(fact);
```

```
raw_dependencies* ptr2 = dynamic_cast<raw_dependencies*>(Treesfact);
size_t siz = ptr->e.size();
size_t siz2 = ptr->r.size();

if (p)
{
switch (p)
{
case 4: //vector2
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(war_dependencies) == typeid(*fact) && typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
war_dependencies* ptr = dynamic_cast<war_dependencies*>(fact);
war_dependencies* ptr2 = dynamic_cast<war_dependencies*>(Treesfact);
size_t siz = ptr->e.size();
size_t siz2 = ptr->r.size();

if (p)
{
switch (p)
{
case 4: //vector2
```

```
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(waw_dependencies) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
waw_dependencies* ptr = dynamic_cast<waw_dependencies*>(fact);
waw_dependencies* ptr2 = dynamic_cast<waw_dependencies*>(Treesfact);
size_t siz = ptr->e.size();
size_t siz2 = ptr->r.size();

if (p)
{
switch (p)
{
case 4: //vector2
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3: //vector
if (ptr2->e.size() != ptr->e.size()) return 0;
else if (ptr2->e.empty() != ptr->e.empty()) return 0;
else if (!(ptr2->e.empty() && ptr->e.empty()))
```

```
for (int i = 0; i < siz; i++)
if (!(ptr->e[i] == ptr2->e[i])) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(schedule) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
schedule* ptr = dynamic_cast<schedule*>(fact);
schedule* ptr2 = dynamic_cast<schedule*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_conditional_execution) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_conditional_execution* ptr =
dynamic_cast<last_conditional_execution*>(fact);
last_conditional_execution* ptr2 =
dynamic_cast<last_conditional_execution*>(Treesfact);
if (p)
{
switch (p)
```

```
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(conditional_operations) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
conditional_operations* ptr =
dynamic_cast<conditional_operations*>(fact);
conditional_operations* ptr2 =
dynamic_cast<conditional_operations*>(Treesfact);
size_t siz = ptr->t.size();
size_t siz2 = ptr->y.size();
size_t siz3 = ptr->u.size();
size_t siz4 = ptr->i.size();

if (p)
{
switch (p)
{
case 8: //vector4
if (ptr2->i.size() != ptr->i.size()) return 0;
else if (ptr2->i.empty() != ptr->i.empty()) return 0;
else if (!(ptr2->i.empty() && ptr->i.empty()))
for (int i = 0; i < siz4; i++)
if (!(ptr->i[i] == ptr2->i[i])) return 0;
case 7: //vector3
if (ptr2->u.size() != ptr->u.size()) return 0;
else if (ptr2->u.empty() != ptr->u.empty()) return 0;
else if (!(ptr2->u.empty() && ptr->u.empty()))
for (int i = 0; i < siz3; i++)
if (!(ptr->u[i] == ptr2->u[i])) return 0;
case 6: //vector2
if (ptr2->y.size() != ptr->y.size()) return 0;
else if (ptr2->y.empty() != ptr->y.empty()) return 0;
else if (!(ptr2->y.empty() && ptr->y.empty()))
```

```
for (int i = 0; i < siz2; i++)
if (!(ptr->y[i] == ptr2->y[i])) return 0;
case 5: //vector
if (ptr2->t.size() != ptr->t.size()) return 0;
else if (ptr2->t.empty() != ptr->t.empty()) return 0;
else if (!(ptr2->t.empty() && ptr->t.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->t[i] == ptr2->t[i])) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_conditional_transition_of_schedule)
== typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
last_conditional_transition_of_schedule* ptr =
dynamic_cast<last_conditional_transition_of_schedule*>(fact);
last_conditional_transition_of_schedule* ptr2 =
dynamic_cast<last_conditional_transition_of_schedule*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(transition_to_be_rescheduled) == typeid(*fact)&& typeid(*fact)
```

```
== typeid(*Treesfact))
{
flag = 0;
transition_to_be_rescheduled* ptr =
dynamic_cast<transition_to_be_rescheduled*>(fact);
transition_to_be_rescheduled* ptr2 =
dynamic_cast<transition_to_be_rescheduled*>(Treesfact);
if (p)
{
switch (p)
{
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_conditional_transition) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_conditional_transition* ptr =
dynamic_cast<last_conditional_transition*>(fact);
last_conditional_transition* ptr2 =
dynamic_cast<last_conditional_transition*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
```

```
}
}
else if (typeid(conditional_transitions) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
conditional_transitions* ptr =
dynamic_cast<conditional_transitions*>(fact);
conditional_transitions* ptr2 =
dynamic_cast<conditional_transitions*>(Treesfact);
if (p)
{
switch (p)
{
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(state) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
state* ptr = dynamic_cast<state*>(fact);
state* ptr2 = dynamic_cast<state*>(Treesfact);
size_t siz = ptr->u.size();
size_t siz2 = ptr->i.size();
```

