

## **ΜΥΥ802 – ΜΕΤΑΦΡΑΣΤΕΣ**

“Προγραμματιστική άσκηση: Η γλώσσα προγραμματισμού EEL”

Γκαβαρδίνας Όθωνας, ΑΜ: 2620

Μπουρλή Στυλιανή, ΑΜ: 2774

## ΠΕΡΙΕΧΟΜΕΝΑ

<b>1. ΕΙΣΑΓΩΓΗ</b>	<b>4</b>
1.1 Ο ορισμός του μεταγλωττιστή. ....	4
1.2 Οι απαιτήσεις από ένα μεταγλωττιστή. ....	4
1.3 Οι φάσεις της μεταγλώττισης. ....	5
<b>2. Η ΓΛΩΣΣΑ EEL</b>	<b>6</b>
2.1 Τα χαρακτηριστικά της γλώσσας EEL. ....	6
2.2 Η γραμματική της γλώσσας EEL. ....	13
<b>3. ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ</b>	<b>15</b>
3.1 Ο ορισμός και η λειτουργία του λεκτικού αναλυτή. ....	15
3.2 Πίνακας καταστάσεων για τη γλώσσα EEL. ....	16
3.3 Αυτόματο καταστάσεων για τη γλώσσα EEL. ....	17
3.4 Επεξήγηση του κώδικα που αφορά το λεκτικό αναλυτή. ....	18
<b>4. ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ</b>	<b>21</b>
4.1 Ο ορισμός και η λειτουργία του συντακτικού αναλυτή. ....	21
4.4 Επεξήγηση του κώδικα που αφορά το συντακτικό αναλυτή. ...	22
<b>5. ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ</b>	<b>27</b>
5.1 Ο ορισμός του ενδιάμεσου κώδικα. ....	27
5.2 Τελεστές – Αρχή και τέλος ενότητας – Συναρτήσεις – Διαδικασίες στον ενδιάμεσο κώδικα. ....	28
5.3 Βοηθητικές υπορουτίνες για τον ενδιάμεσο κώδικα. ....	30
5.4 Επεξήγηση του κώδικα που αφορά τον ενδιάμεσο κώδικα. ....	31
5.5 Αποθήκευση ενδιάμεσου κώδικα σε αρχείο. ....	38
5.6 Μετατροπή ενδιάμεσου κώδικα σε γλώσσα C και αποθήκευση σε αρχείο. ....	39
<b>6. ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ</b>	<b>40</b>
6.1 Ο ορισμός του πίνακα συμβόλων. ....	40
6.2 Εγγράφημα Δραστηριοποίησης. ....	41
6.3 Βοηθητικές υπορουτίνες για τον πίνακα συμβόλων. ....	42
6.4 Επεξήγηση του κώδικα που αφορά τον πίνακα συμβόλων. ....	44
<b>7. ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΛΥΣΗ</b>	<b>46</b>
7.1 Απαιτήσεις που αφορούν τη σημασιολογική ανάλυση. ....	46
7.2 Βοηθητικές υπορουτίνες για τη σημασιολογική ανάλυση. ....	46
7.3 Επεξήγηση του κώδικα που αφορά τη σημασιολογική ανάλυση.	47
<b>8. ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ</b>	<b>51</b>
8.1 Βασικά χαρακτηριστικά τελικού κώδικα. ....	51

8.2 Αρχιτεκτονική MIPS.....	51
8.3 Βοηθητικές συναρτήσεις για τον τελικό κώδικα. ....	53
8.4 Επεξήγηση του κώδικα που αφορά την παραγωγή τελικού κώδικα. ....	55
<b>9. ΤΕΣΤ</b>	<b>60</b>
9.1 ΤΕΣΤ1.....	60
9.2 ΤΕΣΤ2.....	62

# 1. ΕΙΣΑΓΩΓΗ

## 1.1 Ο ορισμός του μεταγλωττιστή

Μεταγλωττιστής ή μεταφραστής (compiler) ονομάζεται ένα πρόγραμμα υπολογιστή, το οποίο διαβάζει κώδικα γραμμένο σε μια γλώσσα προγραμματισμού (αρχικό πρόγραμμα) και τον μεταφράζει σε ισοδύναμο κώδικα σε μια άλλη γλώσσα προγραμματισμού (τελικό πρόγραμμα). Επιπλέον εμφανίζει διαγνωστικά μηνύματα, συνήθως μηνύματα λάθους, μερικές φορές όμως και μηνύματα προειδοποίησης.

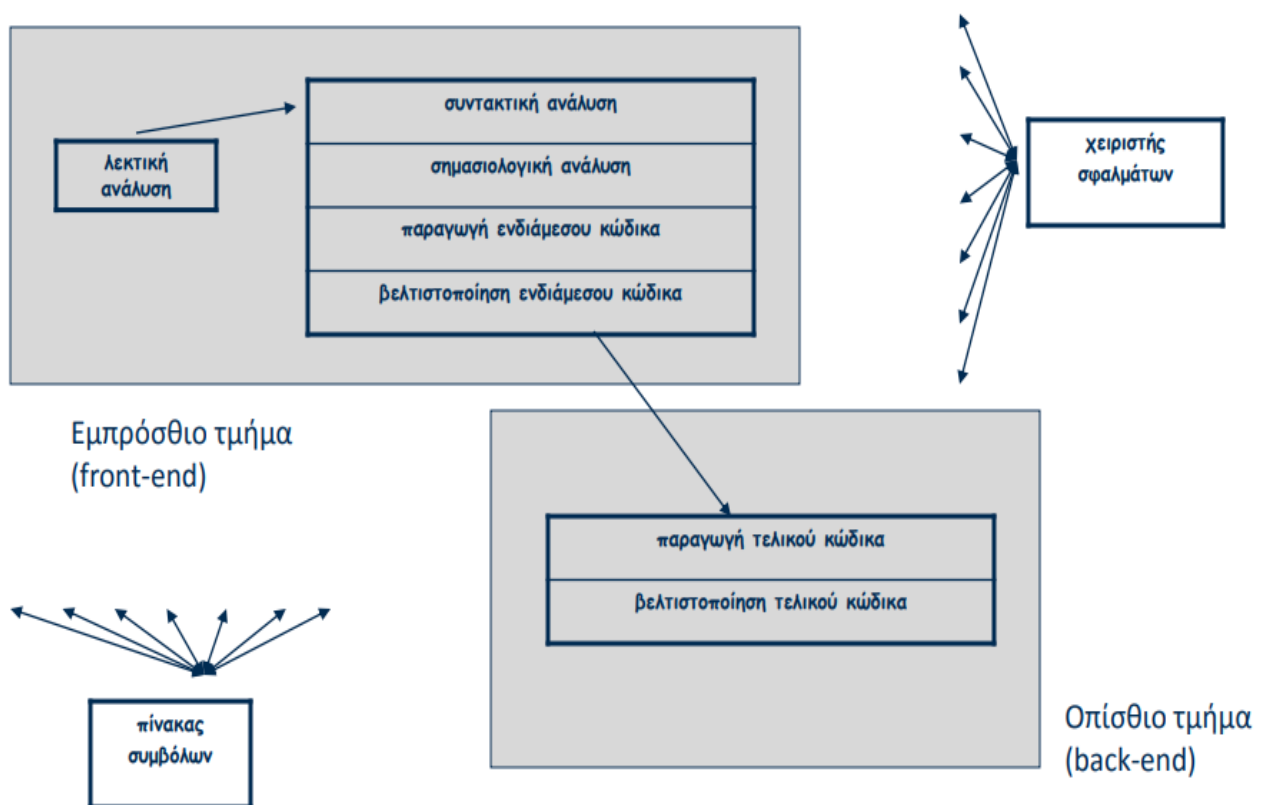
## 1.2 Οι απαιτήσεις από ένα μεταγλωττιστή

Οι βασικές απαιτήσεις που θα πρέπει να ικανοποιεί ένας μεταγλωττιστής είναι οι εξής:

- ✓ Σωστή λειτουργία
- ✓ Συμμόρφωση με προδιαγραφές αρχικής και τελικής γλώσσας
- ✓ Μετάφραση προγραμμάτων αυθαίρετα μεγάλου μήκους
- ✓ Παραγωγή αποδοτικού κώδικα
- ✓ Μικρός χρόνος εκτέλεσης
- ✓ Μικρές απαιτήσεις μνήμης κατά τη μεταγλώττισης
- ✓ Καλά διαγνωστικά μηνύματα
- ✓ Δυνατότητα συνέχισης ύστερα από εντοπισμό σφαλμάτων
- ✓ Μεταφερσιμότητα

### 1.3 Οι φάσεις της μεταγλώττισης

Η μεταγλώττιση ενός προγράμματος μπορεί να διαχωριστεί στις εξής φάσεις: λεκτική ανάλυση, συντακτική ανάλυση, σημασιολογική ανάλυση, παραγωγή ενδιάμεσου κώδικα, βελτιστοποίηση ενδιάμεσου κώδικα, παραγωγή τελικού κώδικα και βελτιστοποίηση τελικού κώδικα. Οι φάσεις αυτές περιγράφονται πιο αναλυτικά σε επόμενα κεφάλαια, όπου παρουσιάζεται και ο τρόπος που χρησιμοποιήθηκαν στη δημιουργία του μεταγλωττιστή που φτιάξαμε για τη γλώσσα μας.



“Οργάνωση Μεταγλωττιστή”

## 2. Η ΓΛΩΣΣΑ EEL

### 2.1 Τα χαρακτηριστικά της γλώσσας EEL

Η EEL (Early Experimental Language) είναι μια μικρή γλώσσα προγραμματισμού. Παρόλο που οι προγραμματιστικές της ικανότητες είναι μικρές, η εκπαιδευτική αυτή γλώσσα περιέχει πλούσια στοιχεία και η κατασκευή του μεταγλωττιστή της έχει να παρουσιάσει αρκετό ενδιαφέρον, αφού περιέχονται σε αυτήν πολλές εντολές που χρησιμοποιούνται από άλλες γλώσσες, καθώς και κάποιες πρωτότυπες. Η EEL υποστηρίζει συναρτήσεις και διαδικασίες, μετάδοση παραμέτρων με αναφορά και τιμή, αναδρομικές κλήσεις, και άλλες ενδιαφέρουσες δομές. Επίσης, επιτρέπει φώλιασμα στη δήλωση συναρτήσεων και διαδικασιών κάτι που λίγες γλώσσες υποστηρίζουν (το υποστηρίζει η Pascal, δεν το υποστηρίζει η C). Από την άλλη όμως πλευρά, η EEL δεν υποστηρίζει βασικά προγραμματιστικά εργαλεία όπως η δομή for, ή τύπους δεδομένων όπως οι πραγματικοί αριθμοί και οι συμβολοσειρές. Οι παραλήψεις αυτές έχουν γίνει ώστε να απλουστευτεί η διαδικασία κατασκευής του μεταγλωττιστή, μία απλούστευση όμως που έχει να κάνει μόνο με τη μείωση των γραμμών κώδικα και όχι με τη δυσκολία κατασκευής του.

#### Λεκτικές μονάδες

Το αλφάβητο της EEL 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 («;», «,», «:=»)

- καθώς και τα σύμβολα ομαδοποίησης («(»,«)»,«[»,«]»)
- και διαχωρισμού σχολίων («/\*»,«\*/»,«//»).

Τα σύμβολα «[» και «]» χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα «(» και «)» στις αριθμητικές παραστάσεις.

Μερικές λέξεις είναι δεσμευμένες:

**program, endprogram,**

**declare, enddeclare,**

**if, then, else, endif,**

**while, endwhile,**

**repeat, endrepeat, exit,**

**switch, case, endswitch,**

**forcase, when, endforcase,**

**procedure, endprocedure, function, endfunction, call, return, in, inout,**

**and, or, not, true, false,**

**input, print**

Οι λέξεις αυτές δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές. Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Υπάρχουν και οι σταθερές true και false. Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα και ψηφία, αρχίζοντας όμως από γράμμα. Ο μεταγλωττιστής λαμβάνει υπόψη του μόνο τα τριάντα πρώτα γράμματα. Οι λευκοί χαρακτήρες (tab, space, return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, σταθερές. Το ίδιο ισχύει και για τα σχόλια, τα οποία

πρέπει να βρίσκονται μέσα στα σύμβολα /\* και \*/ ή να βρίσκονται μετά το σύμβολο // και ως το τέλος της γραμμής.

## Μορφή προγράμματος

```
program id
    declarations
    subprograms
    statements
endprogram
```

## Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η EEL είναι οι ακέραιοι αριθμοί. Οι ακέραιοι αριθμοί πρέπει να έχουν τιμές από –32767 έως 32767. Η δήλωση γίνεται με την εντολή declare. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Το τέλος της δήλωσης των μεταβλητών γίνεται με την εντολή enddeclare.

## Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

- (1) Μοναδιαίοι λογικοί: «not»
- (2) Πολλαπλασιαστικοί: «\*», «/»
- (3) Μοναδιαίοι προσθετικοί: «+», «-»
- (4) Δυαδικοί προσθετικοί: «+», «-»
- (5) Σχεσιακοί «=», «<», «>», «<>», «<=», «>=»
- (6) Λογικό «and»,
- (7) Λογικό «or»



## Δομές της γλώσσας

### Εκχώρησης

Id := expression

Χρησιμοποιείται για την ανάθεση της τιμής μίας μεταβλητής ή μίας σταθεράς, ή μίας έκφρασης σε μία μεταβλητή.

### Απόφασης if

```
if condition then
    statements
[else
    statements]
endif
```

Η εντολή απόφασης **if** εκτιμάει εάν ισχύει η συνθήκη condition και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές που ακολουθούν το **then** έως ότου συναντηθεί **else** ή **endif**. Το **else** δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται σε αγκύλη. Οι εντολές που το ακολουθούν εκτελούνται εάν η συνθήκη condition δεν ισχύει. Το **endif** είναι υποχρεωτικό τμήμα της εντολής.

### Επανάληψης repeat

```
repeat
    statements
endrepeat
```

Η εντολή επανάληψης **repeat** επαναλαμβάνει συνεχώς τις εντολές που βρίσκονται ανάμεσα στο **repeat** και στο **endrepeat** έως ότου εκτελεστεί η **exit**. Με την εντολή **exit** η εκτέλεση μεταφέρεται έξω από τον βρόχο.

### Επιστροφής

return expression

Χρησιμοποιείται μέσα σε συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης.

## Επανάληψης **while**

```
while condition
    statements
endwhile
```

Η εντολή επανάληψης **while** επαναλαμβάνει τις εντολές που βρίσκονται ανάμεσα στο **while** και στο **endwhile** για όσο ισχύει η συνθήκη condition. Αν στον πρώτο έλεγχο, η συνθήκης δεν είναι αληθής, τότε οι εντολές statements δεν εκτελούνται ποτέ.

