

# Εισαγωγή:

Ο Μεταφραστής (compiler) ονομάζεται ένα πρόγραμμα υπολογιστή που διαβάζει κώδικα γραμμένο σε μια γλώσσα προγραμματισμού (γλώσσα υλοποίησης η οποία είναι γλώσσα υψηλού επιπέδου ,στην περίπτωση μας *python*) και τον μεταφράζει σε ισοδύναμο κώδικα σε μια άλλη γλώσσα προγραμματισμού (χαμηλότερου επιπέδου). Το κείμενο της εισόδου ονομάζεται αρχικό πρόγραμμα(Είναι η γλώσσα την οποία θέλουμε να υλοποιήσουμε), ενώ η έξοδος του προγράμματος τελικός κώδικας παράγεται στην τελική γλώσσα (είναι η γλώσσα στην οποία θέλουμε να μεταγλωττιστεί το πρόγραμμα), συνήθως είναι η γλώσσα μηχανής κάποιου επεξεργαστή.

Στην συγκεκριμένη αναφορά κάνουμε λόγο για τον μεταφραστή της γλώσσας προγραμματισμού *Cimple*. Για την καλύτερη κατανόηση του *cimple\_4027\_4085* την χωρίζουμε σε 4 φάσεις , όπου κάθε μια από αυτές περιέχουν λεπτομέρειες σχετικά με την υλοποίηση του μεταφραστή .Ονομαστικά οι φάσεις είναι οι εξής:

- 1) Λεκτική-Συντακτική Ανάλυση
- 2) Παραγωγή Ενδιάμεσου Κώδικα
- 3) Πίνακας Συμβόλων
- 4) Παραγωγή Τελικού Κώδικα

## Το αλφάβητο της *Cimple* αποτελείται από:

- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,...,Z και a,...,z),
- τα αριθμητικά ψηφία (0,...,9),
- τα σύμβολα των αριθμητικών πράξεων (+, -, \*, /),
- τους τελεστές συσχέτισης (<, >, =, <=, >=, <>)
- το σύμβολο ανάθεσης (:=)
- τους διαχωριστές (;, “,”),

- τα σύμβολα ομαδοποίησης ([, ], (, ) , { , })
- του τερματισμού του προγράμματος (.)
- του διαχωρισμού σχολίων (#)
- και της λέξεις(παρακάτω εικόνα)

Οι δεσμευμένες λέξεις είναι:

program	declare				
if	else	while			
switchcase	forcase	incase	case	default	
not	and	or			
function	procedure	call	return	in	inout
input	print				

Για να θέσουμε σε λειτουργία τον compiler (cimple 4027 4085) θα πρέπει πρώτα να κάνουμε compile το πρόγραμμα σε pytho και μετά να δώσουμε σαν input στο τερματικό το όνομα του αρχείου .ci που θέλουμε να κάνουμε compile ,όπως φαίνεται παρακάτω.

```
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
```

```
-
```

```
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.ci
Lexical and Syntax Analyse Complete Successfull!
Intermediate code has been completed Successfully!
Finalcode has been completed Successfully!
```

# 1) Λεκτική-Συντακτική Ανάλυση:

Στην παρούσα φάση χρησιμοποιούμε 2 εργαλεία :

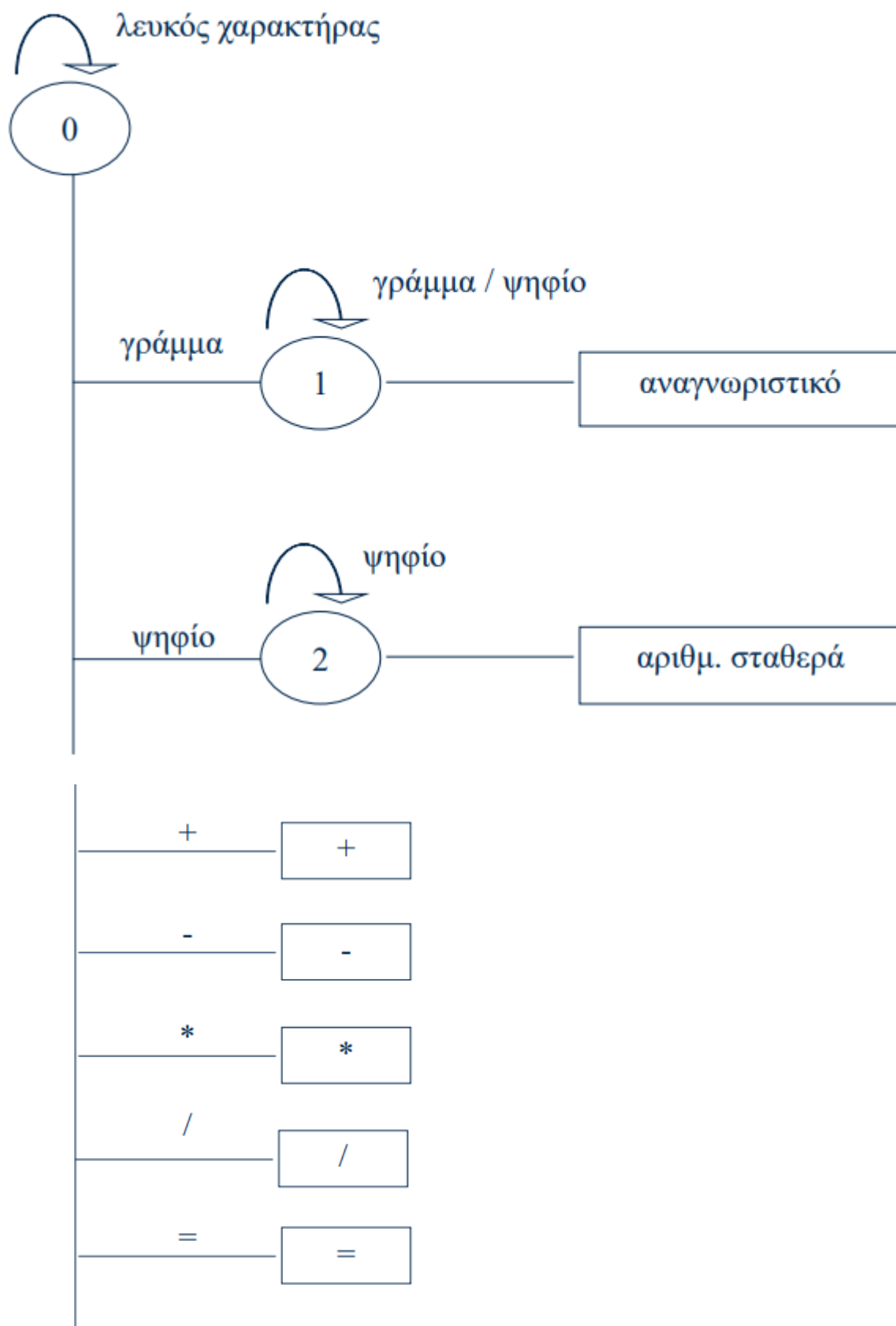
1)Λεκτικός Αναλυτής (Στην δικιά μας περίπτωση lex)

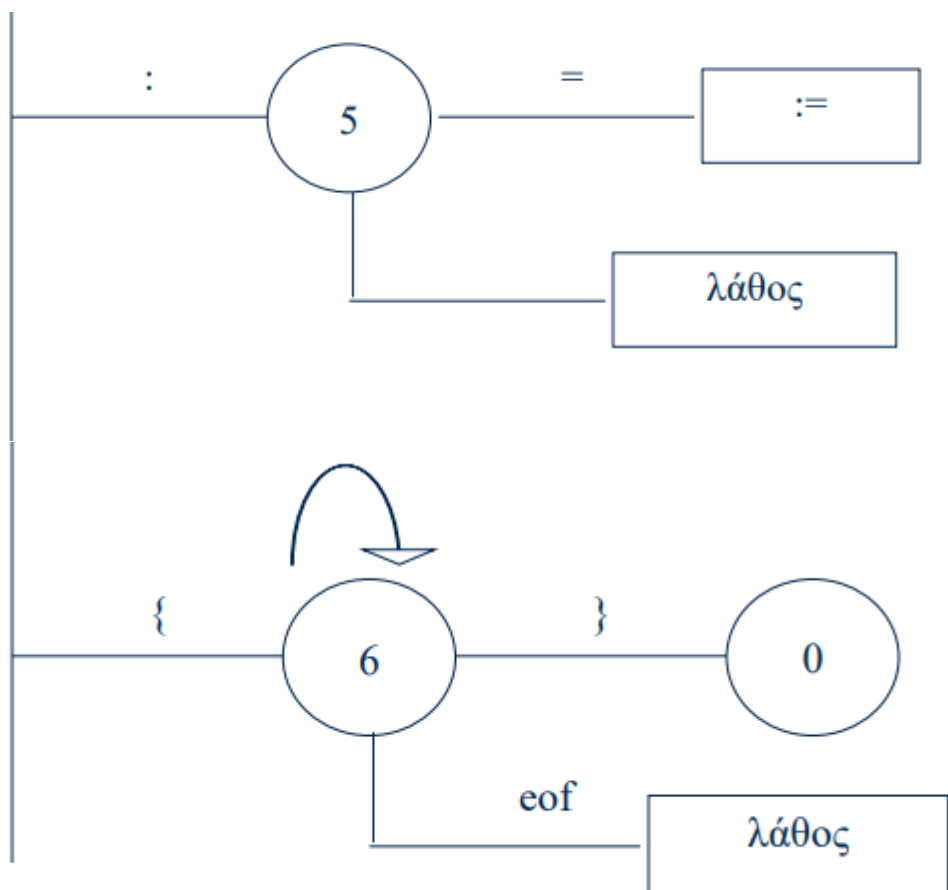
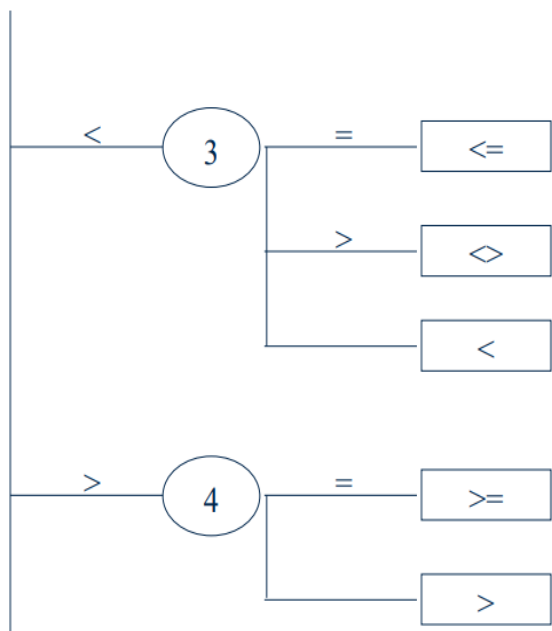
2)Συντακτικός Αναλυτής (Στην δικιά μας περίπτωση suntaktikos)

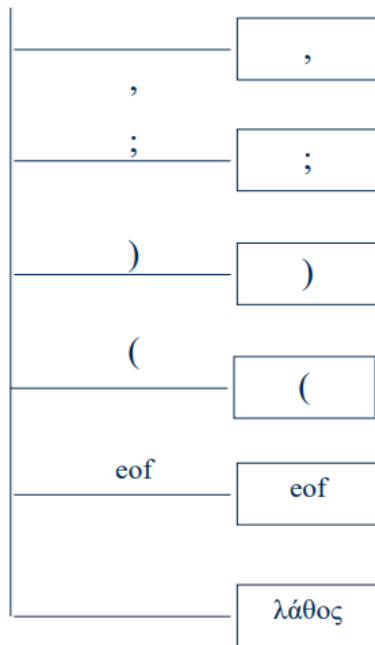
## Λεκτικός Αναλυτής

Ο λεκτικός αναλυτής καλείται ως συνάρτηση από το συντακτικό αναλυτή διαβάσει γράμμα-γράμμα το πηγαίο πρόγραμμα και κάθε φορά που καλείται επιστρέφει την επόμενη λεκτική μονάδα(στην περίπτωση μας το token) και επιστρέφει στο συντακτικό αναλυτή ,έναν ακέραιο που χαρακτηρίζει τη λεκτική μονάδα και την λεκτική μονάδα.