```
if (p)
{
switch (p)
{
case 8: //vector2
if (ptr2->i.size() != ptr->i.size()) return 0;
else if (ptr2->i.empty() != ptr->i.empty()) return 0;
else if (!(ptr2->i.empty() && ptr->i.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->i[i] == ptr2->i[i])) return 0;
case 7: //vector
if (ptr2->u.size() != ptr->u.size()) return 0;
else if (ptr2->u.empty() != ptr->u.empty()) return 0;
else if (!(ptr2->u.empty() && ptr->u.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->u[i] == ptr2->u[i])) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(rescheduled) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
rescheduled* ptr = dynamic_cast<rescheduled*>(fact);
rescheduled* ptr2 = dynamic_cast<rescheduled*>(Treesfact);
if (p)
{
switch (p)
{
```

```
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_rescheduled) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_rescheduled* ptr = dynamic_cast<last_rescheduled*>(fact);
last_rescheduled* ptr2 = dynamic_cast<last_rescheduled*>(Treesfact);
if (p)
{
switch (p)
{
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(raw_cessor) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
raw_cessor* ptr = dynamic_cast<raw_cessor*>(fact);
```

```
raw_cessor* ptr2 = dynamic_cast<raw_cessor*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(war_cessor) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
war_cessor* ptr = dynamic_cast<war_cessor*>(fact);
war_cessor* ptr2 = dynamic_cast<war_cessor*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(waw_cessor) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
waw_cessor* ptr = dynamic_cast<waw_cessor*>(fact);
waw_cessor* ptr2 = dynamic_cast<waw_cessor*>(Treesfact);
if (p)
{
```

```
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(op_resource) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
op_resource* ptr = dynamic_cast<op_resource*>(fact);
op_resource* ptr2 = dynamic_cast<op_resource*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(global_resource) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
global_resource* ptr = dynamic_cast<global_resource*>(fact);
global_resource* ptr2 = dynamic_cast<global_resource*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
```

```
else if (typeid(module_g_resource) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
module_g_resource* ptr = dynamic_cast<module_g_resource*>(fact);
module_g_resource* ptr2 = dynamic_cast<module_g_resource*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(cf_previous_op) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
cf_previous_op* ptr = dynamic_cast<cf_previous_op*>(fact);
cf_previous_op* ptr2 = dynamic_cast<cf_previous_op*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(cf_previous_state) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
```

```
flag = 0;
cf_previous_state* ptr = dynamic_cast<cf_previous_state*>(fact);
cf_previous_state* ptr2 = dynamic_cast<cf_previous_state*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(pred_candidate_examined) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
pred_candidate_examined* ptr =
dynamic_cast<pred_candidate_examined*>(fact);
pred_candidate_examined* ptr2 =
dynamic_cast<pred_candidate_examined*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(reentrant_triangle) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
```

```
{
flag = 0;
reentrant_triangle* ptr = dynamic_cast<reentrant_triangle*>(fact);
reentrant_triangle* ptr2 = dynamic_cast<reentrant_triangle*>(Treesfact);

size_t siz = ptr->t.size();
size_t siz2 = ptr->y.size();
size_t siz3 = ptr->u.size();
size_t siz4 = ptr->i.size();
size_t siz5 = ptr->o.size();
size_t siz6 = ptr->p.size();

if (p)
{
switch (p)
{
case 14:
if (ptr->f != ptr2->f) return 0;
case 13:
if (ptr->d != ptr2->d) return 0;
case 12:
if (ptr->s != ptr2->s) return 0;
case 11:
if (ptr->a != ptr2->a) return 0;
case 10: //vector6
if (ptr2->p.size() != ptr->p.size()) return 0;
else if (ptr2->p.empty() != ptr->p.empty()) return 0;
else if (!(ptr2->p.empty() && ptr->p.empty()))
for (int i = 0; i < siz6; i++)
if (!(ptr->p[i] == ptr2->p[i])) return 0;
case 9: //vector5
if (ptr2->o.size() != ptr->o.size()) return 0;
else if (ptr2->o.empty() != ptr->o.empty()) return 0;
else if (!(ptr2->o.empty() && ptr->o.empty()))
for (int i = 0; i < siz5; i++)
if (!(ptr->o[i] == ptr2->o[i])) return 0;
case 8: //vector4
if (ptr2->i.size() != ptr->i.size()) return 0;
else if (ptr2->i.empty() != ptr->i.empty()) return 0;
else if (!(ptr2->i.empty() && ptr->i.empty()))
for (int i = 0; i < siz4; i++)
```