## Απόφασης **switch**

```
switch expression
    case expression : statements
    (case expression : statements)*
endswitch
```

Η δομή απόφασης **switch** ελέγχει εάν η έκφραση expression που υπάρχει ακριβώς μετά από τη δεσμευμένη λέξη **switch** ισούται με κάποια από τις υπόλοιπες expression που βρίσκονται μετά τα **case**, εξετάζοντας τες κατά σειρά. Για την πρώτη από αυτές που ισχύει ότι είναι ίση με την αρχική expression εκτελούνται οι εντολές που ακολουθούν το σύμβολο “:”. Αφού εκτελεστεί κάποια από τις statements τότε ο έλεγχος μεταβαίνει έξω από τη δομή **switch**. Εάν καμία από τις statements δεν εκτελεστεί, τότε πάλι ο έλεγχος μεταβαίνει έξω από τη δομή **switch**.

## Επανάληψης **for** case

```
for case
    when condition : statements
    (when condition : statements)*
endfor case
```

Η δομή επανάληψης **for** case ελέγχει τις condition που βρίσκονται μετά τα **when**, εξετάζοντας τες κατά σειρά. Για κάθε μία από αυτές που ισχύει εκτελούνται οι statements που ακολουθούν το σύμβολο “:”. Θα εξεταστούν όλες οι condition και θα εκτελεστούν όλες οι statements των οποίων οι condition ισχύουν. Αφότου εξετατούν όλες οι **when** ο έλεγχος μεταβαίνει έξω από τη δομή **for** case εάν καμία από τις statements δεν έχει εκτελεστεί ή

μεταβαίνει στην αρχή της **forcase** εάν έστω και μία από τις stements έχει εκτελεστεί.

### Εξόδου

**print** expression

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του expression

### Εισόδου

**input** id

Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο

### Υποπρογράμματα

Η EEL υποστηρίζει και συναρτήσεις και διαδικασίες.

Διαδικασία:

```
procedure id (formal_pars)
    declarations
    subprograms
    statements
endprocedure
```

Συνάρτηση:

```
function id (formal_pars)
    declarations
    subprograms
    statements
endfunction
```

Η «formal\_pars» είναι η λίστα των τυπικών παραμέτρων. Οι διαδικασίες και οι συναρτήσεις μπορούν να φωλιάσουν η μία μέσα στην άλλη και οι κανόνες εμφάνισης είναι όπως της PASCAL. Η επιστροφή της τιμής μιας συνάρτησης γίνεται με την **return**

Η κλήση μιας διαδικασίας γίνεται με την call

**call** id (actual\_parameters)

ενώ όταν πρόκειται για συνάρτηση, αυτή λαμβάνει μέρος στις αριθμητικές παραστάσεις σαν τελούμενο. π.χ.

$$D = a + f(\mathbf{in} \ x)$$

όπου f η συνάρτηση και x παράμετρος που περνάει με τιμή.

### Μετάδοση παραμέτρων

Η EEL υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

- με σταθερή τιμή. Δηλώνεται με τη λεκτική μονάδα **in**. Αλλαγές στην τιμή της δεν επιστρέφονται στον καλόν πρόγραμμα
- με αναφορά. Δηλώνεται με τη λεκτική μονάδα **inout**. Κάθε αλλαγή στη τιμή της μεταφέρεται και στο πρόγραμμα που κάλεσε τη διαδικασία ή τη συνάρτηση

Στην κλήση μίας συνάρτησης ή μίας διαδικασίας οι πραγματικοί παράμετροι συντάσσονται μετά από τις λέξεις κλειδιά **in** και **inout**, ανάλογα με το αν περνάνε με τιμή ή αναφορά.

## 2.2 Η γραμματική της γλώσσας EEL

```
<program>      ::= program id <block> endprogram

<block>        ::= <declarations> <subprograms> <statements>

<declarations> ::= ε | declare <varlist> enddeclare

<varlist>      ::= ε | id ( , id ) *

<subprograms>  ::= ( <procorfunc> ) *

<procorfunc>   ::= procedure id <procorfuncbody> endprocedure |
                  function id <procorfuncbody> endfunction

<procorfuncbody> ::= <formalpars> <block>

<formalpars>    ::= ( <formalparlist> )

<formalparlist> ::= <formalparitem> ( , <formalparitem> ) * | ε

<formalparitem> ::= in id | inout id

<statements>   ::= <statement> ( ; <statement> ) *

<statement>    ::= ε |
                  <assignment-stat> |
                  <if-stat> |
                  <while-stat> |
                  <repeat-stat> |
                  <exit-stat> |
                  <switch-stat> |
                  <forcase-stat> |
                  <call-stat> |
                  <return-stat> |
                  <input-stat> |
                  <print-stat>

<assignment-stat> ::= id := <expression>

<if-stat>          ::= if <condition> then <statements> <elsepart> endif

<elsepart>        ::= ε | else <statements>

<repeat-stat>     ::= repeat <statements> endrepeat
```

<exit-stat>	::= <b>exit</b>
<while-stat>	::= <b>while</b> <condition> <statements> <b>endwhile</b>
<switch-stat>	::= <b>switch</b> <expression> ( <b>case</b> <expression> : <statements> )+ <b>endswitch</b>
<forcase-stat>	::= <b>forcase</b> ( <b>when</b> <condition> : <statements> )+ <b>endforcase</b>
<call-stat>	::= <b>call</b> id <actualpars>
<return-stat>	::= <b>return</b> <expression>
<print-stat>	::= <b>print</b> <expression>
<input-stat>	::= <b>input</b> id
<actualpars>	::= ( <actualparlist> )
<actualparlist>	::= <actualparitem> ( , <actualparitem> )*   $\epsilon$
<actualparitem>	::= <b>in</b> <expression>   <b>inout</b> id
<return-stat>	::= <b>return</b> <expression>
<condition>	::= <boolterm> ( <b>or</b> <boolterm> )*
<boolterm>	::= <boolfactor> ( <b>and</b> <boolfactor> )*
<boolfactor>	::= <b>not</b> [ <condition> ]   [ <condition> ]   <expression> <relational-oper> <expression>   <b>true</b>   <b>false</b>
<expression>	::= <optional-sign> <term> ( <add-oper> <term> )*
<term>	::= <factor> ( <mul-oper> <factor> )*
<factor>	::= constant   ( <expression> )   id <idtail>
<idtail>	::= $\epsilon$   <actualpars>
<relational-oper>	::= =   <=   >=   >   <   <>
<add-oper>	::= +   -
<mul-oper>	::= *   /
<optional-sign>	::= $\epsilon$   <add-oper>

### 3. ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

#### 3.1 Ο ορισμός και η λειτουργία του λεκτικού αναλυτή

Ο λεκτικός αναλυτής είναι μία συνάρτηση. Ο ρόλος του είναι να διαβάζει γράμμα – γράμμα το πηγαίο πρόγραμμα και να επιστρέφει την επόμενη λεκτική μονάδα και έναν ακέραιο που τη χαρακτηρίζει. Εσωτερικά λειτουργεί σαν ένα αυτόματο καταστάσεων, το οποίο ξεκινά από μία αρχική κατάσταση και με την είσοδο κάθε χαρακτήρα αλλάζει κατάσταση, έως ότου συναντήσει μία τελική κατάσταση. Το αυτόματο καταστάσεων αναγνωρίζει δεσμευμένες λέξεις, σύμβολα της γλώσσας, αναγνωριστικά και σταθερές, καθώς επίσης και λάθη.



### 3.2 Πίνακας καταστάσεων για τη γλώσσα EEL

Οι καταστάσεις για τις γραμμές είναι:

error = -1, OK = -2, state0 = 0, state1 = 1, state2 = 2, state3 = 3, state4 = 4, state5 = 5, state6 = 6, state7 = 7, state8 = 8, state9 = 9

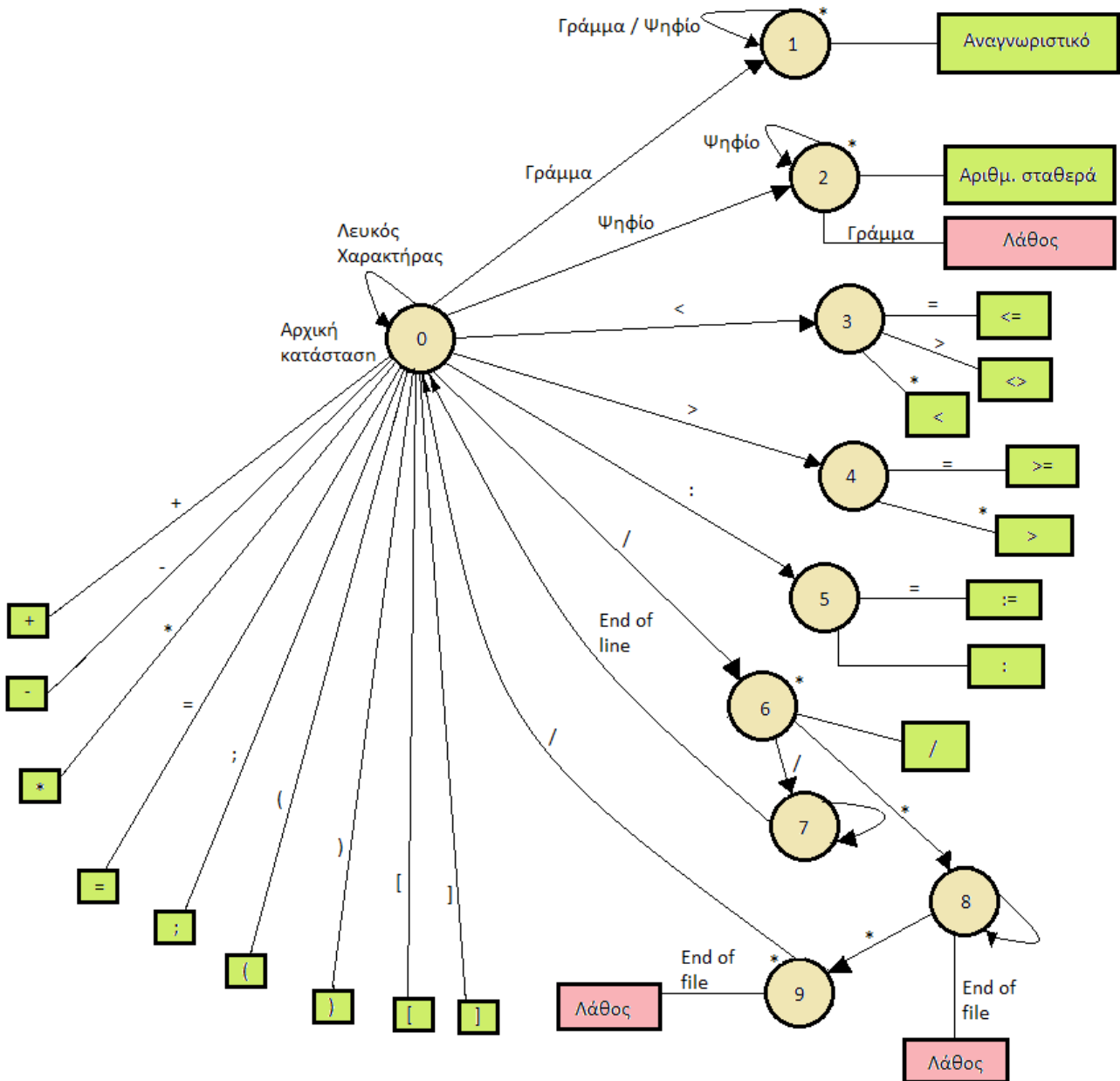
Οι καταστάσεις για τις στήλες είναι:

letter = 0, digit = 1, plus = 2, minus = 3, asterisk = 4, slash = 5, less = 6, greater = 7, equals = 8, colon = 9, semicolon = 10, comma = 11, open\_round\_bracket = 12, close\_round\_bracket = 13, open\_square\_bracket = 14, close\_square\_bracket = 15, end\_of\_file = 16, end\_of\_line = 17, other = 18

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	stat e1	stat e2	OK	OK	OK	stat e6	stat e3	stat e4	OK	stat e5	OK	OK	OK	OK	OK	OK	OK	stat e0	stat e0
1	stat e1	stat e1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
2	erro r	stat e2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
3	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
4	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
5	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
6	OK	OK	OK	OK	stat e8	stat e7	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e7	stat e0	stat e7
8	stat e8	stat e8	stat e8	stat e8	stat e9	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	err or	stat e8	stat e8
9	stat e8	stat e8	stat e8	stat e8	stat e8	stat e0	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	stat e8	err or	stat e8	stat e8



### 3.3 Αυτόματο καταστάσεων για τη γλώσσα EEL



### 3.4 Επεξήγηση του κώδικα που αφορά το λεκτικό αναλυτή

Τον κώδικα που αφορά το λεκτικό αναλυτή τον γράψαμε σε γλώσσα Python και βρίσκεται στη συνάρτηση `lex()`.

Στην αρχή της συνάρτησης έχουμε ορίσει:

- Τις εξής κοινόχρηστες μεταβλητές:

token, στην οποία περνιέται η περιγραφή της κάθε λεκτικής μονάδας  
value, στην οποία περνιέται η τιμή της κάθε λεκτικής μονάδας  
lines, η οποία περιέχει κάθε στιγμή τον αριθμό της γραμμής που  
βρίσκεται υπό επεξεργασία  
eof, που περιέχει τη θέση τερματισμού του αρχείου του πηγαίου κώδικα

- Τις καταστάσεις που αφορούν τις γραμμές και τις στήλες του πίνακα καταστάσεων, καθώς και τον ίδιο τον πίνακα καταστάσεων.

- Τις μεταβλητές:

alphabet, που είναι ένα αλφαριθμητικό με όλα τα γράμματα της άλφα-  
βήτας  
numbers, που είναι ένα αλφαριθμητικό με τους αριθμούς από το 0 έως  
και το 9  
whitespace, που είναι μία λίστα με όλους τους λευκούς χαρακτήρες που  
μπορεί να περιέχει η γλώσσα  
reserved words, που είναι μια λίστα που περιέχει όλες τις δεσμευμένες  
λέξεις της γλώσσας

- Και τέλος, κάποια flags.