Ο λεκτικός αναλυτής(lex) αποτελεί στο κώδικά μας μια συνάρτηση του συντακτικού αναλυτή όπου λειτουργεί σαν αυτόματο καταστάσεων .Δηλαδή ξεκινάει από μια αρχική κατάσταση και εισάγει κάθε φορά τον επόμενο χαρακτήρα μέχρι να φτάσουμε στην τελική κατάσταση .Αφού φτάσουμε επιστρέφουμε στο suntaktiko την επόμενη λεκτική μονάδα και τον ακέραιο που τον χαρακτηρίζει .Στα παρακάτω screenshot το αυτόματο λειτουργίας(Cimple) περιγράφει την λειτουργία του λεκτικού αναλυτή.







Αφού τρέξουμε το πρόγραμμά μας , αρχικά ανοίγει το αρχείο Cimple(πχ area1.ci) που δίνουμε σαν όρισμα στο cmd και καλεί το suntaktikos. Ο suntaktikos καλεί το lex όπου ο lex διαβάζει το πρώτο χαρακτήρα του αρχείου και συμφώνα με το αυτόματο καταστάσεων πηγαίνει στην αντίστοιχη κατάσταση ,Στην περίπτωση που ο λεκτικός αναλυτής διαβάζει τον χαρακτήρα(.) μεταβαίνει σε τερματική κατάσταση και τερματίζει τον συντακτικό αναλυτή.

Παρακάτω έχουμε κάποια παραδείγματα της λειτουργιά του lex

```
Command Prompt
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
areal.ci
Error numbers are between -(2^32)-1 and (2^32)-1
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>_

C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project\areal.ci - Notepad++
16         return (ans);
17     };
18     if(b + a >= 0 ){
19         ans:=a*square(in c);
20         return(ans);
21     };
22     if(c=0 ){
23         return(square(in c));
24     }
25     else{
26         | | |
27         | | |
28         | | |
29         t:=4298967296;
30         return (ans);
31     };
32
33     while(area(in i+a,inout b) >= num){
34         i:=i+1;
35     };
36
```

Στην παραπάνω εικόνα φαίνεται το μήνυμα λάθους του lex που εμφανίζει όταν εισάγουμε αριθμό με τιμή που δεν βρίσκεται ενδιάμεσα από  $[-(2^{31}-1) < x < (2^{31}-1)]$   
 $*(2^{31}-1) = 4294967295$

```
Error number cant be more than (2^32)-1
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
areal.ci
Lexical and Syntax Analyse Complete Successfull!
Intermediate code has been completed Successfully!
Finalcode has been completed Successfully!
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>_

11 declare ans,t;
12 {
13
14     if(a*b = 0 and c + a >= 0 ){
15         ans:=a*b;
16         return (ans);
17     };
18     if(b + a >= 0 ){
19         ans:=a*square(in c);
20         return(ans);
21     };
22     if(c=0 ){
23         return(square(in c));
24     }
25     else{
26         | | |
27         | | |
28         | | |
29         t:=4294967294;
30         return (ans);
31     };
32 }
```

```
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.ci
Error variable name must be less than 30 chars! 3
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>_
```

```
1 program area
2     declare x1,x2,x3;
3     declare y1,y2,y3;
4     declare a,b,c,answerasdfghjklpoiuytreswqsadfd;
5     function square(in x)
6     {
7         return (x*x);
8     }
9
10    function area(in a,inout b)
```

Στην παραπάνω εικόνα βλέπουμε το μήνυμα λάθους που εμφανίζει ο lex όταν συναντήσει αναγνωριστικό με περισσότερους από 30 χαρακτήρες

```
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.ci
Error unclosed comments line: 55
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>_
```

```
24
25     else{
26
27         | | |
28
29         t:=a+b;
30         return (ans);
31
32     };
33     while(area(in i+a,inout b) >= num){
34         i:=i+1;
35     };
36
37 }
38 #main function
39 {
40     input(x1);
41     input(x2);
42     input(x3);
43     input(y1);
44     input(y2);
45     input(y3);
46     a:=x2/x1*x0;
47     b:=y2*y1;
48     c:=y3-x3;
49     answer:= area(in a,inout b);
50     print(answer);
51
52     answer:= area(in a,inout b);
53     print(answer);
54     print(a);
55     call area(in a,inout b);
56 }.
```

Στην παραπάνω εικόνα βλέπουμε το μήνυμα λάθους που εμφανίζει ο lex όταν συναντήσει ανοιχτά σχόλια (γραμμή 38)



\*Σχετικά με τα σχόλια λειτουργούν κανονικά απλά δεν έχουμε προσθέσει μήνυμα λάθους στην περίπτωση που υπήρχαν σχόλια σε εντολές που περιέχουν δεσμευμένες λέξεις

## Συντακτικός Αναλυτής

Επιπρόσθετα ο συντακτικός αναλυτής είναι μια συνάρτηση που υλοποιεί τη συντακτική ανάλυση μια γλώσσας προγραμματισμού .Δηλαδή δέχεται ως input μια ακολουθία λεκτικών μονάδων(token) ενός προγράμματος και ελέγχει αν το πρόγραμμα είναι σύμφωνο με την γραμματική της γλώσσας. Παρακάτω αναφέρουμε συνοπτικά τις συναρτήσεις που υλοποιήσαμε στο κώδικα μας που είναι ανάλογες των κανόνων που απαρτίζουν την γραμματική της Cimple. Οι συναρτήσεις είναι οι εξής:

- program:** Αν το token(λεκτική μονάδα) έχει την λέξη program τότε πάμε στο επόμενο token που καλεί την συνάρτηση ID και εφόσον περάσει από αυτή τότε πάμε στο επόμενο token και καλεί και το block, Αν το token δεν έχει την λέξη program εμφανίσει μήνυμα λάθους.

- block:** Με το που μπαίνει στην συνάρτηση block καλεί τις συναρτήσεις declarations,subprograms,statements.

- declarations:** Αντιστοιχεί στις αρχικοποιήσεις των μεταβλητών. Αν το token έχει την λέξη declare πηγαίνουμε στο επόμενο token και καλούμε την συνάρτηση varlist. Αν αυτό που επιστρέψει η varlist δεν είναι <;> (δηλαδή declare a,b;) τότε να πετάει αντίστοιχο μήνυμα λάθους.

- varlist:** Το token που εισέρχεται πηγαίνει στην ID και εφόσον πάει στο επόμενο ,αν αυτό είναι <, > ελέγχει το επόμενο token. Αν το επόμενο είναι id τότε συνεχίζει την λούπα ,αλλιώς τυπώνει μήνυμα λάθους .Επίσης αν υπάρχουν μεταβλητές που δε χωρίζονται με κόμμα πάλι υπάρχει σχετικό μήνυμα λάθους.

•**subprograms:** Αν το token έχει την λέξη function ή procedure τότε καλούμε την συνάρτηση subprograms.

•**subprogram:** Αν το token έχει την λέξη function ή procedure τότε καλούμε το επόμενο token. Το επόμενο token καλεί την συνάρτηση ID, ανοίγουμε παρένθεση, πάει στο επόμενο token, καλείτε η συνάρτηση formalparlist, κλείνει η παρένθεση και στο επόμενο token καλεί την συνάρτηση block. Σε περίπτωση που λείπει κάποια παρένθεση υπάρχει σχετικό μήνυμα λάθους.

•**formalparlist:** Καλούμε την συνάρτηση formalparitem .Αν το token που επιστρέφει εκείνη περιέχει το ‘,’ πάμε στο επόμενο token και καλούμε την συνάρτηση formalparitem.

•**formalparitem:** Αρχικά αν το token έχει το in ή inout πάμε στο επόμενο token διαφορετικά τυπώνεται το αντίστοιχο μήνυμα λάθους. Το επόμενο token καλεί την συνάρτηση ID και πάμε στο επόμενο token.

•**statements:** Σχετίζεται με τις εκφράσεις που περιέχουν συνδεδεμένες λέξεις. Αν συναντήσει <{>, τότε πάει στο επόμενο token και καλεί την statement. Αν αυτή επιστρέψει ‘,’ τότε πάμε στο επόμενο token και ξανακαλεί την statement μέχρι να βρει ένα token ίσο με <}> ή <.> ή <;>. Τέλος ,αν η συνάρτηση τελειώνει χωρίς το token που επιστρέφει να είναι <}> έχουμε μήνυμα λάθους.

•**statement:** Περιέχει συναρτήσεις που κάθε μία αντιστοιχεί σε μία έκφραση που περιέχει συνδεδεμένες λέξεις. Αν το token που παίρνει αυτή η συνάρτηση είναι το <:=> εκτελεί τη συνάρτηση assingStat. Αν το token που παίρνει αυτή η συνάρτηση είναι το <if> εκτελεί τη συνάρτηση ifStat. Αν το token που παίρνει αυτή η συνάρτηση είναι το <while> εκτελεί τη συνάρτηση whileStat. Αν το token που παίρνει αυτή η συνάρτηση είναι το <switchcase> εκτελεί τη συνάρτηση switchcaseStat. Αν το token που παίρνει αυτή η συνάρτηση είναι το <forecase> εκτελεί τη συνάρτηση forecaseStat. Αν το token που παίρνει αυτή η συνάρτηση είναι το <incase> εκτελεί τη συνάρτηση incaseStat. Αν το token που παίρνει αυτή η συνάρτηση είναι το <return> εκτελεί τη συνάρτηση returnStat. Αν το token

που παίρνει αυτή η συνάρτηση είναι το `<call>` εκτελεί τη συνάρτηση `callStat`.

Αν το token που παίρνει αυτή η συνάρτηση είναι το `<print>` εκτελεί τη συνάρτηση `printStat`. Αν το token που παίρνει αυτή η συνάρτηση είναι το `<input>` εκτελεί τη συνάρτηση `inputStat`.

•**assignStat:** Η συνάρτηση καλεί την ID .Αν το επόμενο token περιέχει το σύμβολο `':='` τότε πηγαίνει στο επόμενο token και καλεί την συνάρτηση `expression`, διαφορετικά τυπώνουμε αντίστοιχο μήνυμα λάθους .Τώρα αν εκείνη δεν επιστρέψει `<;>` τότε υπάρχει σχετικό μήνυμα λάθους(πχ `[t:=a+b;])`).

•**ifStat:** Η συνάρτηση αν πάρει το token `<if>` τότε πηγαίνει στο επόμενο token και αν αυτό είναι `<( >` ,ξαναπηγαίνει στο επόμενο και μπαίνει σε μια λούπα όπου αν το token είναι διαφορετικό από το `<)>` καλεί την `condition`(αυτό γίνεται έτσι ώστε να μπορεί ο compiler να περάσει πολλαπλά `and` και `or` μέσα στο `if`). Τώρα αν από την `condition` επιστραφεί το token `<)>` θα πάει στο επόμενο token και θα καλέσει πρώτα την `statements` και μετά την `elsepart`.