```
if (!(ptr->i[i] == ptr2->i[i])) return 0;
case 7: //vector3
if (ptr2->u.size() != ptr->u.size()) return 0;
else if (ptr2->u.empty() != ptr->u.empty()) return 0;
else if (!(ptr2->u.empty() && ptr->u.empty()))
for (int i = 0; i < siz3; i++)
if (!(ptr->u[i] == ptr2->u[i])) return 0;
case 6: //vector2
if (ptr2->y.size() != ptr->y.size()) return 0;
else if (ptr2->y.empty() != ptr->y.empty()) return 0;
else if (!(ptr2->y.empty() && ptr->y.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->y[i] == ptr2->y[i])) return 0;
case 5: //vector
if (ptr2->t.size() != ptr->t.size()) return 0;
else if (ptr2->t.empty() != ptr->t.empty()) return 0;
else if (!(ptr2->t.empty() && ptr->t.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->t[i] == ptr2->t[i])) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_reentrant_triangle) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_reentrant_triangle* ptr = dynamic_cast<last_reentrant_triangle*>(fact);
last_reentrant_triangle* ptr2 = dynamic_cast<last_reentrant_triangle*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
```

```
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_schedule_state) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_schedule_state* ptr = dynamic_cast<last_schedule_state*>(fact);
last_schedule_state* ptr2 = dynamic_cast<last_schedule_state*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(conditional_incomplete) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
conditional_incomplete* ptr =
dynamic_cast<conditional_incomplete*>(fact);
conditional_incomplete* ptr2 =
dynamic_cast<conditional_incomplete*>(Treesfact);
size_t siz = ptr->t.size();
size_t siz2 = ptr->y.size();
size_t siz3 = ptr->u.size();
size_t siz4 = ptr->i.size();
size_t siz5 = ptr->o.size();
size_t siz6 = ptr->p.size();

if (p)
```

```
{
switch (p)
{
case 21:
if (ptr->x != ptr2->x) return 0;
case 20:
if (ptr->z != ptr2->z) return 0;
case 19:
if (ptr->l != ptr2->l) return 0;
case 18:
if (ptr->k != ptr2->k) return 0;
case 17:
if (ptr->j != ptr2->j) return 0;
case 16:
if (ptr->h != ptr2->h) return 0;
case 15:
if (ptr->g != ptr2->g) return 0;
case 14:
if (ptr->f != ptr2->f) return 0;
case 13:
if (ptr->d != ptr2->d) return 0;
case 12:
if (ptr->s != ptr2->s) return 0;
case 11:
if (ptr->a != ptr2->a) return 0;
case 10: //vector6
if (ptr2->p.size() != ptr->p.size()) return 0;
else if (ptr2->p.empty() != ptr->p.empty()) return 0;
else if (!(ptr2->p.empty() && ptr->p.empty()))
for (int i = 0; i < siz6; i++)
if (!(ptr->p[i] == ptr2->p[i])) return 0;
case 9: //vector5
if (ptr2->o.size() != ptr->o.size()) return 0;
else if (ptr2->o.empty() != ptr->o.empty()) return 0;
else if (!(ptr2->o.empty() && ptr->o.empty()))
for (int i = 0; i < siz5; i++)
if (!(ptr->o[i] == ptr2->o[i])) return 0;
case 8: //vector4
if (ptr2->i.size() != ptr->i.size()) return 0;
else if (ptr2->i.empty() != ptr->i.empty()) return 0;
else if (!(ptr2->i.empty() && ptr->i.empty()))
```

```
for (int i = 0; i < siz4; i++)
if (!(ptr->i[i] == ptr2->i[i])) return 0;
case 7: //vector3
if (ptr2->u.size() != ptr->u.size()) return 0;
else if (ptr2->u.empty() != ptr->u.empty()) return 0;
else if (!(ptr2->u.empty() && ptr->u.empty()))
for (int i = 0; i < siz3; i++)
if (!(ptr->u[i] == ptr2->u[i])) return 0;
case 6: //vector2
if (ptr2->y.size() != ptr->y.size()) return 0;
else if (ptr2->y.empty() != ptr->y.empty()) return 0;
else if (!(ptr2->y.empty() && ptr->y.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->y[i] == ptr2->y[i])) return 0;
case 5: //vector
if (ptr2->t.size() != ptr->t.size()) return 0;
else if (ptr2->t.empty() != ptr->t.empty()) return 0;
else if (!(ptr2->t.empty() && ptr->t.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->t[i] == ptr2->t[i])) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(mixed_incomplete_state_lists) ==
typeid(*fact) && typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
mixed_incomplete_state_lists* ptr =
dynamic_cast<mixed_incomplete_state_lists*>(fact);
mixed_incomplete_state_lists* ptr2 =
dynamic_cast<mixed_incomplete_state_lists*>(Treesfact);
size_t siz = ptr->r.size();
size_t siz2 = ptr->t.size();
```