Τον βασικό κώδικα τον υλοποιήσαμε μέσα σε ένα βρόγχο επανάληψης (while) στον οποίο:

- Αν έχουμε φτάσει σε τελική κατάσταση, δηλαδή OK ή error, ή αν ακόμη έχουμε φτάσει στο τέλος του αρχείου ο βρόγχος τερματίζει. Οι καταστάσεις προκύπτουν κάθε φορά από τον πίνακα καταστάσεων.
- Διαφορετικά, διαβάζουμε ένα χαρακτήρα από το αρχείο του πηγαίου κώδικα και ελέγχουμε:

- Αν ο χαρακτήρας ανήκει στα γράμματα της άλφα-βήτας και άρα είναι γράμμα
- Αν ο χαρακτήρας ανήκει στους αριθμούς και άρα είναι ψηφίο
- Αν ο χαρακτήρας ισούται με κάποιο από τα “+”, “-”, “\*”, “/”, “<”, “>”, “=”, “:”, “;”, “,”, “(”, “)”, “[”, “]”
- Αν ο χαρακτήρας ισούται με “\n”, δηλαδή είναι αλλαγή γραμμής
- Αν ο χαρακτήρας είναι λευκός χαρακτήρας
- Αν ο χαρακτήρας δεν ανήκει στη γλώσσα

➤ Πιο αναλυτικά σε κάθε περίπτωση:

- Αν είναι γράμμα ελέγχουμε αν βρίσκεται σε σχόλια μέσω των flags και αν δεν βρίσκεται συνεχίζουμε να διαβάζουμε μέχρι να φτάσουμε σε κάτι που δεν είναι γράμμα ή αριθμός ή στο τέλος αρχείου. Τότε ελέγχουμε αν η λέξη που προέκυψε έχει το επιθυμητό μέγεθος και αν όχι την κόβουμε κατάλληλα. Επίσης, ελέγχουμε αν η λέξη είναι δεσμευμένη ή απλή λέξη και ανάλογα με το τι είναι της δίνουμε την κατάλληλη περιγραφή.
- Αν είναι ψηφίο ελέγχουμε αν βρίσκεται σε σχόλια μέσω των flags και αν δεν βρίσκεται συνεχίζουμε να διαβάζουμε μέχρι να φτάσουμε σε κάτι που δεν είναι αριθμός ή στο τέλος αρχείου. Αν εντοπίσουμε κάποιο γράμμα στη λέξη μετά από αριθμό, εμφανίζουμε μήνυμα λάθους. Όταν η λέξη τελειώσει ελέγχουμε η λέξη (αριθμός) βρίσκεται στα σωστά όρια ανάλογα της δίνουμε την κατάλληλη περιγραφή.
- Αν είναι κάποιος από τους χαρακτήρες “+” “-” “\*” “:” “,” “(” “)” “[” “]”, του δίνουμε απλώς την κατάλληλη περιγραφή.
- Αν είναι ο χαρακτήρας “/”, τότε ελέγχουμε τον επόμενο χαρακτήρα και αν αυτός είναι “/”, τότε ανοίγουν σχόλια γραμμής, αν είναι \*, τότε ανοίγουν σχόλια πολλών γραμμών, ενώ διαφορετικά είναι απλά σύμβολο “/” οπότε του δίνουμε την κατάλληλη περιγραφή.
- Αν είναι ο χαρακτήρας “<”, τότε ελέγχουμε τον επόμενο χαρακτήρα και αν αυτός δεν είναι “>” ή “=”, τότε είναι απλά σύμβολο “<” και του δίνουμε την κατάλληλη περιγραφή.
- Αν είναι ο χαρακτήρας “>”, τότε ελέγχουμε με τα flags αν ο προηγούμενος χαρακτήρας ήταν ο “<”. Αν ήταν τότε είναι το σύμβολο

“<>” , ενώ διαφορετικά απλά το “>”. Ανάλογα με το τι είναι του δίνουμε την κατάλληλη περιγραφή.

- Αν είναι ο χαρακτήρας “=” , τότε ελέγχουμε αν ο προηγούμενος χαρακτήρας ήταν ο “>”, οπότε είναι το σύμβολο “>=”, ή ο προηγούμενος χαρακτήρας ήταν ο “<”, οπότε είναι το σύμβολο “<=”, ή ο προηγούμενος χαρακτήρας ήταν ο “:”, οπότε είναι το σύμβολο “:=”, ή είναι απλώς το σύμβολο “=”. Σε κάθε περίπτωση του δίνουμε την κατάλληλη περιγραφή.
- Αν είναι ο χαρακτήρας “:” , τότε ελέγχουμε αν ο επόμενος χαρακτήρας είναι “=” και αν δεν είναι τότε είναι απλά το σύμβολο “:”, οπότε του δίνουμε την κατάλληλη περιγραφή.
- Αν ο χαρακτήρας είναι λευκός απλώς προχωράμε.
- Αν ο χαρακτήρας δεν ανήκει στη γλώσσα εμφανίζουμε μήνυμα λάθους.

## 4. ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

### 4.1 Ο ορισμός και η λειτουργία του συντακτικού αναλυτή

Με τον όρο συντακτικό αναλυτή αναφερόμαστε στο κομμάτι κώδικα που εκτελεί τη συντακτική ανάλυση. Είναι αυτός που καλεί το λεκτικό αναλυτή και σε αυτό γίνεται ο έλεγχος για να διαπιστωθεί εάν το πηγαίο πρόγραμμα ανήκει ή όχι στη γλώσσα. Για το λόγο αυτό δημιουργεί το κατάλληλο “περιβάλλον” μέσα από το οποίο αργότερα θα κληθούν σημαντικές υπορουτίνες. Για τη γλώσσα EEL δημιουργήσαμε συντακτικό αναλυτή που εκτελεί τη συντακτική ανάλυση με αναδρομική κατάβαση και βασίζεται στη γραμματική LL(1). Η γραμματική αυτή (L: left to right, L: leftmost derivation, (1): one look-ahead symbol) αναγνωρίζει από αριστερά στα δεξιά, την αριστερότερη δυνατή παραγωγή και όταν βρίσκεται σε δίλλημα ποιον κανόνα να ακολουθήσει της αρκεί να κοιτάξει το αμέσως επόμενο σύμβολο στην συμβολοσειρά εισόδου.

Η εσωτερική λειτουργία του συντακτικού αναλυτή υλοποιείται μέσω ενός συνόλου υποπρογραμμάτων, καθ’ ένα από τα οποία αφορά κάποιον κανόνα γραμματικής. Για κάθε μη τερματικό σύμβολο του πηγαίου προγράμματος καλείται το αντίστοιχο υποπρόγραμμα. Για κάθε τερματικό σύμβολο, εάν ο λεκτικός αναλυτής επιστρέψει λεκτική μονάδα που αντιστοιχεί στο τερματικό αυτό σύμβολο, τότε η λεκτική μονάδα αναγνωρίζεται επιτυχώς, διαφορετικά υπάρχει λάθος και καλείται ο διαχειριστής σφαλμάτων. Όταν αναγνωριστεί και η τελευταία λέξη του πηγαίου προγράμματος, τότε η συντακτική ανάλυση είναι επιτυχής.



## 4.4 Επεξήγηση του κώδικα που αφορά το συντακτικό αναλυτή

Για κάθε κανόνα της γραμματικής της γλώσσας EEL, φτιάξαμε μία συνάρτηση. (Η γραμματική της γλώσσας EEL παρουσιάζεται αναλυτικά στην υποενότητα 2.2.)

Οι συναρτήσεις είναι οι εξής:

- program: Η συνάρτηση αυτή ελέγχει αν το πηγαίο πρόγραμμα ξεκινάει με **program** και **id**, και αν τελειώνει με **endprogram**. Για ότι υπάρχει μετά το **program id** καλεί τη συνάρτηση block. Αν κάτι πάει στραβά εμφανίζει μήνυμα λάθους.
- block: Η συνάρτηση αυτή καλεί πρώτα τη συνάρτηση declarations, στη συνέχεια τη συνάρτηση subprograms και τέλος, τη συνάρτηση statements.
- declarations: Η συνάρτηση αυτή ελέγχει αν έχουμε κενό ή τη δεσμευμένη λέξη **declare**. Αν δει **declare** καλεί τη συνάρτηση varlist και περιμένει στο τέλος να υπάρχει η δεσμευμένη λέξη **enddeclare**, διαφορετικά εμφανίζει μήνυμα λάθους.
- varlist: Η συνάρτηση αυτή αφορά τα περιεχόμενα μεταξύ της declare και της enddeclare και επιτρέπει να είναι **id** ή **id, id,...** ή **κενό (ε)**. Σε κάθε άλλη περίπτωση εμφανίζει μήνυμα λάθους.
- subprograms: Η συνάρτηση αυτή καλεί τη συνάρτηση procorfunc.
- procorfunc: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **procedure** και **id** ή τη δεσμευμένη λέξη **function** και **id**. Στην πρώτη περίπτωση, καλεί τη συνάρτηση procorfuncbody και περιμένει στο τέλος τη δεσμευμένη λέξη **endprocedure**, ενώ στη δεύτερη περίπτωση καλεί τη procorfuncbody και περιμένει στο τέλος τη δεσμευμένη λέξη **endfunction**. Αν κάτι πάει στραβά εμφανίζει μήνυμα λάθους.
- procorfuncbody: Η συνάρτηση αυτή καλεί αρχικά τη συνάρτηση formalpars και στη συνέχεια, τη συνάρτηση block.

- formalpars: Η συνάρτηση αυτή ελέγχει αν έχουμε **άνοιγμα παρένθεσης**, καλεί τη συνάρτηση `formalparlist` και στο τέλος περιμένει **κλείσιμο παρένθεσης**. Αν κάτι πάει στραβά, εμφανίζει μήνυμα λάθους.
- formalparlist: Η συνάρτηση αυτή καλεί τη συνάρτηση `formalparitem`, και όσο διαβάζει κόμμα, δηλαδή υπάρχει ακόμα κάτι, συνεχίζει να καλεί τη συνάρτηση `formalparitem`.
- formalparitem: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **in** και **id** ή τη δεσμευμένη λέξη **inout** και **id**. Σε κάθε άλλη περίπτωση εμφανίζει μήνυμα λάθους.
- statements: Η συνάρτηση αυτή καλεί τη συνάρτηση `statement`, και όσο διαβάζει ερωτηματικό, συνεχίζει να καλεί τη συνάρτηση `statement`.
- statement: Η συνάρτηση αυτή περιμένει **κενό** ή κάποια **περιγραφή λεκτικής μονάδας**. Αν διαβάσει περιγραφή λεκτικής μονάδας, καλεί την αντίστοιχη συνάρτηση.
- assignment-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **id** και έπειτα **ανάθεση**. Τότε καλεί τη συνάρτηση `expression`. Σε κάθε άλλη περίπτωση, εμφανίζει μήνυμα λάθους.
- if-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **if**. Τότε καλεί τη συνάρτηση `condition` και περιμένει τη δεσμευμένη λέξη **then**. Μόλις τη δει καλεί τη συνάρτηση `statements` και έπειτα τη συνάρτηση `elsepart`. Στο τέλος περιμένει τη δεσμευμένη λέξη **endif**. Αν κάτι πάει στραβά εμφανίζει μήνυμα λάθους.
- elsepart: Η συνάρτηση αυτή περιμένει **κενό** ή περιγραφή της λεκτικής μονάδας που αναφέρεται στη δεσμευμένη λέξη **else**. Αν δει τη λέξη αυτή καλεί τη συνάρτηση `statements`.
- repeat-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **repeat**. Τότε καλεί τη συνάρτηση `statements` και περιμένει στο τέλος να δει τη δεσμευμένη λέξη **endrepeat**. Σε κάθε άλλη περίπτωση εμφανίζει μήνυμα

λάθους.

- exit-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **exit**. Διαφορετικά, εμφανίζει μήνυμα λάθους.
- while-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **while**. Σε αυτή την περίπτωση καλεί τη συνάρτηση `condition` και έπειτα τη συνάρτηση `statements`. Στο τέλος, περιμένει τη δεσμευμένη λέξη **endwhile**. Αν κάτι πάει στραβά εμφανίζει μήνυμα λάθους.
- switch-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **switch**. Τότε για κάθε φορά που βλέπει τη δεσμευμένη λέξη **case** καλεί τη συνάρτηση `expression` και για κάθε **άνω κάτω τελεία** μετά από αυτήν καλεί τη συνάρτηση `statements`. Στο τέλος περιμένει τη δεσμευμένη λέξη **endswitch**. Αν κάτι πάει στραβά εμφανίζει μήνυμα λάθους.
- forcase-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **forcase**. Τότε για κάθε φορά που βλέπει τη δεσμευμένη λέξη **when** καλεί τη συνάρτηση `condition` και για κάθε **άνω κάτω τελεία** μετά από αυτήν καλεί τη συνάρτηση `statements`. Στο τέλος περιμένει τη δεσμευμένη λέξη **endforcase**. Αν κάτι πάει στραβά εμφανίζει μήνυμα λάθους.
- call-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **call** και **id**. Τότε καλεί τη συνάρτηση `actualpars`. Διαφορετικά εμφανίζει μήνυμα λάθους.
- return-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **return**. Τότε καλεί τη συνάρτηση `expression`. Διαφορετικά εμφανίζει μήνυμα λάθους.
- print-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **print**. Τότε καλεί τη συνάρτηση `expression`. Διαφορετικά εμφανίζει μήνυμα λάθους.
- input-stat: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **input** και **id**. Διαφορετικά εμφανίζει μήνυμα



λάθους.