•**elsepart:** Η συνάρτηση αν πάρει το token `<else>` τότε πηγαίνει στο επόμενο token και καλεί την `statements`.

•**whileStat:** Η συνάρτηση αν πάρει το token `<while>` τότε πηγαίνει στο επόμενο token και καλεί την `condition`. Τώρα αν από την `condition` επιστραφεί το token `<)>` θα πάει στο επόμενο token και θα καλέσει την `statements`. Τώρα υπάρχουν και δύο μηνύματα λάθους τα οποία είναι όταν δεν κλείνει η παρένθεση του `while` ,δηλαδή(`[while(a>0)]`) ,και όταν μετά την κλήση της `statements` δεν επιστραφεί το token `<;>`.

•**switchcaseStat:** Η συνάρτηση αν πάρει το token <switchcase> πηγαίνει στο επόμενο token.

1)Αν το επόμενο token είναι το case θα πάει στο επόμενο token .Αν συναντήσει <( > θα πάει στο επόμενο και θα καλέσει την condition .Αν η condition επιστρέψει το token < > θα πάει στο επόμενο token και θα καλέσει την statements .Αν η statements επιστρέψει το token <case> θα ξανατρέξει τη διαδικασία .Τώρα αν μετά το token <case> δε βρει <( > θα επιδείξει μήνυμα λάθους ,καθώς και όταν δεν συναντήσει < > μετά την κλήση της condition.2)Αν το επόμενο token είναι το default θα πάει στο επόμενο token και θα καλέσει την statement.

•**forcaseStat:** Η συνάρτηση αν πάρει το token <forcase> πηγαίνει στο επόμενο token.

1)Αν το επόμενο token είναι το case θα πάει στο επόμενο token .Αν συναντήσει <( > θα πάει στο επόμενο και θα καλέσει την condition .Αν η condition επιστρέψει το token < > θα πάει στο επόμενο token και θα καλέσει την statements .Αν η statements επιστρέψει το token <case> θα ξανατρέξει τη διαδικασία .Τώρα αν μετά το token <case> δε βρει <( > θα επιδείξει μήνυμα λάθους ,καθώς και όταν δεν συναντήσει < > μετά την κλήση της condition.2)Αν το επόμενο token είναι το default θα πάει στο επόμενο token και θα καλέσει την statement.(Παραλείψαμε την κλήση της statement)

•**incaseStat:** Η συνάρτηση αν πάρει το token <incase> πηγαίνει στο επόμενο token.

Αν το επόμενο token είναι το case θα πάει στο επόμενο token .Αν συναντήσει <( > θα πάει στο επόμενο και θα καλέσει την condition .Αν η condition επιστρέψει το token < > θα πάει στο επόμενο token και θα καλέσει την statements .Αν η statements επιστρέψει το token <case> θα ξανατρέξει τη διαδικασία .Τώρα αν μετά το token <case> δε βρει <( > θα επιδείξει μήνυμα λάθους ,καθώς και όταν δεν συναντήσει < > μετά την κλήση της condition.

•**returnStat:** Η συνάρτηση αν πάρει το token <return> θα πάει στο επόμενο token. Αν αυτό είναι το <( > θα πάει πάλι στο επόμενο token και θα καλέσει την expression. Αν αυτή επιστρέψει το token <)> πηγαίνει στο επόμενο token (αν δεν υπήρχε βγαίνει σχετικό μήνυμα λάθους). Αν τώρα το πρόγραμμα ήταν στην περίπτωση (του default) και έβλεπε το token <;> θα πήγαινε στο επόμενο, ώστε να περάσει και τα 2 <;>, όπως πχ (default return(x\*2);;).

•**callStat:** Η συνάρτηση αν πάρει το token <call> πηγαίνει στο επόμενο token. Αν αυτό είναι id, τότε πηγαίνει στο επόμενο. Αν το επόμενο είναι το <( > (αν δεν είναι θα υπάρξει σχετικό μήνυμα λάθους), τότε πηγαίνει πάλι στο επόμενο και καλεί την actualparlist. Αν το token που θα επιστρέψει αυτή είναι το <)> θα πάει στο επόμενο token κανονικά, αν όμως δεν είναι αυτό θα επιστρέψει μήνυμα λάθους και θα τερματίσει ο compiler.

•**printStat:** Η συνάρτηση αν πάρει το token <print> πηγαίνει στο επόμενο token. Αν το επόμενο είναι το <( > (αν δεν είναι θα υπάρξει σχετικό μήνυμα λάθους), τότε πηγαίνει πάλι στο επόμενο και καλεί την expression. Αν το token που θα επιστρέψει αυτή είναι το <)> θα πάει στο επόμενο token κανονικά, αν όμως δεν είναι αυτό θα επιστρέψει μήνυμα λάθους και θα τερματίσει ο compiler.

•**inputStat:** Η συνάρτηση αν πάρει το token <input> πηγαίνει στο επόμενο token. Αν το επόμενο είναι το <( > (αν δεν είναι θα υπάρξει σχετικό μήνυμα λάθους), τότε πηγαίνει πάλι στο επόμενο και καλεί την ID. Αν το token που θα επιστρέψει αυτή είναι το <)> θα πάει στο επόμενο token κανονικά, αν όμως δεν είναι αυτό θα επιστρέψει μήνυμα λάθους και θα τερματίσει ο compiler. Επίσης, αν το επόμενο token δεν είναι το <;> τότε θα υπάρξει σχετικό μήνυμα λάθους.

•**actualparlist:** Η συνάρτηση αυτή καλεί την actualparitem και αν αυτή επιστρέψει το token <,> θα πάει στο επόμενο και θα ξανακαλέσει την actualparitem.

•**actualparitem:** Η συνάρτηση αυτή αν το token που θα δεχτεί είναι το <in> θα πάει στο επόμενο token και θα καλέσει την expression, αν όμως συναντήσει το <inout> θα πάει στο επόμενο και θα πάει να ελέγξει αν είναι id.

•**condition:** Η συνάρτηση αυτή καλεί την boolterm και αν αυτή επιστρέψει το token <or> θα πάει στο επόμενο token και θα ξανακαλέσει την boolterm (δηλαδή μια λούπα μέχρι η boolterm να μην επιστρέψει <or>).

•**boolterm:** Η συνάρτηση αυτή καλεί την boolfactor και αν αυτή επιστρέψει το token <and> τότε πηγαίνει στο επόμενο token και καλεί στην boolfactor (μέχρι να μην επιστρέψει token=end).

•**boolfactor:** Η συνάρτηση αυτή αν δεχτεί το token <not> θα πάει στο επόμενο token. **1)** Αν το επόμενο είναι το <[> τότε πηγαίνει στο επόμενο και καλεί την condition (αν δεν είναι εμφανίζεται σχετικό μήνυμα λάθους). Αν η condition επιστρέψει το token <]> τότε πηγαίνει στο επόμενο token. (αν δεν είναι εμφανίζεται σχετικό μήνυμα λάθους). **2)** Αν το επόμενο είναι το <[> , τότε πηγαίνει στο επόμενο token και καλεί την condition , μέχρι αυτή να επιστρέψει το token <]> , και αν αυτή το επιστρέψει πηγαίνει στο επόμενο token (αν όχι εμφανίζεται σχετικό μήνυμα λάθους). **3)** Καλεί την expression , μετά την REL\_OP και τέλος ξανά την expression.

•**expression:** Η συνάρτηση αυτή αντιστοιχεί στα στοιχεία της πρόσθεσης και της αφαίρεσης. Όταν μπαίνει στη συνάρτηση καλεί την optionalSign. Αν αυτή επιστρέψει το token <+> ή το <-> πηγαίνει στο επόμενο token και καλεί την ADD\_OP και την term . Τώρα αν μετά το token υπάρχει <;> τότε υπάρχει σχετικό μήνυμα λάθους.

•**ADD\_OP:** Αντιστοιχεί στην πρόσθεση και στην αφαίρεση. Δηλαδή, αν το token είναι το <+> απλά πηγαίνει στο επόμενο token και αν το token είναι το <-> απλά πηγαίνει στο επόμενο token.

•**term:** Η συνάρτηση αυτή αντιστοιχεί στα στοιχεία του πολλαπλασιασμού και της διαίρεσης. Όταν μπαίνει στη συνάρτηση καλεί την factor. Αν αυτή επιστρέψει το token <\*> ή το </> πηγαίνει στο επόμενο token και καλεί την MUL\_OP και την factor. Τώρα αν μετά το token υπάρχει <;> τότε υπάρχει σχετικό μήνυμα λάθους.

•**factor:** Όταν μπαίνει στη συνάρτηση αυτό περιμένει κάποια σταθερά ή το token <(>.**1)** Αν λοιπόν εισέλθει κάποια σταθερά και το επόμενο token είναι το <( > τότε πηγαίνει στο επόμενο token και καλεί την expression. Αν η expression επιστρέψει το token <)> τότε πηγαίνει στο επόμενο. Αν όχι τότε εμφανίζεται σχετικό μήνυμα λάθους.**2)** Σε αυτή την περίπτωση, αφού το token που θα εισέλθει είναι σταθερά και το επόμενο δεν είναι <( >, τότε ελέγχει αν είναι id και μετά καλεί την idtail.

•**idtail:** Αν λάβει το token <( > απλά πηγαίνει στο επόμενο και καλεί την actualparlist. Αν αυτή επιστρέψει τη λεκτική μονάδα <)> τότε πηγαίνει στην επόμενη. Αν όχι τότε επιστρέφει σχετικό μήνυμα λάθους.

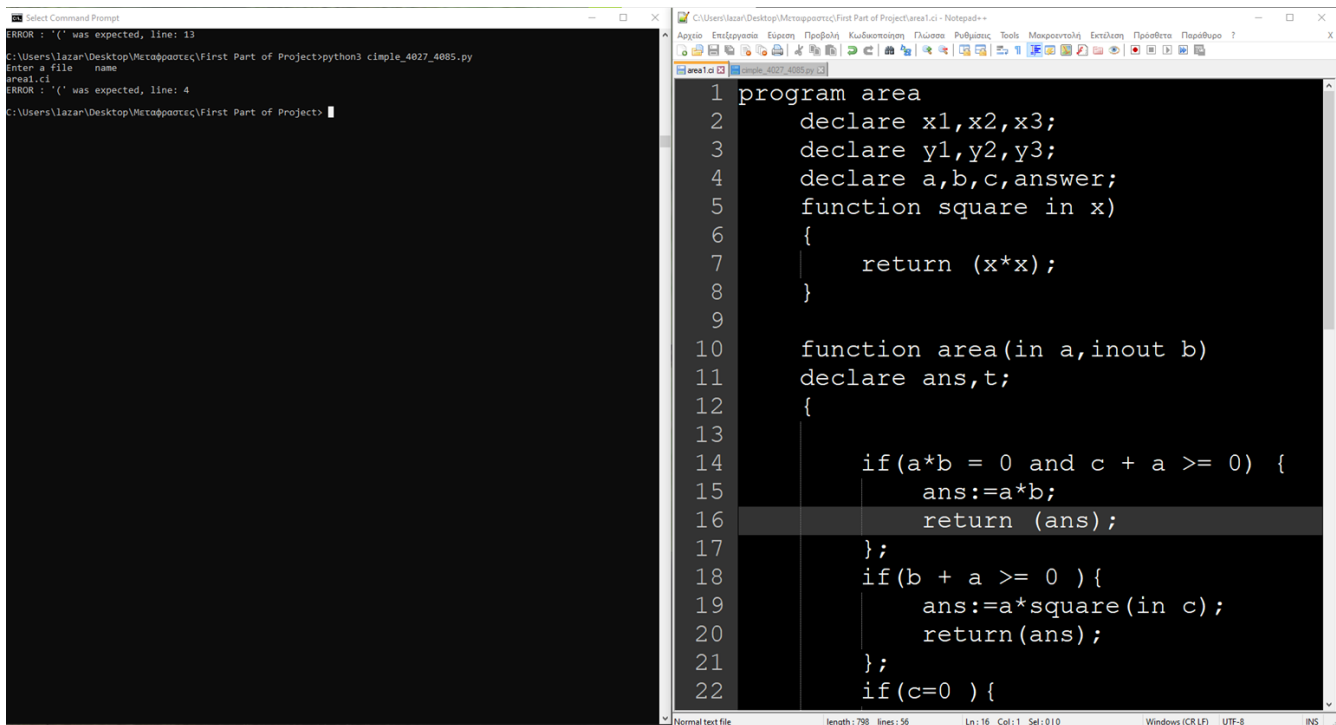
•**optionalSign:** Απλά καλεί την ADD\_OP

•**REL\_OP:** Αυτή η συνάρτηση αντιστοιχεί στους τελεστές συνθήκης. Αν το token είναι «=» πηγαίνει στο επόμενο token. Αν το token είναι «<=» πηγαίνει στο επόμενο token. Αν το token είναι «>=» πηγαίνει στο επόμενο token. Αν το token είναι «>» πηγαίνει στο επόμενο token. Αν το token είναι «<» πηγαίνει στο επόμενο token. Αν το token είναι «<>» πηγαίνει στο επόμενο token.