```
if (p)
{
switch (p)
{
case 5: //vector2
if (ptr2->t.size() != ptr->t.size()) return 0;
else if (ptr2->t.empty() != ptr->t.empty()) return 0;
else if (!(ptr2->t.empty() && ptr->t.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->t[i] == ptr2->t[i])) return 0;
case 4: //vector
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(linear_incomplete_node) ==
typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
flag = 0;
linear_incomplete_node* ptr =
dynamic_cast<linear_incomplete_node*>(fact);
linear_incomplete_node* ptr2 =
dynamic_cast<linear_incomplete_node*>(Treesfact);
size_t siz = ptr->r.size();
size_t siz2 = ptr->t.size();

if (p)
{
switch (p)
{
```

```
case 6:
if (ptr->y != ptr2->y) return 0;
case 5: //vector2
if (ptr2->t.size() != ptr->t.size()) return 0;
else if (ptr2->t.empty() != ptr->t.empty()) return 0;
else if (!(ptr2->t.empty() && ptr->t.empty()))
for (int i = 0; i < siz2; i++)
if (!(ptr->t[i] == ptr2->t[i])) return 0;
case 4: //vector
if (ptr2->r.size() != ptr->r.size()) return 0;
else if (ptr2->r.empty() != ptr->r.empty()) return 0;
else if (!(ptr2->r.empty() && ptr->r.empty()))
for (int i = 0; i < siz; i++)
if (!(ptr->r[i] == ptr2->r[i])) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(incomplete_links) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
incomplete_links* ptr = dynamic_cast<incomplete_links*>(fact);
incomplete_links* ptr2 = dynamic_cast<incomplete_links*>(Treesfact);
if (p)
{
switch (p)
{
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
```

```
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_incomplete) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_incomplete* ptr = dynamic_cast<last_incomplete*>(fact);
last_incomplete* ptr2 = dynamic_cast<last_incomplete*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(global_nils) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
global_nils* ptr = dynamic_cast<global_nils*>(fact);
```

```
global_nils* ptr2 = dynamic_cast<global_nils*>(Treesfact);
if (p)
{
//vector
if (ptr2->q.size() < ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < p; i++)
{
if (ptr->q[i].q != ptr2->q[i].q) return 0;
if (ptr->q[i].w != ptr2->q[i].w) return 0;
}
}
}
else if (typeid(current_module) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
current_module* ptr = dynamic_cast<current_module*>(fact);
current_module* ptr2 = dynamic_cast<current_module*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(last_linear_incomplete_node) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_linear_incomplete_node* ptr =
dynamic_cast<last_linear_incomplete_node*>(fact);
last_linear_incomplete_node* ptr2 =
dynamic_cast<last_linear_incomplete_node*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(operator_instances) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
```

```
flag = 0;
operator_instances* ptr = dynamic_cast<operator_instances*>(fact);
operator_instances* ptr2 = dynamic_cast<operator_instances*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(massively_parallel_style) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
massively_parallel_style* ptr =
dynamic_cast<massively_parallel_style*>(fact);
massively_parallel_style* ptr2 =
dynamic_cast<massively_parallel_style*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(hdl_style) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
hdl_style* ptr = dynamic_cast<hdl_style*>(fact);
hdl_style* ptr2 = dynamic_cast<hdl_style*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(op_instance) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
```

```
flag = 0;
op_instance* ptr = dynamic_cast<op_instance*>(fact);
op_instance* ptr2 = dynamic_cast<op_instance*>(Treesfact);

if (p)
{
switch (p)
{
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_op_instance) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_op_instance* ptr = dynamic_cast<last_op_instance*>(fact);
last_op_instance* ptr2 = dynamic_cast<last_op_instance*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(op_in_a_state) == typeid(*fact)&& typeid(*fact)
```

```
== typeid(*Treesfact))
{
flag = 0;
op_in_a_state* ptr = dynamic_cast<op_in_a_state*>(fact);
op_in_a_state* ptr2 = dynamic_cast<op_in_a_state*>(Treesfact);
if (p)
{
switch (p)
{
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_op_in_a_state) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_op_in_a_state* ptr = dynamic_cast<last_op_in_a_state*>(fact);
last_op_in_a_state* ptr2 = dynamic_cast<last_op_in_a_state*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
```

```
}
}
else if (typeid(signal_instance) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
signal_instance* ptr = dynamic_cast<signal_instance*>(fact);
signal_instance* ptr2 = dynamic_cast<signal_instance*>(Treesfact);
if (p)
{
switch (p)
{
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_signal_instance) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_signal_instance* ptr = dynamic_cast<last_signal_instance*>(fact);
last_signal_instance* ptr2 = dynamic_cast<last_signal_instance*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
```

```
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(output_instance) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
output_instance* ptr = dynamic_cast<output_instance*>(fact);
output_instance* ptr2 = dynamic_cast<output_instance*>(Treesfact);
if (p)
{
switch (p)
{
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(last_output_instance) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
last_output_instance* ptr = dynamic_cast<last_output_instance*>(fact);
last_output_instance* ptr2 = dynamic_cast<last_output_instance*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
```

```
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(operator_instance_stats) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
operator_instance_stats* ptr =
dynamic_cast<operator_instance_stats*>(fact);
operator_instance_stats* ptr2 =
dynamic_cast<operator_instance_stats*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(consecutive_106) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
consecutive_106* ptr = dynamic_cast<consecutive_106*>(fact);
consecutive_106* ptr2 = dynamic_cast<consecutive_106*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(operation_order) == typeid(*fact)&& typeid(*fact)
```