- actualpars: Η συνάρτηση αυτή ελέγχει αν έχουμε **άνοιγμα παρένθεσης**, καλεί τη συνάρτηση actualparlist και στο τέλος περιμένει **κλείσιμο παρένθεσης**. Αν κάτι πάει στραβά, εμφανίζει μήνυμα λάθους.
- actualparlist: Η συνάρτηση αυτή καλεί τη συνάρτηση actualparitem, και όσο διαβάζει κόμμα, δηλαδή υπάρχει ακόμα κάτι, συνεχίζει να καλεί τη συνάρτηση actualparitem.
- actualparitem: Η συνάρτηση αυτή ελέγχει αν έχουμε τη δεσμευμένη λέξη **in**, και τότε καλεί τη συνάρτηση expression, ή τη δεσμευμένη λέξη **inout** και έπειτα **id**. Σε κάθε άλλη περίπτωση εμφανίζει μήνυμα λάθους.
- condition: Η συνάρτηση αυτή καλεί τη συνάρτηση boolterm και για κάθε φορά που διαβάζει τη δεσμευμένη λέξη **or** συνεχίζει να καλεί τη συνάρτηση boolterm.
- boolterm: Η συνάρτηση αυτή καλεί τη συνάρτηση boolfactor και για κάθε φορά που διαβάζει τη δεσμευμένη λέξη **and** συνεχίζει να καλεί τη συνάρτηση boolfactor.
- boolfactor: Η συνάρτηση αυτή περιμένει (1) τη δεσμευμένη λέξη **not** ή (2) **άνοιγμα αγκύλης**, ή (3) ένα από τα “**πρόσθεση**”, “**αφαίρεση**”, “**ψηφίο**”, “**id**”, “**άνοιγμα παρένθεσης**”, ή (4) τη δεσμευμένη λέξη **true** ή (5) τη δεσμευμένη λέξη **false**. Στην πρώτη περίπτωση, μόλις δει **άνοιγμα αγκύλης** καλεί τη συνάρτηση condition και περιμένει **κλείσιμο αγκύλης** στο τέλος. Στη δεύτερη περίπτωση, καλεί τη συνάρτηση condition και περιμένει **κλείσιμο αγκύλης**. Στην τρίτη περίπτωση, καλεί τη συνάρτηση expression, έπειτα τη συνάρτηση relational\_oper και τέλος ξανά τη συνάρτηση expression. Σε κάθε άλλη περίπτωση εκτός από αυτές τις 5 εμφανίζει μήνυμα λάθους.
- expression: Η συνάρτηση αυτή καλεί τη συνάρτηση optional\_sign και έπειτα τη συνάρτηση term. Στη συνέχεια, όσο διαβάζει **πρόσθεση** ή **αφαίρεση** καλεί πρώτα τη

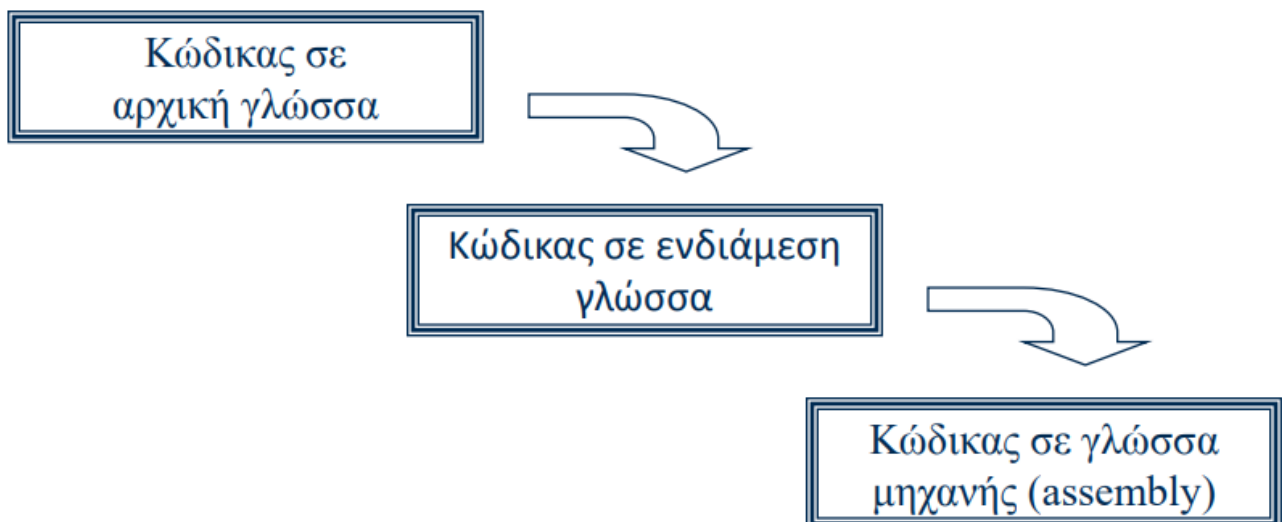
συνάρτηση mul\_oper και μετά τη συνάρτηση factor.

- term: Η συνάρτηση αυτή καλεί τη συνάρτηση factor. Στη συνέχεια, όσο διαβάζει **πολλαπλασιασμό** ή **διαίρεση** καλεί πρώτα τη συνάρτηση mul\_oper και μετά τη συνάρτηση factor.
- factor: Η συνάρτηση αυτή περιμένει **ψηφίο** ή **άνοιγμα παρένθεσης**, όπου καλεί τη συνάρτηση expression και περιμένει στο τέλος **κλείσιμο παρένθεσης**, ή τη δεσμευμένη λέξη **id**, όπου καλεί τη συνάρτηση idtail. Σε κάθε άλλη περίπτωση εμφανίζει μήνυμα λάθους.
- idtail: Η συνάρτηση αυτή περιμένει **κενό**, ή **άνοιγμα παρένθεσης** όπου καλεί τη συνάρτηση actualpars.
- relational-oper: Η συνάρτηση αυτή περιμένει **“ίσο”**, ή **“μικρότερο ίσο”**, ή **“μεγαλύτερο ίσο”**, ή **“μεγαλύτερο”**, ή **“μικρότερο”**, ή **“διάφορο”**. Σε κάθε άλλη περίπτωση εμφανίζει μήνυμα λάθους.
- add-oper: Η συνάρτηση αυτή περιμένει **πρόσθεση** ή **αφαίρεση**. Διαφορετικά εμφανίζει μήνυμα λάθους.
- mul-oper: Η συνάρτηση αυτή περιμένει **πολλαπλασιασμό** ή **διαίρεση**. Διαφορετικά εμφανίζει μήνυμα λάθους.
- optional-sign: Η συνάρτηση αυτή περιμένει **κενό** ή σύμβολο **πρόσθεσης / αφαίρεσης**, όπου καλεί τη συνάρτηση add\_oper.

## 5. ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

### 5.1 Ο ορισμός του ενδιάμεσου κώδικα

Με τον όρο ενδιάμεσο κώδικα αναφερόμαστε στην πραγματικότητα σε ένα σύνολο από τετράδες. Κάθε τετράδα αποτελείται από έναν τελεστή και τρία τελούμενα (πχ. +, a, b, c). Επιπλέον κάθε τετράδα είναι αριθμημένη με ένα μοναδικό αριθμό που τη χαρακτηρίζει (πχ. 100: +, a, b, c). Η εκτέλεση των τετράδων γίνεται σύμφωνα με τον τρόπο που είναι αριθμημένες.



## 5.2 Τελεστές – Αρχή και τέλος ενότητας – Συναρτήσεις – Διαδικασίες στον ενδιάμεσο κώδικα

### Τελεστές αριθμητικών πράξεων

Αφορούν τετράδες της μορφής `op, x, y, z`

- όπου το `op` είναι ένα εκ των: `+`, `-`, `*`, `/`
- τα τελούμενα `x, y` μπορεί να είναι: όνομα μεταβλητών / αριθμητικές σταθερές
- το τελούμενο `z` μπορεί να είναι: όνομα μεταβλητής

### Τελεστής εκχώρησης

Αφορά τετράδες της μορφής `:=, x, _, z`

- το τελούμενο `x` μπορεί να είναι: όνομα μεταβλητής/ αριθμητική σταθερά
- το τελούμενο `z` μπορεί να είναι: όνομα μεταβλητής
- η τιμή του `x` εκχωρείται στη μεταβλητή `z`, δηλαδή αντιστοιχεί στην εκχώρηση `z := x`

### Τελεστής άλματος χωρίς συνθήκη

Αφορά τετράδες της μορφής `jump, _, _, z`

- όπου γίνεται μεταπήδηση χωρίς όρους στη θέση `z`

Και τετράδες της μορφής `relop, x, y, z`

- όπου `relop` είναι ένας από τους τελεστές: `=`, `>`, `<`, `<>`, `>=`, `<=`
- και γίνεται μεταπήδηση στη θέση `z`, αν ισχύει η `x relop y`

### Αρχή και τέλος ενότητας

Αφορά την τετράδα `begin_block, name, _, _`

- όπου δηλώνει την αρχή υποπρογράμματος ή προγράμματος με το όνομα `name`,

Την τετράδα `end_block, name, _, _`

- όπου δηλώνει το τέλος υποπρογράμματος ή προγράμματος με το όνομα `name`

Και την τετράδα `halt, _, _, _`

- όπου δηλώνει τον τερματισμό του προγράμματος

## Συναρτήσεις – Διαδικασίες

Αφορά την τετράδα **par, x, m, \_**

- όπου x παράμετρος συνάρτησης και m ο τρόπος μετάδοσης (CV: μετάδοση με τιμή, REF: μετάδοση με αναφορά, RET: επιστροφή τιμής συνάρτησης)

Την τετράδα **call, name, \_, \_**

- για κλήση συνάρτησης name

Και την τετράδα **ret, x, \_, \_**

- για επιστροφή τιμής συνάρτησης

Παράδειγμα κλήσης συνάρτησης: **x := foo(in a, inout b)**

Παράδειγμα κλήσης διαδικασίας: **call foo(in a, inout b)**



### 5.3 Βοηθητικές υπορουτίνες για τον ενδιάμεσο κώδικα

Για την παραγωγή του ενδιάμεσου κώδικα υλοποιήσαμε και χρησιμοποιήσαμε τις παρακάτω υπορουτίνες:

- nextquad(): Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.
- genquad(op, x, y, z): Δημιουργεί την επόμενη τετράδα (op, x, y, z)
- newtemp(): Δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή.  
Οι προσωρινές μεταβλητές είναι της μορφής T\_1, T\_2, T\_3, ...
- emptylist(): Δημιουργεί μία κενή λίστα ετικετών τετράδων.
- makelist(x): Δημιουργεί μία λίστα ετικετών τετράδων που περιέχει μόνο το x.
- merge(list<sub>1</sub>, list<sub>2</sub>): Δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών list<sub>1</sub>, list<sub>2</sub>.
- backpatch(list, z): Η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο.  
Η backpatch επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

## 5.4 Επεξήγηση του κώδικα που αφορά τον ενδιάμεσο κώδικα

Για την παραγωγή ενδιάμεσου κώδικα χρησιμοποιήσαμε εκτός από τις υπορουτίνες που αναφέραμε πριν, τα εξής:

- quadsList: Μία κοινόχρηστη λίστα για την αποθήκευση των τετράδων που θα παραχθούν.
- temp\_var: Μία κοινόχρηστη μεταβλητή για να ξέρουμε τον αριθμό της προσωρινής μεταβλητής που θα παραχθεί από τη συνάρτηση newtemp().
- program\_name: Μία κοινόχρηστη μεταβλητή για την αποθήκευση του ονόματος του προγράμματος.
- procedures: Μία κοινόχρηστη λίστα που θα περιέχει όλα τα ονόματα των διαδικασιών.
- functiones: Μία κοινόχρηστη λίστα που θα περιέχει όλα τα ονόματα των διαδικασιών.
- is\_function: Μία κοινόχρηστη μεταβλητή flag που χρησιμοποιείται για να γνωρίζουμε εάν έχουμε συνάρτηση ή διαδικασία. Συγκεκριμένα, έχει την τιμή True, εάν έχουμε συνάρτηση και την τιμή False εάν έχουμε διαδικασία.
- exitlist: Μία κοινόχρηστη λίστα που χρησιμοποιείται στις περιπτώσεις που στον κώδικα υπάρχει exit.

Με τη βοήθεια όλων των παραπάνω προσθέσαμε κάποια κομμάτια κώδικα σε κάποιες από τις συναρτήσεις του συντακτικού αναλυτή που ήδη είχαμε υλοποιήσει. Οι συναρτήσεις που τροποποιήσαμε και η λειτουργία που προσθέσαμε σε αυτές φαίνεται αναλυτικά παρακάτω:

- program: Στη συνάρτηση αυτή, αποθηκεύσαμε το όνομα του προγράμματος στην κοινόχρηστη μεταβλητή `program_name` και το περάσαμε ως παράμετρο στη συνάρτηση `block`.
- block: Η συνάρτηση αυτή παίρνει ως όρισμα το `όνομα` ενός προγράμματος ή υποπρογράμματος και δημιουργεί μία τετράδα `begin_block, name, _, _` που δηλώνει την αρχή ενότητας. Αν εντοπίσει ξανά το όνομα του κύριου προγράμματος, δημιουργεί μία τετράδα `halt, _, _, _` που δηλώνει τον τερματισμό του προγράμματος και επίσης κάθε φορά που κλείνει ένα υποπρόγραμμα ή το κύριο πρόγραμμα δημιουργεί μία τετράδα `end_block, name, _, _`.
- procorfunc: Η συνάρτηση αυτή παίρνει το `όνομα` της εκάστοτε συνάρτησης ή διαδικασίας και το περνάει ως όρισμα στη συνάρτηση `procorfuncbody`, η οποία θα το δώσει στη συνέχεια στη συνάρτηση `block`. Επίσης, αν έχουμε συνάρτηση προσθέτει το όνομά της στη λίστα `functions`, ενώ αν έχουμε διαδικασία, προσθέτει το όνομά της στη λίστα `procedures`.
- assignment\_stat: Η συνάρτηση αυτή παίρνει το `όνομα` της μεταβλητής στην οποία πρόκειται να γίνει ανάθεση (`NAME`), παίρνει από τη συνάρτηση `expression` αυτό που θα ανατεθεί στη μεταβλητή (`E_place`) και παράγει μία τετράδα `:=, E_place, _, NAME`.
- if\_stat: Η συνάρτηση αυτή παίρνει από τη συνάρτηση `condition` 2 λίστες που χρειάζονται συμπλήρωση για το που πρέπει να γίνει μεταπήδηση. Η μία είναι η `B_true` και η άλλη είναι η `B_false`. Στο `then` συμπληρώνεται η λίστα `B_true`, αφού για να πάμε εκεί η συνθήκη πρέπει να ισχύει. Επίσης, στο `endif` συμπληρώνεται η λίστα



**B\_false**, αφού για να πάμε εκεί η συνθήκη δεν πρέπει να ισχύει. Στην περίπτωση που υπάρχει **elif**, δημιουργείται μία λίστα **ifList** με τον επόμενο αριθμό τετράδας που πρόκειται να παραχθεί, δημιουργείται μία τετράδα **jump, \_, \_**, συμπληρώνεται η λίστα **Bfalse** και όταν είναι γνωστός ο επόμενος αριθμός τετράδας που θα παραχθεί μετά το **elif**, συμπληρώνεται και η λίστα **ifList**.

➤ repeat\_stat:

Η συνάρτηση αυτή επειδή εκτελεί επανάληψη, αρχικά αποθηκεύει σε μια μεταβλητή **Bquad** τον αριθμό της τετράδας που παράγεται στην αρχή της επανάληψης, ώστε να μπορεί να επιστρέψει σε εκείνο το σημείο. Εάν εντοπίσει **endrepeat** δημιουργεί μια τετράδα **jump, \_, \_**, **Bquad**, ώστε να επιστρέψει στην αρχή της επανάληψης. Επίσης, φτιάχνει μια λίστα **tmplist**, όπου κρατάει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί, συνενώνει αυτή τη λίστα με την **exitlist** που υπάρχει για να ξέρουμε ποια θα είναι η επόμενη τετράδα στην περίπτωση που συναντήσουμε **exit** και τη συμπληρώνει.

➤ exit\_stat:

Η συνάρτηση αυτή αφορά την περίπτωση που συναντήσουμε **exit**, οπότε προσθέτει στη λίστα **exitlist** τον αριθμό της επόμενης τετράδας και δημιουργεί την τετράδα **jump, \_, \_**, που θα συμπληρωθεί όταν θα ξέρουμε ποιος θα είναι ο αριθμός της επόμενης τετράδας μετά το **exit**.