•**INTEGER:** Αντιστοιχεί στους αριθμούς. Αν συναντήσει κάποιο ψηφίο, τότε πηγαίνει στο επόμενο token.

•**ID:** Αντιστοιχεί στις μεταβλητές. Αν λοιπόν το 1<sup>ο</sup> στοιχείο του token είναι χαρακτήρας τότε, είναι λέξη άρα μεταβλητή, πηγαίνει στο επόμενο token. Αυτό βέβαια προϋποθέτει ότι το token δεν ανήκει στις δεσμευμένες λέξεις.

Στα παρακάτω στιγμιότυπα παραθέτουμε μερικά παραδείγματα από τη λειτουργία του συντακτικού αναλυτή.



The image shows two windows side-by-side. The left window is a Command Prompt titled 'Select Command Prompt' showing the execution of a C program. It displays an error message: 'ERROR: '(' was expected, line: 13'. The right window is a Notepad++ editor titled 'C:\Users\lazar\Desktop\Μεταφραστής\First Part of Project\area1.c - Notepad++'. It contains a C program with the following code:

```
1 program area
2     declare x1,x2,x3;
3     declare y1,y2,y3;
4     declare a,b,c,answer;
5     function square in x)
6     {
7         return (x*x);
8     }
9
10    function area(in a,inout b)
11    declare ans,t;
12    {
13
14        if(a*b = 0 and c + a >= 0) {
15            ans:=a*b;
16            return (ans);
17        };
18        if(b + a >= 0 ){
19            ans:=a*square(in c);
20            return(ans);
21        };
22        if(c=0 ){
```

The status bar at the bottom of the Notepad++ window indicates 'length: 798 lines: 56' and 'Ln: 16 Col: 1 Sel: 0|0'.

Στο παραπάνω στιγμιότυπου υπάρχει συντακτικό λάθος στη γραμμή 5, καθώς δεν ανοίγει η παρένθεση της συνάρτησης square.



```
Command Prompt
ERROR : '(' was expected, line: 13
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.c
ERROR : '(' was expected, line: 4
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.c
ERROR : ')' was expected, line: 9
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>
```

```
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project\area1.c - Notepad++
1 program area
2 declare x1,x2,x3;
3 declare y1,y2,y3;
4 declare a,b,c,answer;
5 function square (in x)
6 {
7     return (x*x);
8
9
10 function area(in a,inout b)
11 declare ans,t;
12 {
13
14     if(a*b = 0 and c + a >= 0) {
15         ans:=a*b;
16         return (ans);
17     };
18     if(b + a >= 0) {
19         ans:=a*square(in c);
20         return(ans);
21     };
22     if(c=0 ) {
```

Στο παραπάνω στιγμιότυπο στη γραμμή 9 έχει παραληφθεί το κλείσιμο της αγκύλης }.

```
Command Prompt
Enter a file name
area1.c
Error: areaa function not defined ,line 48
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.c
ERROR : ';' was expected, line: 49
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project>
```

```
C:\Users\lazar\Desktop\Μεταφραστες\First Part of Project\area1.c - Notepad++
36
37 }
38 #main function#
39 {
40     input(x1);
41     input(x2);
42     input(x3);
43     input(y1);
44     input(y2);
45     input(y3);
46     a:=x2/x1*x0;
47     b:=y2*y1;
48     c:=y3-x3;
49     answer:= area(in a,inout b);
50     print(answer);
51
52     answer:= area(in a,inout b);
53     print(answer);
54     print(a);
55     call area(in a,inout b);
56 }
```

Στο παραπάνω στιγμιότυπο στη γραμμή 49 έχει παραληφθεί το ερωτηματικό στο τέλος της εντολής.

```
Command Prompt
C:\Users\lazar\Desktop\Μεταφράστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.c1
Error: bigarea function not defined ,line 48
C:\Users\lazar\Desktop\Μεταφράστες\First Part of Project>
```

```
area1.c1
36
37
38 #main function#
39 {
40     input(x1);
41     input(x2);
42     input(x3);
43     input(y1);
44     input(y2);
45     input(y3);
46     a:=x2/x1*x0;
47     b:=y2*y1;
48     c:=y3-x3;
49     answer:= bigarea(in a,inout b);
50     print(answer);
51
52     answer:= area(in a,inout b);
53     print(answer);
54     print(a);
55     call area(in a,inout b);
56 }
```

Στο παραπάνω στιγμιότυπο στη γραμμή 49 υπάρχει κλήση συνάρτησης η οποία δεν έχει οριστεί πουθενά .

```
Command Prompt
Error: bigarea function not defined ,line 48
C:\Users\lazar\Desktop\Μεταφράστες\First Part of Project>python3 cimple_4027_4085.py
Enter a file name
area1.c1
ERROR : ':' was expected, line: 14
C:\Users\lazar\Desktop\Μεταφράστες\First Part of Project>
```

```
area1.c1
12 {
13
14     if(a*b = 0 and c + a >= 0) {
15         ans=a*b;
16         return (ans);
17     };
18     if(b + a >= 0 ){
19         ans:=a*square(in c);
20         return(ans);
21     };
22     if(c=0 ){
23         return(square(in c));
24     }
25     else{
26
27         | | |
28
29         t:=a+b;
30         return (ans);
31
32     };
33     while(area(in i+a,inout b) >=
```

Στο παραπάνω στιγμιότυπο υπάρχει συντακτικό λάθος στη γραμμή 15 όπου χρησιμοποιείται το σύμβολο “=” για την ισότητα αντί του <:=> .

## 2) Παραγωγή Ενδιάμεσου Κώδικα

Κατά την διαδικασία μετάφρασης ενός αρχικού κώδικα σε τελικό κώδικα ένας μεταφραστής κατασκευάζει ενδιάμεσες αναπαραστάσεις .

Στην περίπτωση μας ο ενδιάμεσος κώδικας είναι ένα σύνολο από τετράδες οι οποίες αποτελούνται από 1 τελεστή και 3 τελούμενα (op,x,y,z)(πχ. +, a ,b,T\_1), όπου ο τελεστής op χρησιμοποιείται για τα x,y και το αποτέλεσμα τους αποθηκεύεται στο τελεστή z. Ο τελεστής op θα μπορεί να είναι ένα από τα (+,-,\*,/) ενώ οι τελεστές x,y μπορούν να περιέχουν ονόματα μεταβλητών ή σταθερές , ο τελεστής z μπορεί να είναι όνομα μεταβλητής. Οι τετράδες είναι αριθμημένες και κάθε τετράδα έχει μπροστά της έναν μοναδικό αριθμό που την χαρακτηρίζει.

Για την δημιουργία του ενδιάμεσου κώδικα χρησιμοποιήσαμε τις παρακάτω βοηθητικές συναρτήσεις.

- **nextquad()**: Η συγκεκριμένη συνάρτηση επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.
- **genquad()**: Η συγκεκριμένη συνάρτηση δημιουργεί την επόμενη τετράδα(op,x,y,z). Στην ουσία όμως εμείς κρατάμε και τον μοναδικό αριθμό που χαρακτηρίζει κάθε τετράδα και μαζί με τα υπόλοιπα δεδομένα τα αποθηκεύουμε σε μια λίστα (ListOfQuads1).
- **newtemp()**: Η συγκεκριμένη συνάρτηση δημιουργεί και επιστρέφει μια νέα προσωρινή μεταβλητή , που είναι της μορφής T\_1,T\_2,T\_3.
- **emptylist()**: Η συγκεκριμένη συνάρτηση δημιουργεί μια κενή λίστα ετικετών τετράδων.
- **makelist()**: Η συγκεκριμένη συνάρτηση δημιουργεί μια λίστα ετικετών τετράδων που περιέχει μόνο το x.
- **merge(list1,list2)**: Η συγκεκριμένη συνάρτηση δημιουργεί μια λίστα ετικετών τετράδων από τη συνένωση των λιστών list1,list2.

• **backpatch(list,z):** Σε αυτή η λίστα list είναι ένας δείκτης σε ετικέτα τετράδας της οποίας το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η συνάρτηση αυτή συμπληρώνει το τελευταίο τελούμενο με την ετικέτα z . Για την παραγωγή του ενδιαμέσου κώδικα πήραμε της παραπάνω συναρτήσεις και της καλούσαμε σε διαφορές συναρτήσεις(τις αναφέρουμε πιο κάτω) του συντακτικού αναλυτή. Επίσης σε αυτό το μέρος χρησιμοποιήσαμε πολλά indexes που άλλοτε επιτρέπουν στις συναρτήσεις να έχουν πρόσβαση στις δομές και άλλοτε όχι.

**block:** Αρχικά, υπάρχει ένα index το Block το οποίο ενεργοποιείται όταν η συνάρτηση φτάσει στο τέλος της, δηλαδή όταν το token που επιστρέφει η block στην program είναι το <.> και καλεί την genquad("halt "," \_ "," \_ ") και την genquad("end\_block ",name," \_ "," \_ "),όπου το name είναι το όνομα του προγράμματος ,δηλαδή της main.Επιπλέον, υπάρχει άλλον ένα block με if στη συνάρτηση με 2 index ,το g5 και το sect .Το g5 ενεργοποιείται όταν ξεκινάει η main στο αρχείο .ci και το sect υπάρχει ώστε μετά την 2<sup>η</sup> κλήση της block(αφού το πρόγραμμα .ci τερματιστεί) να μην ξαναμπεί σε αυτό το if.Αν ισχύουν τα παραπάνω θα καλέσει την genquad("begin\_block ",name," \_ "," \_ "),όπου name το όνομα του προγράμματος ,δηλαδή της main.

**Subprogram:** Αρχικά αφού συναντήσει στο πρόγραμμα .ci κάποιο function θα καλέσει την genquad("begin\_block ",id," \_ "," \_ "),όπου id το όνομα της function. Στη συνέχεια αφού τελειώσει η function θα καλέσει την genquad("end\_block ",id," \_ "," \_ "),όπου id το όνομα της function.