```
== typeid(*Treesfact))
{
flag = 0;
operation_order* ptr = dynamic_cast<operation_order*>(fact);
operation_order* ptr2 = dynamic_cast<operation_order*>(Treesfact);
if (p)
{
switch (p)
{
case 10:
if (ptr->p != ptr2->p) return 0;
case 9:
if (ptr->o != ptr2->o) return 0;
case 8:
if (ptr->i != ptr2->i) return 0;
case 7:
if (ptr->u != ptr2->u) return 0;
case 6:
if (ptr->y != ptr2->y) return 0;
case 5:
if (ptr->t != ptr2->t) return 0;
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(max_parallel_call_order) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
max_parallel_call_order* ptr =
dynamic_cast<max_parallel_call_order*>(fact);
max_parallel_call_order* ptr2 =
dynamic_cast<max_parallel_call_order*>(Treesfact);
if (p)
```

```
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(max_op_order) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
max_op_order* ptr = dynamic_cast<max_op_order*>(fact);
max_op_order* ptr2 = dynamic_cast<max_op_order*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(totalmax_call_order) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
totalmax_call_order* ptr = dynamic_cast<totalmax_call_order*>(fact);
totalmax_call_order* ptr2 = dynamic_cast<totalmax_call_order*>(Treesfact);
if (p)
{
```

```
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(totalmax_gross_depth) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
totalmax_gross_depth* ptr = dynamic_cast<totalmax_gross_depth*>(fact);
totalmax_gross_depth* ptr2 = dynamic_cast<totalmax_gross_depth*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(current_total_max_order_entry) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
current_total_max_order_entry* ptr =
dynamic_cast<current_total_max_order_entry*>(fact);
current_total_max_order_entry* ptr2 =
dynamic_cast<current_total_max_order_entry*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
```

```
}
}
else if (typeid(module_last_state) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
module_last_state* ptr = dynamic_cast<module_last_state*>(fact);
module_last_state* ptr2 = dynamic_cast<module_last_state*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(module_local_list) == typeid(*fact) && typeid(*fact)
== typeid(*Treesfact))
{
flag = 0;
module_local_list* ptr = dynamic_cast<module_local_list*>(fact);
module_local_list* ptr2 = dynamic_cast<module_local_list*>(Treesfact);

if (p)
{
//vector
if (ptr2->q.size() > ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < p; i++)
{
if (ptr->q[i].q != ptr2->q[i].q) return 0;
if (ptr->q[i].w != ptr2->q[i].w) return 0;
if (ptr->q[i].e != ptr2->q[i].e) return 0;
if (ptr->q[i].r != ptr2->q[i].r) return 0;
if (ptr->q[i].t != ptr2->q[i].t) return 0;
if (ptr->q[i].y != ptr2->q[i].y) return 0;
if (ptr->q[i].u != ptr2->q[i].u) return 0;
if (ptr->q[i].i != ptr2->q[i].i) return 0;
if (ptr->q[i].o != ptr2->q[i].o) return 0;
}
}
}
if (flag)
```

```
{
if (typeid(module_local_list_parcs) == typeid(*fact) && typeid(*fact)
== typeid(*Treesfact))
{
module_local_list_parcs* ptr =
dynamic_cast<module_local_list_parcs*>(fact);
module_local_list_parcs* ptr2 =
dynamic_cast<module_local_list_parcs*>(Treesfact);

if (p)
{
//vector
if (ptr2->q.size() > ptr->q.size()) return 0;
else if (ptr2->q.empty() != ptr->q.empty()) return 0;
else if (!(ptr2->q.empty() && ptr->q.empty()))
for (int i = 0; i < p; i++)
{
if (ptr->q[i].q != ptr2->q[i].q) return 0;
if (ptr->q[i].w != ptr2->q[i].w) return 0;
if (ptr->q[i].e != ptr2->q[i].e) return 0;
if (ptr->q[i].r != ptr2->q[i].r) return 0;
if (ptr->q[i].t != ptr2->q[i].t) return 0;
if (ptr->q[i].y != ptr2->q[i].y) return 0;
if (ptr->q[i].u != ptr2->q[i].u) return 0;
if (ptr->q[i].i != ptr2->q[i].i) return 0;
if (ptr->q[i].o != ptr2->q[i].o) return 0;
}
}
}
else if (typeid(last_non_io_found) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
last_non_io_found* ptr = dynamic_cast<last_non_io_found*>(fact);
last_non_io_found* ptr2 = dynamic_cast<last_non_io_found*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(last_local_number) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
```

```
{
last_local_number* ptr = dynamic_cast<last_local_number*>(fact);
last_local_number* ptr2 = dynamic_cast<last_local_number*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(printed_formal_ios_of_called_module) ==
typeid(*fact) && typeid(*fact) == typeid(*Treesfact))
{
printed_formal_ios_of_called_module* ptr =
dynamic_cast<printed_formal_ios_of_called_module*>(fact);
printed_formal_ios_of_called_module* ptr2 =
dynamic_cast<printed_formal_ios_of_called_module*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(it_includes_ifthen) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
it_includes_ifthen* ptr = dynamic_cast<it_includes_ifthen*>(fact);
it_includes_ifthen* ptr2 = dynamic_cast<it_includes_ifthen*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
```