➤ while\_stat:

Η συνάρτηση αυτή επειδή εκτελεί επανάληψη, αρχικά αποθηκεύει σε μια μεταβλητή **Bquad** τον αριθμό της τετράδας που παράγεται στην αρχή της επανάληψης, ώστε να μπορεί να επιστρέψει σε εκείνο το σημείο. Στη συνέχεια, παίρνει από τη συνάρτηση **condition 2** λίστες που χρειάζονται συμπλήρωση. Η μία είναι η **B\_true** και η άλλη είναι η **B\_false**. Η **B\_true** συμπληρώνεται πριν καλέσουμε τη συνάρτηση **statements**, γιατί τότε η συνθήκη ισχύει. Μετά την κλήση της συνάρτησης **statements**, δημιουργείται η τετράδα **jump, \_, \_**, **Bquad** για να πάμε ξανά στην αρχή της επανάληψης. Όταν δεν

θα ισχύει ποια η συνθήκη θα παραχθεί η επόμενη τετράδα εκτός του **while** και εκεί συμπληρώνεται η λίστα **Bfalse**.

➤ switch\_stat:

Η συνάρτηση αυτή δημιουργεί μία **exitlist**, η οποία συμπληρώνεται όταν είναι γνωστός ο αριθμός τετράδας που παράγεται μετά το **endswitch**. Επίσης, αποθηκεύει στη μεταβλητή **EXP\_1** αυτό που επιστρέφει η συνάρτηση **expression**, που καλείται μετά το **switch**. Στη συνέχεια, για κάθε **case** που υπάρχει, αποθηκεύει στη μεταβλητή **EXP\_2** αυτό που επιστρέφει η συνάρτηση **expression**, που καλείται μετά το **case**. Έπειτα, δημιουργεί μία λίστα **R\_true**, που περιέχει τον επόμενο αριθμό τετράδας και δημιουργεί την τετράδα **=, EXP\_1, EXP\_2, \_**, η οποία αφορά την περίπτωση που ισχύει η συνθήκη. Δημιουργεί τη λίστα **R\_false**, που περιέχει τον αριθμό της επόμενης τετράδας και την τετράδα **jump, \_, \_, \_**, η οποία αφορά την περίπτωση που η συνθήκη δεν ισχύει. Τέλος, πριν την κλήση της συνάρτησης **statements**, συμπληρώνεται η λίστα **R\_true**, γιατί τότε η συνθήκη θα ισχύει, και δημιουργείται μία λίστα **tlist** που αφορά το που θα γίνει η επόμενη μεταπήδηση αν η συνθήκη ισχύει. Επειδή όμως αυτή θα είναι μετά το **endswitch**, συνενώνεται με τη λίστα **exitlist**. Αν η συνθήκη δεν ισχύει συμπληρώνεται η λίστα **R\_false** με τον αριθμό της επόμενης τετράδας. Η **exitlist** συμπληρώνεται όταν εντοπιστεί **endswitch**.

➤ forcase\_stat:

Η συνάρτηση αυτή αποθηκεύει στη μεταβλητή **firstquad** τον αριθμό της επόμενης τετράδας, ώστε να γνωρίζει που να επιστρέψει αν κάποια / κάποιες από τις συνθήκες ισχύουν και δημιουργεί μία προσωρινή μεταβλητή **flag** με την τιμή **0**, η οποία χρειάζεται για τη περίπτωση που ισχύει κάποια συνθήκη και πρέπει να επαναληφθεί η διαδικασία. Έπειτα, για κάθε **when** που υπάρχει, παίρνει από τη συνάρτηση **condition 2** λίστες. Η μία λίστα είναι η **COND\_true** και αφορά την περίπτωση που η συνθήκη ισχύει. Συμπληρώνεται πριν την κλήση της συνάρτησης **statements**. Σ' αυτή την περίπτωση αλλάζει και η τιμή της προσωρινής

μεταβλητής `flag` σε `1` και συμπληρώνεται η λίστα `COND_false`, που είναι η άλλη λίστα και αφορά την περίπτωση που δεν ισχύει η συνθήκη. Όταν βρεθεί `endforcase`, αν η τιμή της `flag` είναι `1`, επιστρέφει εκεί που δείχνει η μεταβλητή `firstquad`, αλλιώς συνεχίζει με την επόμενη τετράδα μετά το `endforcase`.

- `call_stat`: Η συνάρτηση αυτή αφορά την κλήση μιας διαδικασίας. Αποθηκεύει το όνομα της διαδικασίας στη μεταβλητή `NAME`, το περνάει ως όρισμα στη συνάρτηση `actualpars` και δημιουργεί την τετράδα `call, NAME, _, _`.
- `actualpars`: Η συνάρτηση αυτή παίρνει ως όρισμα το `id_name` και αν η τιμή της `is_function` είναι `True`, δημιουργεί μία προσωρινή μεταβλητή `return_value`, δημιουργεί την τετράδα `par, return_value, ret, _` και την τετράδα `call, id_name, _, _`, για την κλήση της συνάρτησης. Τέλος, αλλάζει την τιμή της μεταβλητής `is_function` σε `False` και επιστρέφει το `return_value`.
- `actualparitem`: Η συνάρτηση αυτή, αν εντοπίσει `in`, αποθηκεύει αυτό που επιστρέφει η συνάρτηση `expression` στη μεταβλητή `EXP` και δημιουργεί την τετράδα `par, EXP, in, _`. Αν πάλι εντοπίσει `inout`, αποθηκεύει το όνομα της μεταβλητής που βρίσκεται μετά το `inout` στη μεταβλητή `NAME` και δημιουργεί την τετράδα `par, NAME, inout, _`.
- `return_stat`: Η συνάρτηση αυτή αποθηκεύει αυτό που επιστρέφει η συνάρτηση `expression` στη μεταβλητή `E_place`, δημιουργεί την τετράδα `ret, E_place, _, _` και επιστρέφει το `E_place`.
- `print_stat`: Η συνάρτηση αυτή αποθηκεύει αυτό που επιστρέφει η συνάρτηση `expression` στη μεταβλητή `E_place` και δημιουργεί την τετράδα `out, E_place, _, _`.
- `input_stat`: Η συνάρτηση αυτή αποθηκεύει την τιμή που βρίσκεται μετά το `input` στη μεταβλητή `id_place` και δημιουργεί

την τετράδα `inp, id_place, _, _`.

- condition: Η συνάρτηση αυτή παίρνει από τη συνάρτηση `boolterm` 2 λίστες, την `B_true` και την `B_false` και τις επιστρέφει. Αν συναντήσει `or` και όσο εντοπίζει `or`, συμπληρώνει τη `B_false` με τον αριθμό της επόμενης τετράδας. Έπειτα, παίρνει ξανά 2 λίστες από τη συνάρτηση `boolterm`, συνενώνει την παλιά `B_true` με τη νέα `B_true` και στο τέλος επιστρέφει τελικά τη `B_true` και τη νέα `B_false`.
- boolterm: Η συνάρτηση αυτή παίρνει από τη συνάρτηση `boolfactor` 2 λίστες, την `Q_true` και την `Q_false` και τις επιστρέφει. Αν συναντήσει `and` και όσο εντοπίζει `and`, συμπληρώνει την `Q_true` με τον αριθμό της επόμενης τετράδας. Έπειτα, παίρνει ξανά 2 λίστες από τη συνάρτηση `boolfactor`, συνενώνει την παλιά `Q_false` με τη νέα `Q_false` και στο τέλος επιστρέφει τελικά τη νέα `B_true` και τη `B_false`.
- boolfactor: Η συνάρτηση αυτή αν συναντήσει `not` ή άνοιγμα αγκύλης παίρνει από τη συνάρτηση `condition` 2 λίστες, την `R_true` και την `R_false` και τις επιστρέφει. Αν συναντήσει `+`, `-`, ψηφίο, μεταβλητή, ή άνοιγμα παρένθεσης, αποθηκεύει αυτό που επιστρέφει η συνάρτηση `expression` στη μεταβλητή `E1_place`, αποθηκεύει αυτό που επιστρέφει η συνάρτηση `relational_oper` στη μεταβλητή `RELOP` και αυτό που επιστρέφει η συνάρτηση `expression` στη μεταβλητή `E2_place`. Δημιουργεί, στη συνέχεια, τη λίστα `R_true` με τον αριθμό της επόμενης τετράδας και δημιουργεί την τετράδα `RELOP, E1_place, E2_place, _`. Έπειτα, δημιουργεί τη λίστα `R_false` με τον αριθμό της επόμενης τετράδας και δημιουργεί την τετράδα `jump, _, _, _`. Αν τέλος, συναντήσει `true`, δημιουργεί μία λίστα `R_true` με τον αριθμό της επόμενης τετράδας, δημιουργεί την τετράδα `jump, _, _, _` και επιστρέφει την `R_true`, ενώ αν συναντήσει `false`, δημιουργεί μία λίστα `R_false` με τον αριθμό της επόμενης τετράδας, δημιουργεί την τετράδα `jump, _, _, _` και επιστρέφει την

R\_false.

- expression: Η συνάρτηση αυτή αποθηκεύει στη μεταβλητή **SIGN** αυτό που επιστρέφει η συνάρτηση **optional\_sign**. Αν αυτό είναι -, τότε αποθηκεύει στη μεταβλητή **T1\_place** – συν αυτό που επιστρέφει η συνάρτηση **term**, αλλιώς αποθηκεύει στη μεταβλητή **T1\_place** μόνο αυτό που επιστρέφει η συνάρτηση **term**. Έπειτα, για όσο συναντάει + ή –, αποθηκεύει στη μεταβλητή **SIGN** αυτό που επιστρέφει η συνάρτηση **add\_oper**, αποθηκεύει στη μεταβλητή **T2\_place** αυτό που επιστρέφει η συνάρτηση **term**, δημιουργεί μία καινούρια προσωρινή μεταβλητή **w**, την τετράδα **SIGN, T1\_place, T2\_place, w** και αποθηκεύει το **w** στο **T1\_place**. Τέλος, αποθηκεύει το τελικό **T1\_place** στη μεταβλητή **E\_place** και επιστρέφει το **E\_place**.
- term: Η συνάρτηση αυτή αποθηκεύει αυτό που επιστρέφει η συνάρτηση **factor** στο **F1\_place**. Στη συνέχεια, όσο συναντάει \* ή /, αποθηκεύει αυτό που επιστρέφει η συνάρτηση **mul\_oper** στη μεταβλητή **OPER**, αποθηκεύει αυτό που επιστρέφει η συνάρτηση **factor** στη μεταβλητή **F2\_place**, δημιουργεί μία προσωρινή μεταβλητή **w**, δημιουργεί την τετράδα **OPER, F1\_place, F2\_place, w** και αποθηκεύει το **w** στο **F1\_place**. Τέλος, αποθηκεύει το τελικό **F1\_place** στη μεταβλητή **T\_place** και επιστρέφει το **T\_place**.
- factor: Η δημιουργεί μία κενή μεταβλητή **F\_place** και την επιστρέφει. Αν συναντήσει ψηφίο, αποθηκεύει στην **F\_place** την τιμή του ψηφίου και την επιστρέφει. Αν συναντήσει άνοιγμα παρένθεσης, αποθηκεύει στην μεταβλητή **E\_place** αυτό που επιστρέφει η συνάρτηση **expression**, αποθηκεύει στην **F\_place** την **E\_place** και επιστρέφει την **F\_place**. Τέλος, αν συναντήσει **id**, αποθηκεύει το όνομά της στη μεταβλητή **id\_name** και αποθηκεύει στη μεταβλητή **return\_value** αυτό που επιστρέφει η συνάρτηση **idtail**. Αν αυτό είναι **IS\_ID**, δηλαδή μεταβλητή ή σταθερά, αποθηκεύει στην **F\_place** το **id\_name**, αλλιώς αποθηκεύει στην **F\_place**

το `return_value` και σε κάθε περίπτωση επιστρέφει το `F_place`.

- `idtail`: Η συνάρτηση αυτή αν συναντήσει `άνοιγμα παρένθεσης`, αλλάζει την τιμή της μεταβλητής `is_function` σε `True`, αποθηκεύει αυτό που επιστρέφει η συνάρτηση `actualpars` στη μεταβλητή `return_value` και επιστρέφει το `return_value`. Σε κάθε άλλη περίπτωση επιστρέφει `IS_ID`.
- `relational_oper`: Η συνάρτηση αυτή αν συναντήσει `=` ή `<=` ή `>=` ή `>` ή `<` ή `<>`, το αποθηκεύει στη μεταβλητή `RELOP` και επιστρέφει το `RELOP`.
- `add_oper`: Η συνάρτηση αυτή αν συναντήσει `+` ή `-`, το αποθηκεύει στη μεταβλητή `SIGN` και επιστρέφει το `SIGN`.
- `mul_oper`: Η συνάρτηση αυτή αν συναντήσει `*` ή `/`, το αποθηκεύει στη μεταβλητή `OPER` και επιστρέφει το `OPER`.
- `optional_sign`: Η συνάρτηση αυτή αν συναντήσει `+` ή `-` το αποθηκεύει στη μεταβλητή `SIGN` και το επιστρέφει. Διαφορετικά, επιστρέφει `κενό`.

## 5.5 Αποθήκευση ενδιάμεσου κώδικα σε αρχείο

Αφού παραχθεί ο ενδιάμεσος κώδικας, στη συνέχεια αποθηκεύεται σε ένα **int αρχείο**. Αυτό επιτυγχάνεται με τη βοήθεια της συνάρτησης **productIntFile**. Πιο συγκεκριμένα, η συνάρτηση αυτή παίρνει το όνομα του προγράμματος που μεταγλωττίζουμε, δημιουργεί ένα `.int` αρχείο με το ίδιο όνομα, και γράφει σε αυτό το αρχείο όλες τις τετράδες που έχουν δημιουργηθεί από την παραγωγή ενδιάμεσου κώδικα και βρίσκονται στη λίστα `quadsList`. Τέλος, κλείνει το αρχείο.