**assignStat:** Αρχικά, αν κατά την ισότητα συναντήσει κάποια συνάρτηση, όπως πχ.([answer:= area(in a)]) ,τότε αν η συνάρτηση έχει 1 όρισμα θα καλέσει την genquad("par ",Helpa," CV"," \_ ") , αν η Helpa μεταδίδεται με τιμή , ή την genquad("par ",Helpa," REF"," \_ "),αν η Helpa μεταδίδεται με αναφορά. Αν η συνάρτηση έχει 2 ορίσματα,όπως πχ. [answer:= area(in a,inout b)], τότε , για το πρώτο όρισμα θα καλέσει την genquad("par ",Helpa," CV"," \_ ") ή την genquad("par ",Helpa," REF"," \_ ") ,αν το Helpa μεταδίδεται

αντίστοιχα με τιμή ή με αναφορά, και για το δεύτερο θα καλέσει την `genquad("par ",Helpb," CV"," _")` ή την `genquad("par ",Helpb," REF"," _")` αν το `Helpb` μεταδίδεται αντίστοιχα με τιμή ή με αναφορά. Στη συνέχεια θα καλέσει την `genquad("par ",w," RET"," _")` όπου `w` είναι η τιμή πλέον που επιστρέφει η συνάρτηση όταν κληθεί και μετά καλείται η `genquad("call ",token.tokenString," _"," _")`, όπου `token.tokenString` το όνομα της συνάρτησης. Τώρα, έχουμε 3 περιπτώσεις σχετικά με την κλήση της `genquad` για την ισότητα : **1)** αν είναι της μορφής `πχ.[t:=a]` τότε καλεί την `genquad(":=",Eplace," _",id)` όπου `Eplace` το `t` και `id` το `a`, **2)** αν είναι της μορφής `πχ.[answer:= area(in a,inout b)]` τότε καλεί την `genquad(":=",id+" ", " _",w)` όπου `id` το `answer` και `w` αυτό που επιστρέφει η `area` όταν κληθεί, **3)** αν είναι της μορφής `πχ.[t:=a+b]` καλεί την `genquad(":=",id+" ", " _",w1)`, όπου `id` το `t` και `w1` το αποτέλεσμα της πράξης `a+b`.

**ifStat:** Καλεί την `genquad("jump "," _"," _"," _")` στο σημείο που κλείνει η παρένθεση του `if`. (Κανονικά θα έπρεπε να καλεί την `makelist` και την `backpatch` ώστε να αλλάζει το `z` και να κάνει `jump` στην εντολή που θέλουμε).

**whileStat:** Αρχικά, έχουμε 2 περιπτώσεις για το εσωτερικό της `while`, **1)** αν συναντήσει κάποια συνάρτηση, τότε αν η συνάρτηση έχει 1 όρισμα, όπως `πχ.([ while(area(in a) >= num)]`, θα καλέσει την `genquad("par ",Helpa," CV"," _")`, αν η `Helpa` μεταδίδεται με τιμή, ή την `genquad("par ",Helpa," REF"," _")`, αν η `Helpa` μεταδίδεται με αναφορά. Αν η συνάρτηση έχει 2 ορίσματα, όπως `πχ. while(area(in a,inout b) >= num)`, τότε, για το πρώτο όρισμα θα καλέσει την `genquad("par ",Helpb," CV"," _")` ή την `genquad("par ",Helpb," REF"," _")`, αν το `Helb` μεταδίδεται αντίστοιχα με τιμή ή με αναφορά, και για το δεύτερο θα καλέσει την `genquad("par ",Helpb," CV"," _")` ή `genquad("par ",Helpb," REF"," _")` αν το `Helpb` μεταδίδεται αντίστοιχα με τιμή ή με αναφορά. **2)** Αν συναντήσει κάποια συνάρτηση, τότε αν η συνάρτηση έχει 1 όρισμα, όπως `πχ.([ while(area(in a+b) >= num)]`, θα καλέσει την `genquad("par ",w1," CV"," _")`, αν η `w1` (αποτέλεσμα `a+b` στο παράδειγμά μας) μεταδίδεται με

τιμή , ή την `genquad("par ",w1," REF"," _")`,αν η `w1`(αποτέλεσμα `a+b` στο παράδειγμά μας) μεταδίδεται με αναφορά. Αν η συνάρτηση έχει 2 ορίσματα, όπως πχ. `while(area(in i+a,inout b) >= num)`, τότε , για το πρώτο όρισμα θα καλέσει την `genquad("par ",w1," CV"," _")` ή την `genquad("par ",w1," REF"," _")`,αν το `w1`(αποτέλεσμα `a+b` στο παράδειγμα μας) μεταδίδεται αντίστοιχα με τιμή ή με αναφορά, και για το δεύτερο θα καλέσει την `genquad("par ",w1," CV"," _")` ή `genquad("par ",w1," REF"," _")` αν `w1`(αποτέλεσμα `a+b` στο παράδειγμα μας) μεταδίδεται αντίστοιχα με τιμή ή με αναφορά. Στη συνέχεια θα καλέσει την `genquad("par ",w," RET"," _")` όπου `w` είναι η τιμή πλέον που επιστρέφει η συνάρτηση όταν κληθεί και μετά καλείται η `genquad("call ", fd," _"," _")` ,όπου `fd` το όνομα της συνάρτησης. Τώρα, έχουμε 5 περιπτώσεις σχετικά με το σύμβολο σύγκρισης **α)** καλεί την `genquad(jpeg+" ",w+" ",opr," _")` αν το σύμβολο είναι «=»,**β)** καλεί την `genquad(jpeg+" ",w+" ",opr1," _")` αν το σύμβολο είναι «>»,**γ)** καλεί την `genquad(jpeg+" ",w+" ",opr2," _")` αν το σύμβολο είναι «<»,**δ)** καλεί την `genquad(jpeg+" ",w+" ",opr3," _")` αν το σύμβολο είναι «>=»,**ε)** `genquad(jpeg+" ",w+" ",opr4," _")` αν το σύμβολο είναι «<=»,**στ)** `genquad(opr5+" ",w+" ",opr5," _")` αν το σύμβολο είναι «<>»,όπου σε κάθε περίπτωση ισχύει ,`jpeg`=σύμβολο σύγκρισης,`w`= τιμή που επιστρέφει η συνάρτηση και `opr(0-5)` το 2<sup>ο</sup> μέλος της σύγκρισης. Τέλος ,μόλις κλείσει η παρένθεση από τη `while` καλείται η `genquad("jump "," _"," _"," _")`((Κανονικά θα έπρεπε να καλεί την `backpatch` ώστε να αλλάζει το `z` και να κάνει `jump` στην εντολή που θέλουμε).)

**switchcaseStat:** Μόλις επιστραφεί το token <> τότε καλεί την `genquad("jump"," _"," _"," _")` .

**forcaseStat:** Μόλις επιστραφεί το token <> τότε καλεί την `genquad("jump"," _"," _"," _")` .

**incaseStat:** Μόλις επιστραφεί το token <> τότε καλεί την `genquad("jump"," _"," _"," _")` .

**returnStat:** Αρχικά, αν μέσα στο return συναντήσει κάποια συνάρτηση, όπως πχ. ([return(square(in c))]) ,τότε αν η συνάρτηση έχει 1 όρισμα θα καλέσει την genquad("par ",Helpa," CV"," \_"), αν η Helpa μεταδίδεται με τιμή , ή την genquad("par ",Helpa," REF"," \_"),αν η Helpa μεταδίδεται με αναφορά. Αν η συνάρτηση έχει 2 ορίσματα,όπως πχ. [return(area(in a,inout b))], τότε , για το πρώτο όρισμα θα καλέσει την genquad("par ",Helpa," CV"," \_") ή την genquad("par ",Helpa," REF"," \_") ,αν το Helpa μεταδίδεται αντίστοιχα με τιμή ή με αναφορά, και για το δεύτερο θα καλέσει την genquad("par ",Helpa," CV"," \_") ή την genquad("par ",Helpa," REF"," \_") αν το Helpa μεταδίδεται αντίστοιχα με τιμή ή με αναφορά. Στη συνέχεια θα καλέσει την genquad("par ",w," RET"," \_") όπου w είναι η τιμή πλέον που επιστρέφει η συνάρτηση όταν κληθεί και μετά καλείται η genquad("call "," \_ "," \_",fd),όπου fd το όνομα της συνάρτησης. Τώρα, έχουμε 3 περιπτώσεις σχετικά με την κλήση της genquad : **1)** κλήση συνάρτησης μέσα στο return,καλεί την genquad("retv "," \_ "," \_",w) ,όπου w είναι αυτό που επιστρέφει η συνάρτηση.**2)** πράξη μέσα στο return,όπως πχ.([return(x\*x)]),τότε καλεί την genquad("retv ",w1," \_ "," \_"),όπου w1 είναι το αποτέλεσμα της πράξης.**3)** όταν επιστρέφει απλά μια τιμή ,όπως πχ.([return(x)]),τότε καλεί την genquad("retv ",Eplace," \_ "," \_"),όπου Eplace η μεταβλητή.(Οι κλήσεις των genquad("out "," \_ "," \_",Eplace) είναι λαθος ,δεν έπρεπε να υπάρχουν).

**callStat:** Αρχικά, αν η call καλεί συνάρτηση που έχει 1 όρισμα([call area(in x)]) θα καλέσει την genquad("par ",CallList[0]," CV"," \_"), αν η CallList[0] μεταδίδεται με τιμή , ή την genquad("par ",CallList[0]," REF"," \_"),αν η CallList[0] μεταδίδεται με αναφορά. Αν η συνάρτηση έχει 2 ορίσματα ,όπως πχ. [call (area(in a,inout b))], τότε , για το πρώτο όρισμα θα καλέσει την genquad("par ",CallList[0]," CV"," \_") ή την genquad("par ",CallList[0]," REF"," \_") ,αν το CallList[0] μεταδίδεται αντίστοιχα με τιμή ή με αναφορά, και για το δεύτερο θα καλέσει την genquad("par ",CallList[1]," CV"," \_") ή την genquad("par ",CallList[1]," REF"," \_") αν το CallList[1] μεταδίδεται αντίστοιχα με τιμή ή με αναφορά. Στη συνέχεια θα καλέσει την genquad("call ",ap," \_ "," \_") όπου ap είναι το όνομα της συνάρτησης όταν κληθεί .

**printStat:** Αρχικά, αν η print καλεί συνάρτηση που έχει 1 όρισμα([print(area(in x))]) θα καλέσει την genquad("par ",Helpr," CV"," \_"), αν η Helpr μεταδίδεται με τιμή, ή την genquad("par ",Helpr," REF"," \_"),αν η Helpr μεταδίδεται με αναφορά. Αν η συνάρτηση έχει 2 ορίσματα, όπως πχ. [print(area(in a,inout b))], τότε, για το πρώτο όρισμα θα καλέσει την genquad("par ",Helpr," CV"," \_") ή την genquad("par ",Helpr," REF"," \_"),αν το Helpr μεταδίδεται αντίστοιχα με τιμή ή με αναφορά, και για το δεύτερο θα καλέσει την genquad("par ",Helprb," CV"," \_") ή την genquad("par ",Helprb," REF"," \_") αν το Helprb μεταδίδεται αντίστοιχα με τιμή ή με αναφορά. Στη συνέχεια θα καλέσει τις genquad("par ",w," RET"," \_") και genquad("call ",fd," \_"," \_"),όπου w είναι αυτό που επιστρέφει η συνάρτηση που καλείται και το fd το όνομα της συνάρτησης. Μετά έχουμε 3 περιπτώσεις.**1)**Αν μέσα στην print υπάρχει κλήση συνάρτησης καλεί την genquad("out "," \_ "," \_ ",w), όπου w αυτό που επιστρέφει η συνάρτηση όταν κληθεί.**2)**Αν μέσα στην print υπάρχει πράξη([print(x+a)]) καλεί την genquad("out "," \_ "," \_ ",w1),όπου w1 το αποτέλεσμα της πράξης.**3)** Αν μέσα στην print υπάρχει απλά μια μεταβλητή καλεί την genquad("out "," \_ "," \_ ",Eplace),όπου Eplace το όνομα της μεταβλητής.