```
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(it_includes_conditional_targeting) ==
typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
it_includes_conditional_targeting* ptr
= dynamic_cast<it_includes_conditional_targeting*>(fact);
it_includes_conditional_targeting* ptr2
= dynamic_cast<it_includes_conditional_targeting*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(targets_conditional_variable) ==
typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
targets_conditional_variable* ptr =
dynamic_cast<targets_conditional_variable*>(fact);
targets_conditional_variable* ptr2 =
dynamic_cast<targets_conditional_variable*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
```

```
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(variable_has_been_listed) ==
typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
variable_has_been_listed* ptr =
dynamic_cast<variable_has_been_listed*>(fact);
variable_has_been_listed* ptr2 =
dynamic_cast<variable_has_been_listed*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(resetstyle) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
resetstyle* ptr = dynamic_cast<resetstyle*>(fact);
resetstyle* ptr2 = dynamic_cast<resetstyle*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(checkstyle) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
checkstyle* ptr = dynamic_cast<checkstyle*>(fact);
checkstyle* ptr2 = dynamic_cast<checkstyle*>(Treesfact);
```

```
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(total_local_entry) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
total_local_entry* ptr = dynamic_cast<total_local_entry*>(fact);
total_local_entry* ptr2 = dynamic_cast<total_local_entry*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(complex_next_state_operation_depth) ==
typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
complex_next_state_operation_depth* ptr =
dynamic_cast<complex_next_state_operation_depth*>(fact);
complex_next_state_operation_depth* ptr2 =
dynamic_cast<complex_next_state_operation_depth*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(output_filename) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
output_filename* ptr = dynamic_cast<output_filename*>(fact);
output_filename* ptr2 = dynamic_cast<output_filename*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(hdl_io_pass) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
hdl_io_pass* ptr = dynamic_cast<hdl_io_pass*>(fact);
```

```
hdl_io_pass* ptr2 = dynamic_cast<hdl_io_pass*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(current_hdl_style) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
current_hdl_style* ptr = dynamic_cast<current_hdl_style*>(fact);
current_hdl_style* ptr2 = dynamic_cast<current_hdl_style*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(call_ios_have_been_reset) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
call_ios_have_been_reset* ptr =
dynamic_cast<call_ios_have_been_reset*>(fact);
call_ios_have_been_reset* ptr2 =
dynamic_cast<call_ios_have_been_reset*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(debug_mode) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
debug_mode* ptr = dynamic_cast<debug_mode*>(fact);
debug_mode* ptr2 = dynamic_cast<debug_mode*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(print_C_main_body) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
```

```
print_C_main_body* ptr = dynamic_cast<print_C_main_body*>(fact);
print_C_main_body* ptr2 = dynamic_cast<print_C_main_body*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(cac_mode) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
cac_mode* ptr = dynamic_cast<cac_mode*>(fact);
cac_mode* ptr2 = dynamic_cast<cac_mode*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(path) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
path* ptr = dynamic_cast<path*>(fact);
path* ptr2 = dynamic_cast<path*>(Treesfact);
if (p)
{
switch (p)
{
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(max_path) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
max_path* ptr = dynamic_cast<max_path*>(fact);
max_path* ptr2 = dynamic_cast<max_path*>(Treesfact);
if (p)
{
```

```
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(min_path) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
min_path* ptr = dynamic_cast<min_path*>(fact);
min_path* ptr2 = dynamic_cast<min_path*>(Treesfact);
if (p)
{
switch (p)
{
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(op_belongs_to_state) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
op_belongs_to_state* ptr = dynamic_cast<op_belongs_to_state*>(fact);
op_belongs_to_state* ptr2 = dynamic_cast<op_belongs_to_state*>(Treesfact);
if (p)
{
switch (p)
{
case 4:
if (ptr->r != ptr2->r) return 0;
case 3:
if (ptr->e != ptr2->e) return 0;
case 2:
if (ptr->w != ptr2->w) return 0;
case 1:
```

```
if (ptr->q != ptr2->q) return 0;
}
}
}
else if (typeid(top_module) == typeid(*fact)&& typeid(*fact) == typeid(*Treesfact))
{
top_module* ptr = dynamic_cast<top_module*>(fact);
top_module* ptr2 = dynamic_cast<top_module*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else if (typeid(package_name) == typeid(*fact)&& typeid(*fact)
== typeid(*Treesfact))
{
package_name* ptr = dynamic_cast<package_name*>(fact);
package_name* ptr2 = dynamic_cast<package_name*>(Treesfact);
if (p)
{
if (ptr->q != ptr2->q) return 0;
}
}
else
{
return 0;
}
} // if (flag)
return 1;
}
```

7.4 AVLHashTable.h