## 5.6 Μετατροπή ενδιάμεσου κώδικα σε γλώσσα C και αποθήκευση σε αρχείο

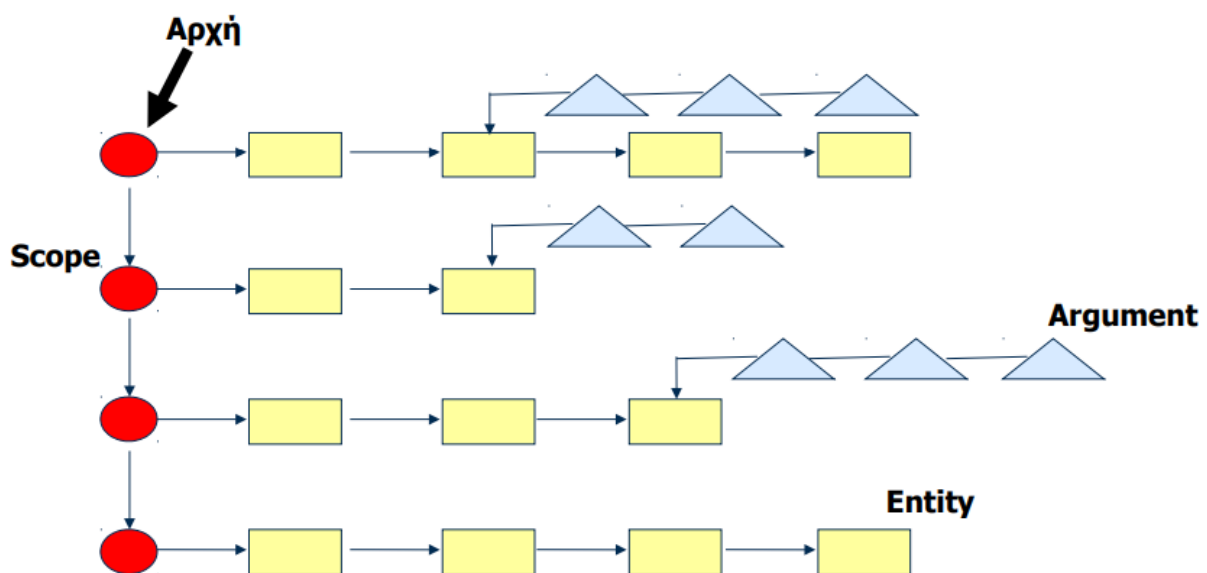
Ο ενδιάμεσος κώδικας που παράγεται μέσω της συνάρτησης **productCFile** μετατρέπεται σε **γλώσσα C** και αποθηκεύεται σε ένα **c αρχείο**. Πιο συγκεκριμένα, η συνάρτηση παίρνει το όνομα του προγράμματος που μεταγλωττίζεται, δημιουργεί ένα .c αρχείο και σε αυτό γράφει μία συνάρτηση **main**, που περιέχει:

- Δήλωση όλων των μεταβλητών που χρησιμοποιούνται
- Μία κενή γραμμή με το Label 0
- Τις τετράδες μεταποιημένες σε εντολές σε γλώσσα C,  
πχ. Η τετράδα :=, 1,\_,a γράφεται ως a := 1; ή η τετράδα jump,\_,\_,  
16, γράφεται ως go\_to L\_16;  
(Όλες οι εντολές σε γλώσσα C είναι αριθμημένες με ένα Label.)
- Μία γραμμή με το τελευταίο Label και περιεχόμενο {}.

## 6. ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

### 6.1 Ο ορισμός του πίνακα συμβόλων


Ο πίνακας συμβόλων είναι μία δυναμική δομή, η οποία περιέχει πληροφορίες μεταβλητών, υποπρογραμμάτων, παραμέτρων υποπρογραμμάτων και τύπων δεδομένων, που εμφανίζονται μέσα στο πρόγραμμα.



“Μορφή του πίνακα συμβόλων”

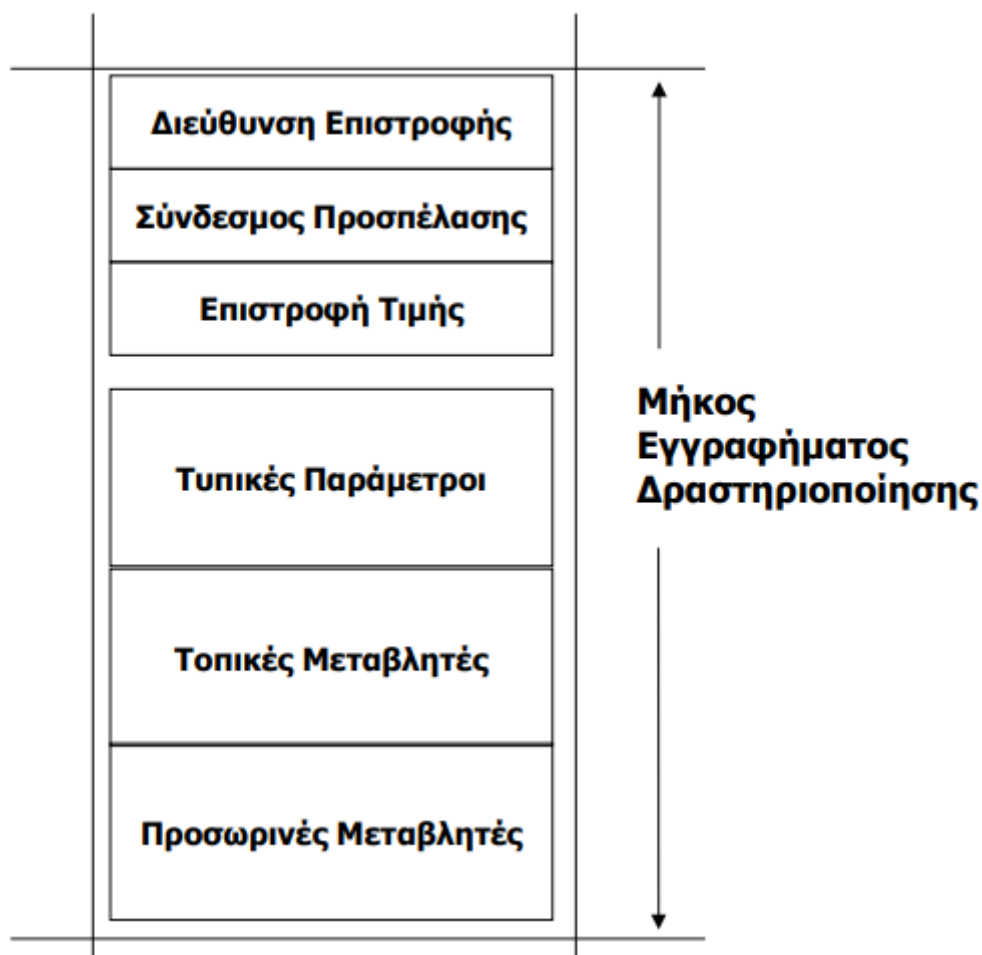
  
Scope

  
Entity

  
Argument



## 6.2 Εγγραφήμα Δραστηριοποίησης



- **Διεύθυνση Επιστροφής:** η διεύθυνση στην οποία θα μεταβεί η ροή του προγράμματος όταν ολοκληρωθεί η εκτέλεση της συνάρτησης.
- **Σύνδεσμος Προσπέλασης:** δείχνει σε ποιο σημείο του εγγραφήματος δραστηριοποίησης πρέπει να αναζητηθούν μεταβλητές, οι οποίες δεν είναι τοπικές, αλλά η συνάρτηση έχει δικαίωμα να χρησιμοποιήσει.
- **Επιστροφή Τιμής:** η διεύθυνση στην οποία θα γραφεί το αποτέλεσμα της συνάρτησης όταν αυτό υπολογιστεί.
- **Τυπικές Παράμετροι:** χώρος αποθήκευσης παραμέτρων συνάρτησης
  - αποθηκεύεται η τιμή, αν πρόκειται για πέρασμα με τιμή
  - αποθηκεύεται η διεύθυνση, αν πρόκειται για πέρασμα με αναφορά

- **Τοπικές Μεταβλητές:** χώρος αποθήκευσης τοπικών μεταβλητών.
- **Προσωρινές Μεταβλητές:** χώρος αποθήκευσης προσωρινών μεταβλητών.

### 6.3 Βοηθητικές υπορουτίνες για τον πίνακα συμβόλων

Για τη δημιουργία του πίνακα συμβόλων υλοποιήσαμε και χρησιμοποιήσαμε τις παρακάτω υπορουτίνες:

- addScope(): Δημιουργεί ένα νέο `scope` και το προσθέτει στη λίστα με τα `scope`. Το νέο `scope` που δημιουργείται έχει δομή λεξικού και περιέχει μια `λίστα από entities`, το βάθος φωλιάσμάτος του – `nestingLevel` και το `current_offset`, δηλαδή την απόστασή του από την κορυφή της στοίβας.
- removeScope(): Διαγράφει ένα `scope` από τη λίστα με τα `scopes`. Όταν τελειώσει η μετάφραση μιας συνάρτησης διαγράφεται η εγγραφή του `scope`. Οι συναρτήσεις μπορούν να χρησιμοποιηθούν μόνο αφού έχει ολοκληρωθεί η μετάφρασή τους, δηλαδή δεν επιτρέπονται χρήσεις αναδρομής. Γι' αυτό το λόγο έχει προστεθεί ένα `flag READY` σε κάθε συνάρτηση, που δηλώνει αν έχει τελειώσει η μετάφρασή της.
- addEntity(): Δημιουργεί ένα νέο `entity` και το προσθέτει στη λίστα των `entities` του τρέχοντος `scope`. Αυτό συμβαίνει όταν εντοπιστεί δήλωση μεταβλητής, δημιουργία νέας προσωρινής μεταβλητής, δήλωση νέας συνάρτησης και δήλωση τυπικής παραμέτρου συνάρτησης. Πιο συγκεκριμένα, η συνάρτηση `addEntity` παίρνει ως όρισμα το όνομα του `entity`, `entity_name`, τον τύπο του, `entity_type` και το βοηθητικό όρισμα `par3`. Κάθε `entity` που δημιουργείται έχει δομή τύπου λεξικού και

περιέχει το **entity\_name** και το **entity\_type**. Ιδιαίτερα, αν ο τύπος του είναι **variable**, περιέχει επιπλέον το **offset**, αν είναι **procorfunc**, περιέχει επιπλέον χαρακτηρισμό **function** εάν είναι συνάρτηση, **procedure** αν είναι διαδικασία, μια **λίστα από arguments**, την ετικέτα της πρώτης τετράδας της - **start\_quad**, το μήκος του εγγραφήματος δραστηριοποίησής της - **frame\_length** και το **flag READY**. Αν πάλι ο τύπος του entity είναι **parameter**, περιέχει επιπλέον το **offset** και το **parmode**, το οποίο είναι **in** αν αφορά πέρασμα με τιμή και **inout** αν αφορά πέρασμα με αναφορά. Τέλος, αν ο τύπος του entity είναι **tempvar**, τότε αυτό περιέχει επιπλέον το **offset**.

- addArgument(): Δημιουργεί και προσθέτει ένα νέο **argument** στη λίστα με τα arguments του τελευταίου entity του προηγούμενου scope. Αυτό συμβαίνει όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης. Το νέο argument που δημιουργείται έχει δομή λεξικού και περιέχει το **parmode**, δηλαδή **in** για πέρασμα με τιμή και **inout** για πέρασμα με αναφορά.
- searchEntity(): Εκτελεί την αναζήτηση ενός entity με βάση το όνομά του. Πιο συγκεκριμένα, ξεκινάει την αναζήτηση από την αρχή – πρώτη γραμμή του πίνακα και αν το entity δεν βρεθεί σε αυτήν, εκτελεί αναζήτηση στην αμέσως επόμενη γραμμή. Επιστρέφει το πρώτο entity με το ζητούμενο όνομα που θα συναντήσει, ενώ αν δεν εντοπίσει κάποιο entity με αυτό το όνομα, επιστρέφει μήνυμα λάθους. Τέλος, σημαντικό είναι το γεγονός ότι εκτός από το όνομα ελέγχει αν ο τύπος του entity που βρέθηκε είναι ίδιος με το ζητούμενο.
- calculateFL(): Χρησιμοποιείται για τον υπολογισμό του μήκους εγγραφήματος δραστηριοποίησης - **framelength** μιας συνάρτησης. Για την περίπτωση του

υπολογισμού του `framelength` του κύριου προγράμματος, γίνεται έλεγχος αν έχει μείνει ένα μόνο `score` στη λίστα με τα `scores` και το `framelength` του αποθηκεύεται σε μία κοινόχρηστη δομή λεξικού `main_scope`, η οποία έχει δημιουργηθεί γι' αυτό το σκοπό. Σε κάθε άλλη περίπτωση το `frame length` αποθηκεύεται στο τελευταίο `entity` του προηγούμενου `score`.

- `setSQ()`: Παίρνει ως όρισμα την ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης – `start_quad` και το αποθηκεύει στο τελευταίο `entity` του προηγούμενου `score`.

## 6.4 Επεξήγηση του κώδικα που αφορά τον πίνακα συμβόλων

Για τη δημιουργία του πίνακα συμβόλων χρησιμοποιήσαμε εκτός από τις υπορουτίνες που αναφέραμε πριν, τα εξής:

- `scores`: Μία κοινόχρηστη λίστα για την αποθήκευση των `scores`.
- `main_scope`: Ένα κοινόχρηστο λεξικό για την αποθήκευση του `frame_length` και του `start_quad` του κύριου προγράμματος.

Με τη βοήθεια όλων των παραπάνω προσθέσαμε κάποια κομμάτια κώδικα σε κάποιες από τις συναρτήσεις του συντακτικού αναλυτή που ήδη είχαμε υλοποιήσει. Οι συναρτήσεις που τροποποιήσαμε και η λειτουργία που προσθέσαμε σε αυτές φαίνεται αναλυτικά παρακάτω:

- `program`: Η συνάρτηση αυτή όταν εντοπίσει `program` και `id` καλεί τη συνάρτηση `addScope` για τη δημιουργία ενός νέου `score` που αφορά το κύριο πρόγραμμα.

- block:

Η συνάρτηση αυτή ελέγχει αν το όνομα που παίρνει ως όρισμα είναι ίδιο με το όνομα του κύριου προγράμματος. Αν είναι, αποθηκεύει το `start_quad` του στο `main_scope`. Διαφορετικά, αφορά κάποιο υποπρόγραμμα, οπότε καλεί τη συνάρτηση `setSQ` για να αποθηκεύσει το `start_quad` του. Όταν τελειώσει τον πρόγραμμα/ υποπρόγραμμα καλεί τη συνάρτηση `calculateFL` για να υπολογίσει το `frame_length` και τη `removeScope` για να διαγράψει το συγκεκριμένο `scope`.
- varlist:

Αν εντοπίσει `id` και για κάθε `id` που υπάρχει στη συνέχεια, καλεί την `addEntity` με την τιμή του `id`, για να δημιουργήσει ένα καινούριο `entity` γι' αυτό.
- procorfunc:

Η συνάρτηση αυτή και στην περίπτωση διαδικασίας και στην περίπτωση συνάρτησης καλεί την `addEntity` για δημιουργία νέου `entity` και στη συνέχεια την `addScope` για δημιουργία νέου `scope`.
- formalparitem:

Η συνάρτηση αυτή αν εντοπίσει `in` ή `inout`, το αποθηκεύει στη μεταβλητή `PARMODE` και καλεί τη συνάρτηση `addArgument` με όρισμα το `PARMODE`. Στη συνέχεια, μόλις βρει `id` παίρνει την τιμή του και καλεί τη συνάρτηση `addEntity` για να δημιουργήσει ένα καινούριο `entity` γι' αυτό.