**inputStat:** Καλεί την genquad("inp ",idplace," \_ "," \_ ") ,όπου idplace είναι η μεταβλητή που αποθηκεύεται το input που δίνουμε, όπως πχ.[input(x)],idplace=x.

**boolterm:** Μόλις διαβάσει το token <and> και πάει στο επόμενο καλεί την genquad("jump "," \_ "," \_ "," \_ ").

**expression:** Στην πρόσθεση, όπως πχ.[t:=a+b], καλεί την genquad("+",T1place+" ",T2place+" ",w1), όπου T1place είναι στο παράδειγμά μας το a, το T2place το b, και το w1 το αποτέλεσμα τους. Στην αφαίρεση όπως πχ.[t:=a-b], καλεί την genquad("-",T1place+" ",T2place+" ",w1),όπου



T1place είναι στο παράδειγμά μας το a , το T2place το b, και το w1 το αποτέλεσμα τους.

**term:** Στον πολλαπλασιασμό, όπως πχ.[t:=a\*b], καλεί την genquad("\*",F1place+" ",F2place+" ",w1) ,όπου F1place είναι στο παράδειγμά μας το a , το F2place το b, και το w1 το αποτέλεσμα τους. Στην διαίρεση όπως πχ.[t:=a/b], καλεί την genquad("/ ",F1place+" ",F2place+" ",w1),όπου F1place είναι στο παράδειγμά μας το a , το F2place το b, και το w1 το αποτέλεσμα τους.

**REL\_OP:** Στη συνάρτηση αυτή έχουμε περιπτώσεις σχετικά με το σύμβολο σύγκρισης.

Σε κάθε μια περίπτωση υπάρχει ένα block που ελέγχει ,1<sup>ov</sup> αν στο 1<sup>o</sup> μέρος της σύγκρισης υπάρχει πράξη για να πάρει το αποτέλεσμα της πράξης ,όπως πχ.(a+b>=c), αυτό το κάνει με την μεταβλητή bo, και 2<sup>ov</sup> δεν επιτρέπει με τη μεταβλητή gr να γίνεται κλήση στην genquad όταν υπάρχει σύγκριση μέσα σε κλήση συνάρτησης. Σε κάθε μια περίπτωση καλούμε την genquad("= ",rtx+" ",opr(0-5)+" ","\_ ") ,αν δεν προϋπάρχει πράξη(πχ. a=b ,rtx =a ,opr(0-5)=b), και genquad("= ",w1+" ",opr(0-5)+" ","\_ ") ,αν προϋπάρχει πράξη (πχ. a+b=c,w1=a+b,opr(0-5)=c).

Όλα τα παραπάνω καταγράφονται σε μια λίστα και εκτυπώνονται στο αρχείο test.int

Παρακάτω παραθέτουμε ένα παράδειγμα με τη λειτουργία του ενδιάμεσου κώδικα με την .int μορφή του

test.int

```

1 1: begin_block square _ _
2 2: * x x T_1
3 3: retv T_1 _ _
4 4: out _ _ x
5 5: end_block square _ _
6 6: begin_block area _ _
7 7: * a b T_2
8 8: = T_2 0 _
9 9: jump _ _ _
10 10: + c a T_3
11 11: >= T_3 0 _
12 12: jump _ _ _
13 13: * a b T_4
14 14: := ans _ T_4
15 15: retv ans _ _
16 16: out _ _ ans
17 17: + b a T_5
18 18: >= T_5 0 _
19 19: jump _ _ _
20 20: * a square T_6
21 21: := ans _ T_6
22 22: retv ans _ _
23 23: out _ _ ans
24 24: = c 0 _
25 25: jump _ _ _
26 26: par c CV _ _
27 27: par T_7 RET _
28 28: call _ _ square
29 29: retv _ _ T_7
30 30: out _ _ square
31 31: + a b T_8
32 32: := t _ T_8
33 33: retv ans _ _
34 34: out _ _ ans
35 35: + i a T_9
36 36: par T_9 CV _
37 37: par T_9 REF _
38 38: par T_10 RET _
39 39: call area _ _
40 40: >= T_10 num _
41 41: jump _ _ _
42 42: + i 1 T_11
43 43: := i _ T_11
44 44: end_block area _ _
45 45: begin_block area _ _
46 46: inp x1 _ _
47 47: inp x2 _ _
48 48: inp x3 _ _

```

area1.ci

```

1 program area
2 declare x1,x2,x3;
3 declare y1,y2,y3;
4 declare a,b,c,answer;
5 function square (in x)
6 {
7     return (x*x);
8 }
9 function area(in a,inout b)
10 declare ans,t;
11 {
12     if(a*b = 0 and c + a >= 0) {
13         ans:=a*b;
14         return (ans);
15     };
16     if(b + a >= 0 ){
17         ans:=a*square(in c);
18         return(ans);
19     };
20     if(c=0){
21         return(square(in c));
22     }
23     else{
24         t:=a+b;
25         return (ans);
26     };
27     while(area(in i+a,inout b) >= num){
28         i:=i+1;
29     };
30 }
31 #main function#
32 {
33     input(x1);
34     input(x2);
35     input(x3);
36     input(y1);
37     input(y2);
38     input(y3);
39     a:=x2/x1*x0;
40     b:=y2*y1;
41     c:=y3-x3;
42     answer:= area(in a,inout b);
43     print(answer);
44     return(area(in h,inout g));
45     answer:= area(in a,inout b);
46     print(area(in a,inout b));
47     print(a);
48     call area(in a,inout b);
49 }.

```

test.int

```
41 41: jump _ _
42 42: + i 1 T_11
43 43: := i _ T_11
44 44: end_block area _ _
45 45: begin_block area _ _
46 46: inp x1 _ _
47 47: inp x2 _ _
48 48: inp x3 _ _
49 49: inp y1 _ _
50 50: inp y2 _ _
51 51: inp y3 _ _
52 52: / x2 x1 T_12
53 53: * T_12 x0 T_13
54 54: := a _ T_13
55 55: * y2 y1 T_14
56 56: := b _ T_14
57 57: -y3 x3 T_15
58 58: := c _ T_15
59 59: par i CV _
60 60: par b REF _
61 61: par T_16 RET _
62 62: call area _ _
63 63: := answer _ T_16
64 64: out _ _ answer
65 65: par a CV _
66 66: par a REF _
67 67: par T_17 RET _
68 68: call _ _ area
69 69: retv _ _ T_17
70 70: out _ _ area
71 71: par h CV _
72 72: par g REF _
73 73: par T_18 RET _
74 74: call area _ _
75 75: := answer _ T_18
76 76: par a CV _
77 77: par b REF _
78 78: par T_19 RET _
79 79: call area _ _
80 80: out _ _ T_19
81 81: out _ _ a
82 82: par a CV _
83 83: par b REF _
84 84: call area _ _
85 85: halt _ _
86 86: end_block area _ _
87
```

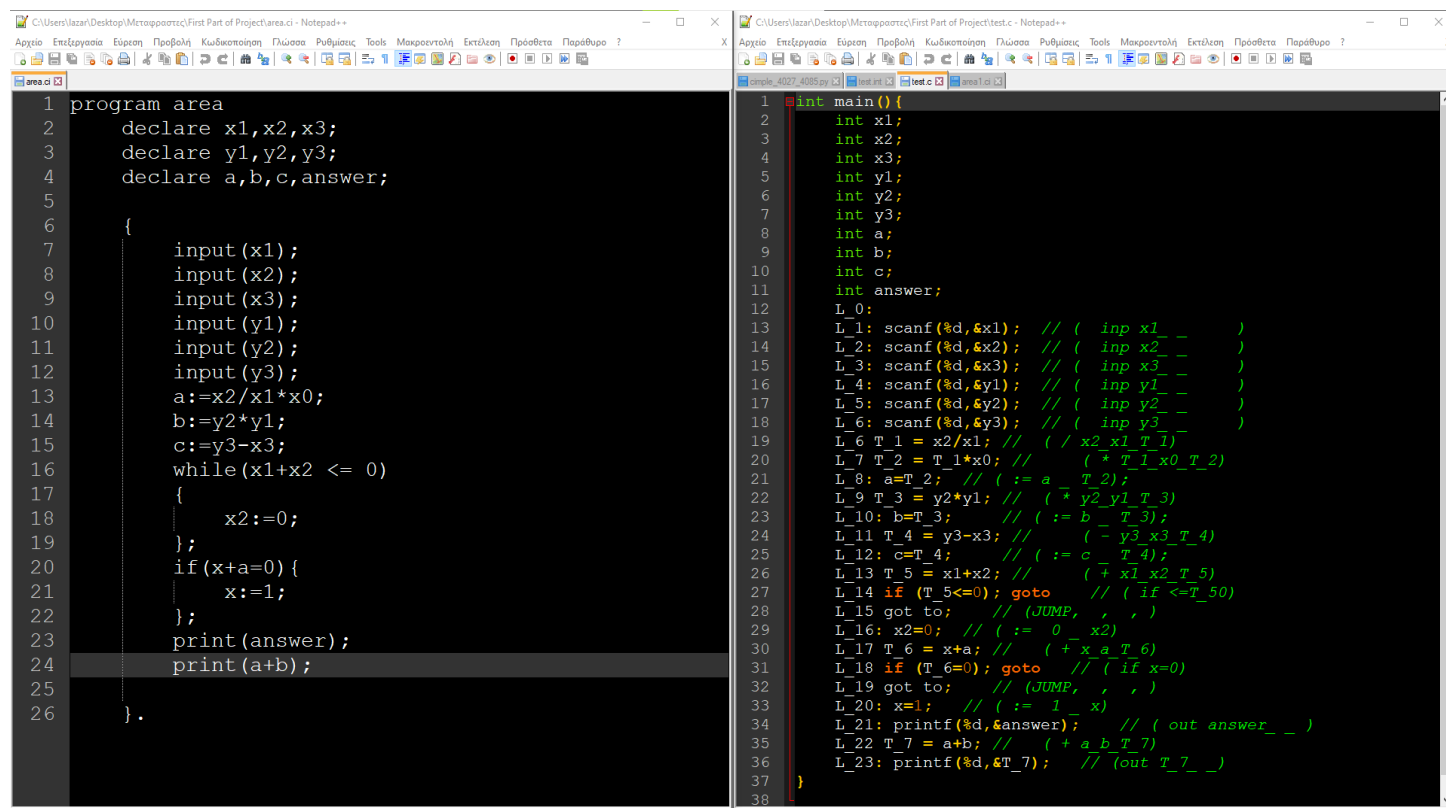
Επίσης από τον ενδιάμεσο κώδικα μπορούμε να παράξουμε και το ισοδύναμο σε C. Από κάθε τετράδα του ενδιάμεσου κώδικα παράγουμε μία γραμμή γλώσσας C η οποία κάνει την ίδια δουλειά με την τετράδα. Σκοπός είναι με έναν compiler της C να μπορέσουμε να τρέξουμε το πρόγραμμα που φτιάξαμε ώστε να μπορούμε να ελέγξουμε με σιγουριά την ορθή λειτουργία του. Στο ισοδύναμο C δεν μας ενδιαφέρει να ελέγξουμε την ορθή μετατροπή κώδικα της αρχικής γλώσσας που περιέχει συναρτήσεις. Αν ο κώδικας της αρχικής γλώσσας περιέχει συναρτήσεις, τότε το ισοδύναμο C δεν το παράγουμε καθόλου

Αρχικά έχουμε τοποθετήσει ένα index ,το WriteOnC, το οποίο αν έχει την τιμή 1 τότε μπορεί να γράψει το παράγων αρχείο C. Αν υπάρχουν συναρτήσεις θα πάρει την τιμή 0 και δεν θα παραχθεί. Παρακάτω παραθέτουμε τις συναρτήσεις που σχετίζονται με την παραγωγή του αρχείου C. Στις παρακάτω συναρτήσεις γίνεται η προσθήκη των εγγραφών που χρειάζονται για τη δημιουργία του αρχείου C.