```
    #pragma once
#include "AVL.h"
#include "interfacetxt.h"
#include <list>
#include <forward_list>

using namespace std;

constexpr auto N = 997;
//constexpr auto N = 1;

class HashTable
{
std::vector<AVLTree>Table;
public:

HashTable()
{
Table.reserve(N);
for (int i = 0; i < N; i++)
Table.push_back(AVLTree());
}

/// @brief loads data to the memory
/// @param Line <- fact to be loaded
void assertz(string Line);

/// @brief looks for fact
/// @param Line <- fact to look out
/// @return true if dound else false
bool findfact(string Line);

/// @brief erases data from the memory
/// @param Line <- 3 cases-examples:
/// @param "(*)" <- all data ||
/// @param "type_def(1,"something",2,*)" <- all the specific
type data and matched parameters till "*" ||
/// @param "type_def(1,"boolean",1,"standard",0,"single_t",0,0,0)"
```

```
<- all the excact copies.
void retractall(string Line);

/// @brief loads file of data to the memory
/// @param Line <- name(with extension e.x.: .txt) of file to be loaded
void consult(string Line);

/// @brief extracts all the loaded data to file(using existing file
will override its components)
/// @param Line <- name(with extension e.x.: .txt) of file to be extracted
void save(string filename);

/// @brief combines 2 simple lists into 1 simple list
/// @param T <- template parameter
/// @param list1 <- to be considered as the initial of the list
/// @param list2 <- to be considered as the sequel of the list
/// @param list3 <- to be considered as the extracted list
/// @returns list3
template<typename T>
list<T> concat(list<T> list1, list<T> list2, list<T> list3);

/// @brief extracts all the loaded data to string
list<string> exportToList();

/// @brief extracts all the loaded data to forward list of strings
forward_list<string> exportToFList();

/// @brief extracts all the loaded data to strings separated with coma
string exportToString();

/// @brief extracts all the matched data with Line to simple list of strings
/// @param Line <- fact to be compared
list<string> matchedToList(string Line);

/// @brief extracts all the matched data with Line to forward list of strings
/// @param Line <- fact to be compared
forward_list<string> matchedToFList(string Line);
}HT;

void HashTable::assertz(string Line)
{
```

```
string factInput;
int Treepos;
size_t  fullhash;
fullhash = (hash<string>{}(factInput));
Treepos = fullhash % N;
Table[Treepos].Root = Table[Treepos].Insert(Table[Treepos].Root,
makeInstanceOf(Line), fullhash);
Table[Treepos].Root = Table[Treepos].balance(Table[Treepos].Root);
}

bool HashTable::findfact(string Line)
{
size_t Treepos = 0;
bool flag = false;
bool* ptrflg = &flag;

if (Line.find("*", 0) != Line.npos)
{
factstar fs;
fs = makefactstar(Line);
factstar* ptrfs = &fs;
for (int i = N - 1; i >= 0; i--)
{
if (Table[i].NumNodes)
{
readnodes(Table[i].Root, ptrfs, ptrflg);
if (flag) return true;
}
}
}
else
{
size_t Hash = (hash<string>{}(Line));
Treepos = Hash % N;
if (Table[Treepos].NumNodes)
{
readnodes(Table[Treepos].Root, Hash, ptrflg);
if (flag) return true;
}
}
return false;
```

```
}

void HashTable::retractall(string Line)
{
bool duplicate = false;
bool* ptrdup = &duplicate;
size_t Factcount = 0, FactcountTree = 0, Treepos = 0;

for (int k = 0; k < N; k++)
FactcountTree += Table[k].NumNodes;

if (!FactcountTree)
{
return;
}
else if (Line == "(*)")
{
for (int i = N - 1; i >= 0; i--)
{
if (Table[i].NumNodes)
{
Table[i].~AVLTree();
}
}
}
else if (Line.find("*", 0) != Line.npos)
{
factstar fs;
fs = makefactstar(Line);
factstar* ptrfs = &fs;
for (int i = N - 1; i >= 0; i--)
{
if (Table[i].NumNodes)
{
while (!duplicate)//check for duplicate
{
*ptrdup = true;
Table[i].Root = Table[i].delNode(Table[i].Root, ptrfs, ptrdup);
Table[i].Root = Table[i].balance(Table[i].Root);
}
*ptrdup = false;
```

```
}
}
}
else
{
size_t Hash = (hash<string>{}(Line));
Treepos = Hash % N;
while (!duplicate)//check for duplicate
{
*ptrdup = true;
Table[Treepos].Root = Table[Treepos].delNode(Table[Treepos].Root,
Hash, ptrdup);
Table[Treepos].Root = Table[Treepos].balance(Table[Treepos].Root);
}
*ptrdup = false;
}
}

void HashTable::consult(string Line)
{
vector<string>AllLines;
string ALine;
int Treepos{};
size_t fullhash;
ifstream File;