## 8. ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΛΥΣΗ

### 7.1 Απαιτήσεις που αφορούν τη σημασιολογική ανάλυση

Για την υλοποίηση της σημασιολογικής ανάλυσης ήταν απαραίτητη η ικανοποίηση των παρακάτω απαιτήσεων:

- ✓ Κάθε συνάρτηση να περιέχει μέσα της τουλάχιστον ένα return
- ✓ Να μην υπάρχει return οπουδήποτε αλλού, εκτός από συνάρτηση
- ✓ Να υπάρχει exit μόνο σε δομή επανάληψης repeat
- ✓ Κάθε μεταβλητή, συνάρτηση ή διαδικασία να μην έχει δηλωθεί πάνω από μία φορά στο βάθος φωλιάσματος στο οποίο βρίσκεται
- ✓ Κάθε μεταβλητή, συνάρτηση ή διαδικασία να δηλώνεται και να χρησιμοποιείται με τον ίδιο τρόπο
- ✓ Οι παράμετροι με τις οποίες καλούνται οι συναρτήσεις και οι διαδικασίες να είναι όπως έχουν δηλωθεί και με την ίδια σειρά

### 7.2 Βοηθητικές υπορουτίνες για τη σημασιολογική ανάλυση

Για τη σημασιολογική ανάλυση υλοποιήσαμε και χρησιμοποιήσαμε τις παρακάτω υπορουτίνες:

- checkEntityExists(): Η συνάρτηση αυτή παίρνει ως όρισμα το **όνομα** ενός entity και εκτελεί αναζήτηση στο τρέχον scope του πίνακα συμβόλων. Εάν εντοπίσει κάποιο entity με το ίδιο όνομα, επιστρέφει μήνυμα λάθους.
- checkArguments(): Η συνάρτηση αυτή παίρνει ως όρισμα το **όνομα** μιας συνάρτησης ή μιας διαδικασίας και τη **λίστα των παραμέτρων της**. Στη συνέχεια, χρησιμοποιεί

τη συνάρτηση `searchEntity`, για να πάρει το `entity` της. Γι' αυτό το σκοπό έχει προστεθεί μία επιπλέον λειτουργία στη συνάρτηση `searchEntity`, η οποία εκτελείται όταν δοθεί ως όρισμα το όνομα μιας συνάρτησης ή διαδικασίας και ο όρος "PROCORFUNC". Σ' αυτή την περίπτωση η `searchEntity` επιστρέφει το `entity` της συνάρτησης ή διαδικασίας που ζητείται. Τέλος, η συνάρτηση `checkArguments`, ελέγχει αν τα ορίσματα στη λίστα παραμέτρων είναι **ίδια** με αυτά που έχουν δηλωθεί, και αν επιπλέον έχουν το **ίδιο πλήθος** και την **ίδια σειρά**.

### 7.3 Επεξήγηση του κώδικα που αφορά τη σημασιολογική ανάλυση

Για τη σημασιολογική ανάλυση χρησιμοποιήσαμε εκτός από τις υπορουτίνες που αναφέραμε πριν, τα εξής:

- `needsReturn`: Μία κοινόχρηστη λίστα, η οποία περιέχει, με τη σειρά που μεταγλωττίζονται το κυρίως πρόγραμμα, οι συναρτήσεις και οι διαδικασίες, **true** και **false** ανάλογα με το αν επιτρέπεται ή δεν επιτρέπεται να υπάρχει **return** μέσα τους.
- `hasReturn`: Μία κοινόχρηστη λίστα, η οποία περιέχει, με τη σειρά που μεταγλωττίζονται το κυρίως πρόγραμμα, οι συναρτήσεις και οι διαδικασίες, **true** και **false** ανάλογα με το αν εντοπίστηκε ή όχι **return** σε αυτά.
- `repeatChecker`: Μία κοινόχρηστη λίστα, η οποία χρησιμοποιείται για το αν τα `exit` στο πρόγραμμα βρίσκονται μόνο μέσα σε δομές επανάληψης `repeat`. Συγκεκριμένα, αποθηκεύει ένα στοιχείο **false** κάθε φορά που βρίσκει ένα `repeat`, κάνει **true** το τελευταίο στοιχείο κάθε φορά που βρίσκει ένα `exit` και

διαγράφει το τελευταίο στοιχείο αν αυτό είναι true και έχει τελειώσει το repeat. Αν διαγραφούν όλα τα στοιχεία σημαίνει ότι όλα τα exit αντιστοιχούν σε κάποιο repeat. Σε κάθε άλλη περίπτωση εμφανίζεται μήνυμα λάθους.

Με τη βοήθεια όλων των παραπάνω προσθέσαμε κάποια κομμάτια κώδικα σε κάποιες από τις συναρτήσεις του συντακτικού αναλυτή που ήδη είχαμε υλοποιήσει. Οι συναρτήσεις που τροποποιήσαμε και η λειτουργία που προσθέσαμε σε αυτές φαίνεται αναλυτικά παρακάτω:

- program: Η συνάρτηση αυτή αφορά το κυρίως πρόγραμμα, οπότε προσθέτει στις λίστες `needsReturn` και `hasReturn` ένα στοιχείο `false`.
- block: Η συνάρτηση αυτή ελέγχει αν το τελευταίο στοιχείο της λίστας `needsReturn` είναι ίδιο με το τελευταίο στοιχείο της λίστας `hasReturn`. Αν είναι το `εξάγει` από την κάθε λίστα, διαφορετικά εμφανίζει μήνυμα λάθους.
- varlist: Η συνάρτηση αυτή όσο εντοπίζει `id`, παίρνει την τιμή του και μέσω της συνάρτησης `checkEntityExists` ελέγχει αν αυτό ήδη υπάρχει.
- procorfunc: Η συνάρτηση αυτή προσθέτει ένα στοιχείο `false` στη λίστα `needsReturn` και ένα στοιχείο `false` στη λίστα `hasReturn` σε περίπτωση διαδικασίας, ενώ σε περίπτωση συνάρτησης προσθέτει ένα στοιχείο `true` στη λίστα `needsReturn` και ένα στοιχείο `false` στη λίστα `hasReturn`, το οποίο θα γίνει true εάν βρεθεί στη συνάρτηση `return`. Επιπλέον μέσω της συνάρτησης `checkEntityExists` ελέγχεται αν το όνομα της συνάρτησης/ διαδικασίας υπάρχει ήδη.
- formalparitem: Η συνάρτηση αυτή εάν εντοπίσει `in` και `id` ή `inout` και `id`, παίρνει την τιμή του `id` και μέσω της συνάρτησης `checkEntityExists` ελέγχει αν αυτό υπάρχει ήδη.



- repeat\_stat: Η συνάρτηση αυτή όταν εντοπίζει **repeat** προσθέτει ένα στοιχείο **false** στη λίστα **repeatChecker**. Επίσης, όταν εντοπίζει **endrepeat** ελέγχει αν το τελευταίο στοιχείο είναι **true**, δηλαδή αν έχει βρεθεί **exit** μέσα στη **repeat**. Αν είναι, το διαγράφει, αλλιώς εμφανίζει μήνυμα λάθους.
- exit\_stat: Η συνάρτηση αυτή αφορά την περίπτωση που μέσα στο πρόγραμμα εντοπίζεται **exit**. Οπότε ελέγχει αν το μήκος της **repeatChecker** είναι μεγαλύτερο του μηδενός και αν είναι αλλάζει την τιμή του τελευταίου στοιχείου σε **true**. Διαφορετικά, το **exit** δεν βρίσκεται μέσα σε δομή επανάληψης **repeat**, οπότε εμφανίζει μήνυμα λάθους.
- call\_stat: Η συνάρτηση αυτή αν εντοπίζει **call** και **id**, παίρνει την τιμή του **id** και μέσω της συνάρτησης **searchEntity** ελέγχει αν αυτό υπάρχει.  
(Η **searchEntity** ορίζεται στην ενότητα 6.3)
- actualparlist: Η συνάρτηση αυτή κάθε φορά που εντοπίζει μία **παράμετρο** συνάρτησης ή διαδικασίας, προσθέτει το **parmode** της - **in** ή **inout** - σε μία λίστα **PARS** και στο τέλος, καλεί τη συνάρτηση **ckeckArguments** με όρισμα το **όνομα** της συνάρτησης/ διαδικασίας και τη λίστα **PARS**. Το **parmode** της κάθε παραμέτρου το παίρνει από τη συνάρτηση **actualparitem**.
- actualparitem: Η συνάρτηση αυτή επιστρέφει για κάθε παράμετρο το **parmode** της, **in** ή **inout**.
- return\_stat: Η συνάρτηση αυτή αν εντοπίζει **return**, αλλάζει την τιμή του τελευταίου στοιχείου της λίστας **hasReturn** σε **true**.
- input\_stat: Η συνάρτηση αυτή αν εντοπίζει **input** και **id**, παίρνει την τιμή του **id** και μέσω της συνάρτησης

`searchEntity` ελέγχει αν αυτό υπάρχει.  
(Η `searchEntity` ορίζεται στην ενότητα 6.3)

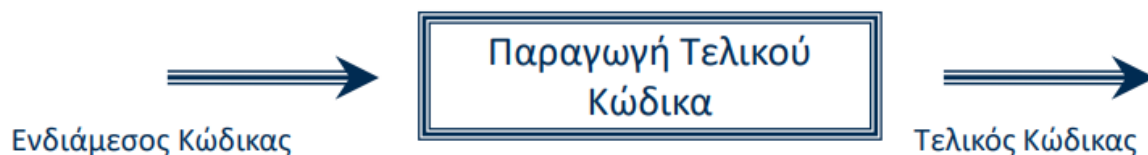
➤ factor:

Η συνάρτηση αυτή αν εντοπίσει `id`, παίρνει την τιμή του `id` και μέσω της συνάρτησης `searchEntity` ελέγχει αν αυτό υπάρχει.  
(Η `searchEntity` ορίζεται στην ενότητα 6.3)

## 8. ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

### 8.1 Βασικά χαρακτηριστικά τελικού κώδικα

Ο τελικός κώδικας παράγεται από τις εντολές του ενδιάμεσου κώδικα. Κύριες ενέργειες σ' αυτή τη φάση είναι ότι οι μεταβλητές απεικονίζονται στην μνήμη (στοίβα), το πέρασμα παραμέτρων και η κλήση συναρτήσεων. Τελικά δημιουργείται κώδικας για τον επεξεργαστή MIPS.



### 8.2 Αρχιτεκτονική MIPS

➤ Χρήσιμοι καταχωρητές:

- 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

➤ Χρήσιμες εντολές 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

➤ Χρήσιμες εντολές 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

➤ Χρήσιμες εντολές 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<=\$t2
- bge \$t1,\$t2,label              jump to label if \$t1>=\$t2
- bne \$t1,\$t2,label              jump to label if \$t1<>\$t2

➤ Χρήσιμες εντολές για 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 τη διεύθυνση επιστροφής συνάρτησης)

### 8.3 Βοηθητικές συναρτήσεις για τον τελικό κώδικα

Για τη δημιουργία του τελικού κώδικα υλοποιήσαμε και χρησιμοποιήσαμε τις παρακάτω βοηθητικές συναρτήσεις:

- addToASM(): Η συνάρτηση αυτή γράφει στο **.asm αρχείο**, που αφορά τον τελικό κώδικα, αυτό που της δίνεται ως **όρισμα**.
- getScopeDistance(): Η συνάρτηση αυτή παίρνει ως όρισμα μία μεταβλητή - **variable**, και ξεκινώντας από το τελευταίο scope στον πίνακα συμβόλων αναζητά τη μεταβλητή αυτή. Για κάθε scope που συναντά αυξάνει ένα μετρητή **distance**. Αυτός ο μετρητής δείχνει ουσιαστικά την απόσταση της μεταβλητής από το τρέχον scope. Τέλος, η συνάρτηση επιστρέφει αυτό το μετρητή.
- getEntityField(): Η συνάρτηση αυτή παίρνει ως όρισμα το όνομα ενός entity – **entity\_name**, και ένα πεδίο - **field** και επιστρέφει το αντίστοιχο πεδίο για το συγκεκριμένο entity. Εκτός από τα πεδία ενός entity μπορεί να επιστρέψει ένα **ολόκληρο entity** ή το **scope** στο οποίο βρίσκεται το entity.
- gnlvcode(): Η συνάρτηση αυτή μεταφέρει στον καταχωρητή **\$t0**, τη μη τοπική μεταβλητή – **v** που της δίνεται ως όρισμα. Συγκεκριμένα, από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μεταβλητή και μέσα από το σύνδεσμο προσπέλασης την εντοπίζει. Αρχικά γράφει στο αρχείο την εντολή **lw \$t0, -4(\$sp)**. Έπειτα μέχρι να φτάσει στο scope που βρίσκεται η μεταβλητή, γράφει στο αρχείο την εντολή **lw \$t0, -4(\$t0)**. Τέλος, παίρνει το **offset** της μεταβλητής που της δίνεται ως όρισμα, το αποθηκεύει στην **entityOffset** και γράφει στο αρχείο την εντολή **addi \$t0, \$t0, - entityOffset**.
- checkVariableType(): Η συνάρτηση αυτή παίρνει ως όρισμα μία μεταβλητή – **variable** και χρησιμοποιείται στις

συναρτήσεις `loadvr` και `storerv`, που περιγράφονται παρακάτω. Αν η μεταβλητή που παίρνει ως όρισμα είναι ψηφίο, επιστρέφει `immediate`. Αν η μεταβλητή έχει βάθος φωλιάσματος μηδέν, ανήκει στο κυρίως πρόγραμμα, και τότε η συνάρτηση επιστρέφει `global`. Αν ο τύπος της μεταβλητής είναι `variable`, και ανήκει στο τρέχον `scope`, επιστρέφει `varCurr`, ενώ αν δεν ανήκει στο τρέχον `scope`, επιστρέφει `varNoCurr`. Αν ο τύπος της μεταβλητής είναι `parameter`, έχει `parmode in` και ανήκει στο τρέχον `scope`, επιστρέφει `parInCurr`, ενώ αν δεν ανήκει στο τρέχον `scope`, επιστρέφει `parInNoCurr`. Αν ο τύπος της μεταβλητής είναι `parameter`, έχει `parmode inout` και ανήκει στο τρέχον `scope`, επιστρέφει `parInoutCurr`, ενώ αν δεν ανήκει στο τρέχον `scope`, επιστρέφει `parInoutNoCurr`. Τέλος, αν ο τύπος είναι `tempvar`, επιστρέφει `tempvar`.

➤ `loadvr()`:

Η συνάρτηση αυτή παίρνει ως όρισμα μια μεταβλητή `v` και έναν καταχωρητή `r`, και μεταφέρει τα δεδομένα από την `v` στον `r`. Συγκεκριμένα, καλεί τη βοηθητική συνάρτηση `checkVariableType` και αποθηκεύει το αποτέλεσμα της σε μια μεταβλητή – `vType`. Στη συνέχεια, με βάση την τιμή της μεταβλητής, αν είναι `immediate` γράφει στο αρχείο `li $tr,v`. Αν είναι `global` γράφει `lw $tr,-offset($s0)`. Αν είναι `varCur`, `parInCurr` ή `tempVar` γράφει `lw $tr,-offset($sp)`. Αν είναι `parInoutCurr` γράφει `lw $t0,-offset($sp)` και `lw $tr,($t0)`. Αν είναι `varNoCurr` ή `parInNoCurr` καλεί την `gnlvcode()` και γράφει `lw $tr,($t0)`. Τέλος, αν είναι `parInoutNoCurr` καλεί την `gnlvcode()` και γράφει `lw $t0, ($t0)` και `lw $tr,($t0)`.