**A)program, B)block, C)assingStat, D)ifStat, E)ifStat, F)whileStat, G)printStat, H)inputStat, I)expression, J)term.**

**Όλα τα παραπάνω περιέχονται σε μια λίστα ,την CList, και εκτυπώνονται στο αρχείο test.c**

## Παρακάτω φαίνεται η λειτουργία του αρχείου C στο πρόγραμμα μας area.ci



```
1 program area
2   declare x1,x2,x3;
3   declare y1,y2,y3;
4   declare a,b,c,answer;
5
6   {
7       input(x1);
8       input(x2);
9       input(x3);
10      input(y1);
11      input(y2);
12      input(y3);
13      a:=x2/x1*x0;
14      b:=y2*y1;
15      c:=y3-x3;
16      while(x1+x2 <= 0)
17      {
18          x2:=0;
19      };
20      if(x+a=0){
21          x:=1;
22      };
23      print(answer);
24      print(a+b);
25
26  }.
```

```
1 int main(){
2     int x1;
3     int x2;
4     int x3;
5     int y1;
6     int y2;
7     int y3;
8     int a;
9     int b;
10    int c;
11    int answer;
12    L_0:
13    L_1: scanf("%d",&x1); // ( inp x1 _ )
14    L_2: scanf("%d",&x2); // ( inp x2 _ )
15    L_3: scanf("%d",&x3); // ( inp x3 _ )
16    L_4: scanf("%d",&y1); // ( inp y1 _ )
17    L_5: scanf("%d",&y2); // ( inp y2 _ )
18    L_6: scanf("%d",&y3); // ( inp y3 _ )
19    L_7 T_1 = x2/x1; // ( / x2 x1 T_1 )
20    L_8 T_2 = T_1*x0; // ( * T_1 x0 T_2 )
21    L_9: a=T_2; // ( := a T_2 )
22    L_10 T_3 = y2*y1; // ( * y2 y1 T_3 )
23    L_11: b=T_3; // ( := b T_3 )
24    L_12 T_4 = y3-x3; // ( - y3 x3 T_4 )
25    L_13: c=T_4; // ( := c T_4 )
26    L_14 T_5 = x1+x2; // ( + x1 x2 T_5 )
27    L_15 if (T_5<=0); goto // ( if <=T_50 )
28    L_16 got to; // ( JUMP, , , )
29    L_17: x2=0; // ( := 0 x2 )
30    L_18 T_6 = x+a; // ( + x a T_6 )
31    L_19 if (T_6=0); goto // ( if x=0 )
32    L_20 got to; // ( JUMP, , , )
33    L_21: x=1; // ( := 1 x )
34    L_22 T_7 = a+b; // ( + a b T_7 )
35    L_23: printf("%d",&answer); // ( out answer _ )
36    L_24 T_7 = a+b; // ( + a b T_7 )
37    L_25: printf("%d",&T_7); // (out T_7 _ )
38 }
```

### 3) Πίνακας Συμβόλων:

Ο πίνακας συμβόλων είναι στην ουσία μια λίστα που κρατάει τις εξής πληροφορίες:

- Μεταβλητή string name int type int offset (απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης)
- Συνάρτηση string name int type int startQuad (ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης) list argument (λίστα παραμέτρων) int framelength (μήκος εγγραφήματος δραστηριοποίησης)
- Σταθερά string name string value (τιμή της σταθεράς)
- Παράμετρος string name int parMode (τρόπος περάσματος) int offset (απόσταση από την κορυφή της στοίβας)

- Προσωρινή μεταβλητή string name int offset (απόσταση από την κορυφή της στοίβας)

Για να δημιουργηθεί ο πίνακας συμβόλων θα πρέπει να τηρούνται οι εξής προϋποθέσεις :

- α) κάθε συνάρτηση έχει μέσα της τουλάχιστον ένα return
- β) δεν υπάρχει return έξω από συνάρτηση
- γ) μέσα σε μία διαδικασία δεν επιτρέπεται να υπάρχει return

Εμείς λοιπόν χρησιμοποιήσαμε ένα index ,το

WriteOnArrayOfSymbols ,όπου αν αυτό ισούται με 1, τότε μπορεί να γράψει το περιεχόμενο της λίστας ArrayOfSyms(που περιέχει τις πληροφορίες του πίνακα συμβόλων) στο αρχείο ArrayOfSymbols.txt.

Οι συναρτήσεις που χρησιμοποιήσαμε για να καταγράψουμε τις πληροφορίες είναι οι εξής : declarations, varlist, expression και term.

Παρακάτω παραθέτουμε ένα παράδειγμα ημιτελούς πίνακα συμβόλων.

```

1 ['x1', 'variable', 12]
2 ['x2', 'variable', 16]
3 ['x3', 'variable', 20]
4 ['y1', 'variable', 24]
5 ['y2', 'variable', 16]
6 ['y3', 'variable', 20]
7 ['a', 'variable', 24]
8 ['b', 'variable', 16]
9 ['c', 'variable', 20]
10 ['answer', 'variable', 24]
11 ['T_1', 'temporary variable', 12]
12 ['ans', 'variable', 12]
13 ['t', 'variable', 16]
14 ['T_2', 'temporary variable', 16]
15 ['T_3', 'temporary variable', 20]
16 ['T_4', 'temporary variable', 24]
17 ['T_5', 'temporary variable', 28]
18 ['T_6', 'temporary variable', 32]
19 ['T_8', 'temporary variable', 36]
20 ['T_9', 'temporary variable', 40]
21 ['T_11', 'temporary variable', 44]
22 ['T_12', 'temporary variable', 48]
23 ['T_13', 'temporary variable', 52]
24 ['T_14', 'temporary variable', 56]
25 ['T_15', 'temporary variable', 60]
26

1 program area
2 declare x1,x2,x3;
3 declare y1,y2,y3;
4 declare a,b,c,answer;
5 function square(in x)
6 {
7     return (x*x);
8 }
9
10 function area(in a,inout b)
11 declare ans,t;
12 {
13     if(a*b = 0 and c + a >= 0) {
14         ans=a*b;
15         return (ans);
16     }
17     if(b + a >= 0){
18         ans=a*square(in c);
19         return(ans);
20     }
21     if(c=0){
22         return(square(in c));
23     }
24     else{
25         t:=a*b;
26         return (ans);
27     }
28     while(area(in i+a,inout b) >= num){
29         i:=i+1;
30     }
31 }
32 #main function#
33 {
34     input(x1);
35     input(x2);
36     input(x3);
37     input(y1);
38     input(y2);
39     input(y3);
40     a:=x2/x1*x0;
41     b:=y2*y1;
42     c:=y3-x3;
43     answer:= area(in a,inout b);
44     print(answer);
45
46     answer:= area(in a,inout b);
47     print(area(in a,inout b));
48     print(a);
49     call area(in a,inout b);
50 }

```

\*Σημειωση. Για την καλύτερη ανάγνωση του screenshot παρακαλούμε να κάνετε λίγο μεγέθυνση.

\*Σημειωση. Δεν έχουμε κάνει σωστά τα offset των temporary variables.

## 4) Παραγωγή Τελικού Κώδικα:

Η παραγωγή τελικού κώδικα αποτελεί την τελευταία φάση της μετάφρασης. Για την παραγωγή του τελικού κώδικα θα πρέπει η κάθε εντολή του τελικού κώδικα να παράγεται από τις αντίστοιχες του ενδιάμεσου κώδικα, ενώ παράλληλα οι μεταβλητές να απεικονίζονται στην μνήμη και το πέρασμα παραμέτρων αλλά και η κλήση συναρτήσεων. Για το τέλος δημιουργούμε σε αυτή την φάση κώδικα για τον επεξεργαστή MIPS.

### Καταχωρητές που θα μας φανούν χρήσιμοι:

- καταχωρητές προσωρινών τιμών: \$t0...\$t7
- καταχωρητές οι τιμές των οποίων διατηρούνται ανάμεσα σε κλήσεις συναρτήσεων: \$s0...\$s7
- καταχωρητές ορισμάτων: \$a0...\$a3
- καταχωρητές τιμών: \$v0,\$v1
- stack pointer \$sp
- frame pointer \$fp
- return address \$ra

### Εντολές που θα μας φανούν χρήσιμες για αριθμητικές πράξεις:

- add \$t0,\$t1,\$t2  $t0=t1+t2$
- sub \$t0,\$t1,\$t2  $t0=t1-t2$
- mul \$t0,\$t1,\$t2  $t0=t1*t2$
- div \$t0,\$t1,\$t2  $t0=t1/t2$

### Εντολές που θα μας φανούν χρήσιμες για μετακίνηση δεδομένων:

- move \$t0,\$t1  $t0=t1$  μεταφορά ανάμεσα σε καταχωρητές
- li \$t0, value  $t0=$ value σταθερά σε καταχωρητή
- lw \$t1,mem  $t1=[$ mem] περιεχόμενο μνήμης σε καταχωρητή
- sw \$t1,mem  $[$ mem $]=t1$  περιεχόμενο καταχωρητή σε μνήμη
- lw \$t1,(\$t0)  $t1=[t0]$  έμμεση αναφορά με καταχωρητή
- sw \$t1,-4(\$sp)  $t1=[sp-4]$  έμμεση αναφορά με βάση τον \$sp

### Εντολές που θα μας φανούν χρήσιμες για άλματα:

- b label branch to label
- beq \$t1,\$t2,label jump to label if  $t1=t2$
- blt \$t1,\$t2,label jump to label if  $t1<t2$
- bgt \$t1,\$t2,label jump to label if  $t1>t2$
- ble \$t1,\$t2,label jump to label if  $t1\leq t2$
- bge \$t1,\$t2,label jump to label if  $t1\geq t2$
- bne \$t1,\$t2,label jump to label if  $t1\neq t2$

## Εντολές που θα μας φανούν χρήσιμες στην κλήση συναρτήσεων:

- j label jump to label
- jal label κλήση συνάρτησης
- jr \$ra άλμα στη διεύθυνση που έχει ο καταχωρητής στο παράδειγμα είναι ο \$ra που έχει την διεύθυνση επιστροφής συνάρτησης

## loadvr

- αν v είναι σταθερά li \$tr,v
- αν v είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα lw \$tr,-offset(\$s0)

## Storerv

- αν v είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα sw \$tr,-offset(\$s0)
- αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή sw \$tr,-offset(\$sp)
- αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον lw \$t0,-offset(\$sp), sw \$tr,(\$t0)
- αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον gnlvcode(v), sw \$tr,(\$t0)
- αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον gnlvcode(v) ,lw \$t0,(\$t0) ,sw \$tr,(\$t0)