File.open(Line , fstream::in);

if (File.is_open())
{
while (getline(File, ALine))
{
AllLines.push_back(ALine);
}
File.close();
AllLines.shrink_to_fit();
for (int i = 0; i < AllLines.size(); i++)
{
fullhash = (hash<string>{}(AllLines[i]));
Treepos = fullhash % N;
Table[Treepos].Root = Table[Treepos].Insert(Table[Treepos].Root,
```

```
makeInstanceOf(AllLines[i]), fullhash);
Table[Treepos].Root = Table[Treepos].balance(Table[Treepos].Root);
}
}
}

void HashTable::save(string filename)
{
size_t ts, Factcount = 0;
vector<string>AllLines;
vector<string>* ptr = &AllLines;

for (int i = 0; i < N; i++)
Factcount += Table[i].NumNodes;
if (Factcount)
{
fstream     File(filename, ios::out | ios::in | ios::trunc);

if (!File.is_open())
return;

for (int i = 0; i < N; i++)
{
Table[i].storeToVector(Table[i].Root, ptr);
}

sort(AllLines.begin(), AllLines.end());

ts = AllLines.size();
for (int i = 0; i < ts; i++)
{
File << AllLines[i] << endl;
}
File.close();
}
}

template<typename T>
list<T> HashTable::concat(list<T> list1, list<T> list2, list<T> list3)
{
if (!list1.empty())
```

```
for (T x : list1)
list3.push_back(x);
if (!list2.empty())
for (T x : list2)
list3.push_back(x);
return list3;
}

list<string> HashTable::exportToList()
{
list<string> result{};

for (int i = 0; i < N; i++)
{
if (Table[i].Root != nullptr)
readnodes(Table[i].Root, &result);
}
return result;
}

forward_list<string> HashTable::exportToFList()
{
forward_list<string> result{};

for (int i = 0; i < N; i++)
{
if (Table[i].Root != nullptr)
readnodes(Table[i].Root, &result);
}
return result;
}

string HashTable::exportToString()
{
string result{};

for (int i = 0; i < N; i++)
{
if (Table[i].Root != nullptr)
readnodes(Table[i].Root, &result);
}
```

```
return result;
}

list<string> HashTable::matchedToList(string Line)
{
list<string>result{};

factstar fc;
fc = makefactstar(Line);
for (int i = 0; i < N; i++)
{
if(Table[i].Root != nullptr)
{
readnodes(Table[i].Root, &fc, &result);
}
}
return result;
}

forward_list<string> HashTable::matchedToFList(string Line)
{
forward_list<string>result{};

factstar fc;
fc = makefactstar(Line);
for (int i = 0; i < N; i++)
{
if (Table[i].Root != nullptr)
{
readnodes(Table[i].Root, &fc, &result);
}
}
return result;
}

void readnodes(AVLTree::Node* n, list<string>* ptr)
{
if (n->Left != nullptr)
readnodes(n->Left, ptr);
if (n->Right != nullptr)
readnodes(n->Right, ptr);
```

```
ptr->push_back(makeStringOf(n->Data));
}

void readnodes(AVLTree::Node* n, factstar* ptr, list<string>* ptrL)
{
if (n->Left != nullptr)
readnodes(n->Left, ptr, ptrL);
if (n->Right != nullptr)
readnodes(n->Right, ptr, ptrL);
if (matchfactsstar(n->Data, ptr))
ptrL->push_back(makeStringOf(n->Data));
}

void readnodes(AVLTree::Node* n, factstar* ptr, forward_list<string>* ptrFL)
{
if (n->Left != nullptr)
readnodes(n->Left, ptr, ptrFL);
if (n->Right != nullptr)
readnodes(n->Right, ptr, ptrFL);
if (matchfactsstar(n->Data, ptr))
ptrFL->push_front(makeStringOf(n->Data));
}

void readnodes(AVLTree::Node* n, forward_list<string>* ptr)
{
if (n->Left != nullptr)
readnodes(n->Left, ptr);
if (n->Right != nullptr)
readnodes(n->Right, ptr);
ptr->push_front(makeStringOf(n->Data));
}

void readnodes(AVLTree::Node* n, string* ptr)
{
if (n->Left != nullptr)
readnodes(n->Left, ptr);
if (n->Right != nullptr)
readnodes(n->Right, ptr);
*ptr += makeStringOf(n->Data);
*ptr += ",";
}
```

```
void readnodes(AVLTree::Node* n, factstar* ptr, bool* flag)
{
if (matchfactsstar(n->Data, ptr))
*flag = true;
if (!(*flag) && n->Left != nullptr)
readnodes(n->Left, ptr, flag);
if (!(*flag) && n->Right != nullptr)
readnodes(n->Right, ptr, flag);
}

void readnodes(AVLTree::Node* n, size_t hashv, bool* flag)
{
if (n == nullptr) return;
if (n->HashV == hashv)
{
*flag = true;
return;
}
if (n->HashV > hashv && !(*flag))
readnodes(n->Left, hashv, flag);
if (n->HashV < hashv && !(*flag))
readnodes(n->Right, hashv, flag);
}
```

Λίστα Σχημάτων