## εκχώρηση

- :=, x, “\_”, z  
    loadvr(x, \$t1)  
    storerv(\$t1, z)

## εντολές αριθμητικών πράξεων

- op x,y,z  
    loadvr(x, \$t1)  
    loadvr(y, \$t2)  
    op \$t1,\$t1,\$t2 op: add,sub,mul,div  
    storerv(\$t1,z)



## Εντολές Εισόδου-Εξόδου

- out “\_”, “\_”, x  
li \$v0,1  
loadvr(x,\$a0)  
syscall
- in “\_”, “\_”, x  
li \$v0,5  
syscall  
storerv(\$v0,x)

## Επιστροφή Τιμής Συνάρτησης

- retv “\_”, “\_”, x  
loadvr(x, \$t1)  
lw \$t0,-8(\$sp)  
sw \$t1,(\$t0)

αποθηκεύεται ο x στη διεύθυνση που είναι αποθηκευμένη στην 3η θέση του εγγραφήματος δραστηριοποίησης

- εναλλακτικά μπορούμε να γράψουμε το αποτέλεσμα στον \$v0, και μετά πρέπει να φροντίσουμε να το πάρουμε από εκεί

- loadvr(x, \$t1)  
move \$v0,\$t1

## Παράμετροι Συνάρτησης

- par,x,CV, \_  
loadvr(x, \$t0)  
sw \$t0, -(12+4i)(\$fp)  
όπου i ο αύξων αριθμός της παραμέτρου

- par,x,REF, \_ ν αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή

- addi \$t0,\$sp,-offset  
sw \$t0,-(12+4i)(\$fp)

- par,x,REF, \_ ν αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά

- lw \$t0,-offset(\$sp)  
sw \$t0,-(12+4i)(\$fp)

- par,x,RET, \_ γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή

```
addi $t0,$sp,-offset
```

```
sw $t0,-8($fp)
```

## **Κλήση Συνάρτησης**

- αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα

```
lw $t0,-4($sp)
```

```
sw $t0,-4($fp)
```

- αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα

```
lw $t0,-4($sp)
```

```
sw $t0,-4($fp)
```

- αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας

```
sw $sp,-4($fp)
```

- στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα

```
addi $sp,$sp,framelength
```

- καλούμε τη συνάρτηση

```
jal f
```

- και όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα

```
addi $sp,$sp,-framelength
```

- μέσα στην κληθείσα

στην αρχή κάθε συνάρτησης αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον \$ra η jal

```
sw $ra,($sp)
```

- στην τέλος κάθε συνάρτησης κάνουμε το αντίστροφο, παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της συνάρτησης και την βάζουμε πάλι στον \$ra. Μέσω του \$ra επιστρέφουμε στην καλούσα

```
lw $ra,($sp)
```

```
jr $ra
```

## Αρχή Προγράμματος και Κυρίως Πρόγραμμα

- το κυρίως πρόγραμμα δεν είναι το πρώτο πράγμα που μεταφράζεται, οπότε στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος  
j Lmain
- φυσικά η j Lmain πρέπει να δημιουργηθεί όταν ξεκινά η μετάφραση της main
- στη συνέχεια πρέπει να κατεβάσουμε τον \$sp κατά framelength της main  
addi \$sp,\$sp,framelength
- και να σημειώσουμε στον \$s0 το εγγράφημα δραστηριοποίησης της main ώστε να έχουμε εύκολη πρόσβαση στις global μεταβλητές  
move \$s0,\$sp

Οι συναρτήσεις που χρησιμοποιήθηκαν για την συμπλήρωση της λίστας(FinalCodeList),που περιλαμβάνει τον τελικό κώδικα, είναι οι εξής: program,block,subprogram,assingStat,whileStat,ifStat,switchcaseStat, forcaseStat, returnStat,callStat,printStats,boolterm,expression καιterm,

## Παρακάτω παραθέτουμε ένα παράδειγμα του τελικού κώδικα όταν δίνουμε το area1.ci στον compiler

```
1 L0:      b Lmain:
2 L1:      sw $ra,-0($sp)
3          mul $t1,$t1,t2
4          sw $t1,-12($sp)
5 L3:      lw $t1,-12($sp)
6          lw $t1,($t0)
7          lw $t0,-8($sp)
8          sw $t1,($t0)
9 L4:      lw $ra,-0($sp)
10         jr $ra
11 L5:      sw $ra,-0($sp)
12 L6:      lw $t1,-24($sp)
13         lw $t2,-16($sp)
14         mul $t1,$t1,t2
15         sw $t1,-16($sp)
16 L7:      b _
17 L8:      lw $t1,-20($sp)
18         lw $t2,-24($sp)
19         add $t1,$t1,t2
20         sw $t1,-20($sp)
21 L9:      b _
22 L10:     lw $t1,-24($sp)
23         lw $t2,-16($sp)
24         mul $t1,$t1,t2
25         sw $t1,-24($sp)
26 L11:     lw $t1,-offset($sp)
27         lw $t0,-24($sp)
28         sw $t1,($t0)
29 L12:     lw $t1,-12offset($sp)
30         lw $t1,($t0)
```

```
1 program area
2 declare x1,x2,x3;
3 declare y1,y2,y3;
4 declare a,b,c,answer;
5 function square(in x)
6 {
7     return (x*x);
8 }
9 function area(in a,inout b)
10 declare ans,t;
11 {
12     if(a*b = 0 and c + a >= 0) {
13         ans:=a*b;
14         return (ans);
15     };
16     if(b + a >= 0 ){
17         ans:=a*square(in c);
18         return(ans);
19     };
20     if(c=0){
21         return(square(in c));
22     }
23     else{
24         t:=a*b;
25         return (ans);
26     };
27     while(area(in i+a,inout b) >= num) {
28         i:=i+1;
29     };
30 }
31 #main function#
32 {
33     input(x1);
34     input(x2);
35     input(x3);
36     input(y1);
37     input(y2);
38     input(y3);
39     a:=x2/x1*x0;
40     b:=y2*y1;
41     c:=y3-x3;
42     answer:= area(in a,inout b);
43     print(answer);
44
45     answer:= area(in a,inout b);
46     print(area(in a,inout b));
47     print(a);
48     call area(in a,inout b);
49 }.
```

```

31      lw $t0,-8($sp)
32      sw $t1,($t0)
33 L13:   lw $t1,-12($sp)
34      li $v0,1
35      move $a0, $t1
36      syscall
37 L14:   lw $t1,-16($sp)
38      lw $t2,-24($sp)
39      add $t1,$t1,t2
40      sw $t1,-28($sp)
41 L15:   b _
42 L16:   lw $t1,-24($sp)
43      mul $t1,$t1,t2
44      sw $t1,-32($sp)
45 L17:   lw $t1,-offset($sp)
46      lw $t0,-32($sp)
47      sw $t1,($t0)
48 L18:   lw $t1,-12offset($sp)
49      lw $t1,($t0)
50      lw $t0,-8($sp)
51      sw $t1,($t0)
52 L19:   lw $t1,-12($sp)
53      li $v0,1
54      move $a0, $t1
55      syscall
56 L20:   b _
57 L21:   addi $fp,$sp, offset
58      lw $t1,-20($sp)
59      sw $t1,-20($fp)
60 L22:   addi $t0,$sp,-offset

```

```
cimble_4027_4085.py x test.int x test.c x test.asm x
61      sw $t0,-8($fp)
62 L23:  lw $t1,-32offset($sp)
63      li $v0,1
64      move $a0, $t1
65      syscall
66 L24:  lw $t1,-24($sp)
67      lw $t2,-16($sp)
68      add $t1,$t1,t2
69      sw $t1,-36($sp)
70 L25:  lw $t1,-offset($sp)
71      lw $t0,-36($sp)
72      sw $t1,($t0)
73 L26:  lw $t1,-12offset($sp)
74      lw $t1,($t0)
75      lw $t0,-8($sp)
76      sw $t1,($t0)
77 L27:  lw $t1,-12($sp)
78      li $v0,1
79      move $a0, $t1
80      syscall
81      lw $t2,-24($sp)
82      add $t1,$t1,t2
83      sw $t1,-40($sp)
84 L29:  addi $fp,$sp, offset
85      lw $t1,-40($sp)
86      sw $t1,-40($fp)
87 L30:  addi $t0,$sp,-offset
88      sw $t0,-8($fp)
89 L31:  b _
90      lw $t2,1
```

```
91      add $t1,$t1,t2
92      sw $t1,-44($sp)
93 L33:   lw $t1,-offset($sp)
94      lw $t0,-44($sp)
95      sw $t1,($t0)
96 L34:   lw $ra,-0($sp)
97      jr $ra
98 Lmain:
99 L35:   addi $sp, $sp, framelength
100      move $s0, $sp
101 L36:   li $v0, 5
102      syscall
103      sw $v0,-offset($sp)
104 L37:   li $v0, 5
105      syscall
106      sw $v0,-offset($sp)
107 L38:   li $v0, 5
108      syscall
109      sw $v0,-offset($sp)
110 L39:   li $v0, 5
111      syscall
112      sw $v0,-offset($sp)
113 L40:   li $v0, 5
114      syscall
115      sw $v0,-offset($sp)
116 L41:   li $v0, 5
117      syscall
118      sw $v0,-offset($sp)
119      div $t1,$t1,t2
120      sw $t1,-48($sp)
```

```
cimple_4027_4085.py x test.int x test.c x test.asm x
121      mul $t1,$t1,t2
122      sw $t1,-52($sp)
123 L44:   lw $t1,-offset($sp)
124      lw $t0,-52($sp)
125      sw $t1,($t0)
126      mul $t1,$t1,t2
127      sw $t1,-56($sp)
128 L46:   lw $t1,-offset($sp)
129      lw $t0,-56($sp)
130      sw $t1,($t0)
131      sub $t1,$t1,t2
132      sw $t1,-60($sp)
133 L48:   lw $t1,-offset($sp)
134      lw $t0,-60($sp)
135      sw $t1,($t0)
136 L49:   addi $t0,$sp,-offset
137      sw $t0,-8($fp)
138 L50:   li $v0,1
139      lw $a0,-offset($s0)
140      syscall
141 L51:   addi $fp,$sp, offset
142      lw $t1,-24($sp)
143      sw $t1,-24($fp)
144 L52:   addi $t0,$sp,-offset
145      sw $t0,-8($fp)
146 L53:   addi $fp,$sp, offset
147      lw $t1,-24($sp)
148      sw $t1,-24($fp)
149 L54:   li $v0,1
150      lw $a0,-offset($s0)
```



```

151      syscall
152 L55:   li $v0,1
153      lw $a0,-offset($s0)
154      syscall
155 L56:   addi $fp,$sp, offset
156      lw $t1,-24($sp)
157      sw $t1,-24($fp)
158 L57:
159 L58:
160

```

Σημείωση Τα περισσότερα offset δεν είναι σωστά και σε αρκετά σημεία προτιμήσαμε να βάλουμε τη λέξη offset από το να βάλουμε έναν τυχαίο αριθμό τύπου  $12+4*i$ . Επίσης δεν έχουμε φτιάξει τα if και αυτά με το βάθος φωλιάσματος.

Ελπίζουμε να μη κουράσαμε τον αναγνώστη και η αναφορά να φάνηκε κατατοπιστική για την κατανόηση της εργασίας μας και να μπορεί να είναι κατανοητή σε μελλοντική χρήση!