➤ `storerv()`:

Η συνάρτηση αυτή παίρνει ως όρισμα μια μεταβλητή `v` και έναν καταχωρητή `r`, και μεταφέρει τα δεδομένα από τον `r` στη `v`. Συγκεκριμένα, καλεί τη βοηθητική συνάρτηση `checkVariableType` και αποθηκεύει το

αποτέλεσμά της σε μια μεταβλητή – **vType**. Στη συνέχεια, με βάση την τιμή της μεταβλητής, αν είναι **global** γράφει **sw \$tr,-offset(\$s0)**. Αν είναι **varCur**, **parInCurr** ή **tempVar** γράφει **sw \$tr,-offset(\$sp)**. Αν είναι **parInoutCurr** γράφει **lw \$t0,-offset(\$sp)** και **sw \$tr,(\$t0)**. Αν είναι **varNoCurr** ή **parInNoCurr** καλεί την **glnvcode()** και γράφει **sw \$tr,(\$t0)**. Τέλος, αν είναι **parInoutNoCurr** καλεί την **glnvcode()** και γράφει **lw \$t0, (\$t0)** και **sw \$tr,(\$t0)**.

## 8.4 Επεξήγηση του κώδικα που αφορά την παραγωγή τελικού κώδικα

Για την παραγωγή τελικού κώδικα χρησιμοποιήσαμε εκτός από τις βοηθητικές συναρτήσεις που αναφέραμε πριν, τα εξής:

- asm\_name: Μία κοινόχρηστη μεταβλητή, που περιέχει το όνομα του προγράμματος που μεταγλωττίζεται.
- asm\_file: Μία κοινόχρηστη μεταβλητή που περιέχει το .asm αρχείο, το οποίο έχει ίδιο όνομα με το πρόγραμμα που μεταγλωττίζεται και σε αυτό θα γραφούν οι εντολές του τελικού κώδικα.
- current\_quadToASM: Μία κοινόχρηστη μεταβλητή που χρησιμοποιείται στην παραγωγή του τελικού κώδικα. Σ' αυτήν αποθηκεύεται η ετικέτα της τελευταίας τετράδας του ενδιάμεσου κώδικα που διαβάστηκε.
- pars: Μία κοινόχρηστη λίστα για την αποθήκευση παραμέτρων.

Με τη βοήθεια όλων των παραπάνω δημιουργήσαμε δύο συναρτήσεις, από τις οποίες η μία παράγει τον τελικό κώδικα – **finalCode** και η άλλη εκτελεί μια λειτουργία της πρώτης – **createCall**. Η λειτουργία τους παρουσιάζεται αναλυτικά παρακάτω:

➤ finalCode:

Η συνάρτηση αυτή, αν δεν έχει διαβαστεί καμία τετράδα του ενδιαμέσου κώδικα, γράφει στο αρχείο το **label L0**, το οποίο αποτελείται από ένα jump στη συνάρτηση **main**, για την σωστή εκκίνηση του προγράμματος. Στη συνέχεια, διαβάζει ομάδες τετράδων του ενδιαμέσου κώδικα, σύμφωνα με τον τρόπο που αυτός έχει παραχθεί και γράφει στο αρχείο ένα κομμάτι τελικού κώδικα, ανάλογα με το πρώτο στοιχείο της κάθε τετράδας. Συγκεκριμένα, η πρώτη τετράδα κάθε ομάδας είναι αυτή που βρίσκεται μετά την τελευταία τετράδα της προηγούμενης ομάδας. Για να γνωρίζει κάθε φορά την ετικέτα της πρώτης τετράδας της ομάδας, χρησιμοποιεί τη μεταβλητή **current\_quadToASM**. Ιδιαίτερη είναι η περίπτωση που το πρώτο στοιχείο της τετράδας είναι το **begin\_block**. Τότε γίνεται έλεγχος αν πρόκειται για το κυρίως πρόγραμμα, όπου προστίθεται η ετικέτα **Lmain**, στην οποία γίνεται το πρώτο jump, κατεβαίνει ο **\$sp** κατά **frameLength** της main και σημειώνεται στον **\$s0** το εγγράφημα δραστηριοποίησης της main. Σε περίπτωση που το πρώτο στοιχείο της τετράδας είναι το **par**, τότε πρόκειται για παράμετρο και αποθηκεύεται το **entity** της, μαζί με το **parmode** της στη λίστα **pars**. Σε περίπτωση που το πρώτο στοιχείο της τετράδας είναι το **call** αποθηκεύεται το **entity** το οποίο αναφέρεται, καλείται η συνάρτηση **createCall**, και τέλος, μηδενίζεται η λίστα με τις παραμέτρους **pars**.

➤ createCall:

Η συνάρτηση αυτή διαχειρίζεται τις παραμέτρους των συναρτήσεων. Συγκεκριμένα, αποθηκεύει στη μεταβλητή **LABEL** τον αριθμό ετικέτας στον οποίο θα αρχίσει να γράφει, γιατί, διαβάζοντας τις παραμέτρους, η σειρά των label είχε αλλάξει. Ξεχωρίζει η περίπτωση που γράφεται η πρώτη παράμετρος, στην οποία πρέπει να μεταφερθεί ο δείκτης στοίβας στην κληθείσα – **add \$sp, \$sp,**



`framelength`. Για τις παραμέτρους των συναρτήσεων ή διαδικασιών ανάλογα με το αν έχουν `paramode in` ή `inout`, αν έχουν ίδιο βάθος φωλιάσματος με το τρέχον, και ανάλογα με τον τύπο τους, παράγουν διαφορετικό τελικό κώδικα. Εδώ υπάρχει και η περίπτωση της μεταβλητής στην οποία θα περάσει η τιμή επιστροφής, αν έχουμε συνάρτηση – `ret`. Στη συνέχεια, ανάλογα με το `nesting level` της συνάρτησης `call` προστίθεται διαφορετικός κώδικας. Τέλος, λαμβάνεται από την πρώτη θέση του εγγραφήματος δραστηριοποίησης η διεύθυνση επιστροφής της συνάρτησης και εισάγεται πάλι στον `$ra`. Μέσω αυτού επιστρέφει στην καλούσα.

Η συνάρτηση `finalCode` καλείται στη συνάρτηση `block` του ενδιάμεσου κώδικα πριν διαγραφεί το τρέχον `scope` με τη συνάρτηση `removeScope`. Ακόμη στην συνάρτηση `finalCode` προστίθεται ως όρισμα `true` αν πρόκειται για την κύρια συνάρτηση `main`, ενώ `false` για οποιαδήποτε άλλη περίπτωση.

## 9. ΤΕΣΤ

Παρακάτω παρουσιάζονται δύο τεστ σε γλώσσα EEL, που δείχνουν τη σωστή λειτουργία του μεταγλωττιστή μας. Σε screenshot φαίνεται ο αρχικός κώδικας του καθενός, ο κώδικας στο .int αρχείο, ο κώδικας στο .c αρχείο και ο κώδικας στο .asm αρχείο.

### 9.1 ΤΕΣΤ1

Αρχικός κώδικας:

```
program example1
  declare d, i, g, f enddeclare

  procedure two (in g)
    function three (in g, inout x, inout m)
      declare k, j enddeclare
      k:=g;
      k:=g;
      repeat
        if k>i then
          exit
        endif;
        j:=j*k;
        k:=k+g;
      endrepeat;
      m:=j;
      return m+1;
      x:=7;
    endfunction
    i:=three (in i+2, inout d, inout f)
  endprocedure

  function one (in g)
    call two(in g);
    return 1;      //
  endfunction

  i:=5;
  g:=1;
  //call one (in g)
endprogram
```

Κώδικας σε .int αρχείο:

```
0:begin_block,three,_,_
1::=,g,_,k
2::=,g,_,k
3:>,k,i,5
4:jump,_,_,6
5:jump,_,_,11
6:*,j,k,T_0
7::=,T_0,_,j
8:+,k,g,T_1
9::=,T_1,_,k
10:jump,_,_,3
11::=,j,_,m
12:+,m,l,T_2
13:ret,T_2,_,_
14::=,7,_,x
15:end_block,three,_,_
16:begin_block,two,_,_
17:+,i,2,T_3
18:par,T_3,in,_
19:par,d,inout,_
20:par,f,inout,_
21:par,T_4,ret,_
22:call,three,_,_
23::=,T_4,_,i
24:end_block,two,_,_
25:begin_block,one,_,_
26:par,g,in,_
27:call,two,_,_
28:ret,l,_,_
29:end_block,one,_,_
30:begin_block,example1,_,_
31::=,5,_,i
32::=,l,_,g
33:halt,_,_,_
34:end_block,example1,_,_

```

Κώδικας σε .c αρχείο:

```
#include <stdio.h>

int main()
{
    int k,k,k,T_0,j,T_1,k,m,T_2,x,T_3,i,g;
    L_0:
    L_1: k=g;
    L_2: k=g;
    L_3: if (k>i) goto L_4;
    L_4: goto L_5;
    L_5: goto L_10;
    L_6: T_0=j*k;
    L_7: j=T_0;
    L_8: T_1=k+g;
    L_9: k=T_1;
    L_10: goto L_2;
    L_11: m=j;
    L_12: T_2=m+1;
    L_13: x=7;
    L_14: T_3=i+2;
    L_15: i=T_4;
    L_16: i=5;
    L_17: g=1;
    L_18: {}
}
```

Κώδικας σε .asm αρχείο:

```

L0:
    j Lmain
L1:
three:
    sw $ra, ($sp)
L2:
    lw $t1, -12($sp)
    sw $t1, -24($sp)
L3:
    lw $t1, -12($sp)
    sw $t1, -24($sp)
L4:
    lw $t1, -24($sp)
    lw $t2, -16($s0)
    bgt $t1, $t2, L6
L5:
    j L7
L6:
    j L12
L7:
    lw $t1, -28($sp)
    lw $t2, -24($sp)
    mul $t1, $t1, $t2
    sw $t1, -32($sp)
L8:
    lw $t1, -32($sp)
    sw $t1, -28($sp)
L9:
    lw $t1, -24($sp)
    lw $t2, -12($sp)
    add $t1, $t1, $t2
    sw $t1, -36($sp)
L10:
    lw $t1, -36($sp)
    sw $t1, -24($sp)
L11:
    j L4
L12:
    lw $t1, -28($sp)
    lw $t0, -20($sp)
    sw $t1, ($t0)
L13:
    lw $t0, -20($sp)
    lw $t1, ($t0)
    li $t2, 1
    add $t1, $t1, $t2
    sw $t1, -40($sp)
L14:
    lw $t1, -40($sp)
    lw $t0, -8($sp)
    sw $t1, ($t0)
L15:
    li $t1, 7
    lw $t0, -16($sp)
    sw $t1, ($t0)

L16:
    lw $ra, ($sp)
    jr $ra
L17:
two:
    sw $ra, ($sp)
L18:
    lw $t1, -16($s0)
    li $t2, 2
    add $t1, $t1, $t2
    sw $t1, -16($sp)
L19:
    addi $fp, $sp, 44
    lw $t0, -16($sp)
    sw $t0, -12($fp)
L20:
    lw $t0, -4($sp)
    addi $t0, $t0, -12
    sw $t0, -16($fp)
L21:
    lw $t0, -4($sp)
    addi $t0, $t0, -24
    sw $t0, -20($fp)
L22:
    addi $t0, $sp, -20
    sw $t0, -8($fp)
L23:
    lw $t0, -4($sp)
    sw $t0, -4($fp)
    addi $sp, $sp, 44
    jal three
    addi $sp, $sp, -44
L24:
    lw $t1, -20($sp)
    sw $t1, -16($s0)
L25:
    lw $ra, ($sp)
    jr $ra
L26:
one:
    sw $ra, ($sp)
L27:
    addi $fp, $sp, 24
    lw $t0, -12($sp)
    sw $t0, -12($fp)
L28:
    sw $sp, -4($fp)
    addi $sp, $sp, 24
    jal two
    addi $sp, $sp, -24
L29:
    li $t1, 1
    lw $t0, -8($sp)
    sw $t1, ($t0)

L30:
    lw $ra, ($sp)
    jr $ra
Lmain:
L31:
    addi $sp, $sp, 28
    move $s0, $sp
L32:
    li $t1, 5
    sw $t1, -16($s0)
L33:
    li $t1, 1
    sw $t1, -20($s0)
L34:

```

## 9.2 ΤΕΣΤ2

Αρχικός κώδικας:

```
program example2
  declare x,y,z enddeclare

  procedure p1(in x, inout z, inout v)
    declare w enddeclare

    procedure p2(inout z)
      declare q enddeclare

      procedure p3(inout a, inout b)
        declare k enddeclare
        if v<>0 then
          v:=z+b;
          a:=1;
        else
          a:=v/b;
        endif;
        k:=x;
      endprocedure

      q:=y+w;
      z:=q*x;
      call p3(inout q, inout v);
    endprocedure

    if x<y then
      w:=x+y;
    else
      w:=x*y;
    endif;
    call p2(inout z);
  endprocedure

  x:=1;
  y:=2;
  call p1(in x+y, inout z, inout z); //
endprogram
```

Κώδικας σε .int αρχείο:

```
0:begin_block,p3,_,_
1:<>,v,0,3
2:jump,_,_,7
3:+,z,b,T_0
4::=,T_0,_,v
5::=,l,_,a
6:jump,_,_,9
7:/,v,b,T_1
8::=,T_1,_,a
9::=,x,_,k
10:end_block,p3,_,_
11:begin_block,p2,_,_
12:+,y,w,T_2
13::=,T_2,_,q
14:*,q,x,T_3
15::=,T_3,_,z
16:par,q,inout,_
17:par,v,inout,_
18:call,p3,_,_
19:end_block,p2,_,_
20:begin_block,p1,_,_
21:<,x,y,23
22:jump,_,_,26
23:+,x,y,T_4
24::=,T_4,_,w
25:jump,_,_,28
26:*,x,y,T_5
27::=,T_5,_,w
28:par,z,inout,_
29:call,p2,_,_
30:end_block,p1,_,_
31:begin_block,example2,_,_
32::=,l,_,x
33::=,2,_,y
34:+,x,y,T_6
35:par,T_6,in,_
36:par,z,inout,_
37:par,z,inout,_
38:call,p1,_,_
39:halt,_,_,_
40:end_block,example2,_,_
```

Κώδικας σε .c αρχείο:

```
#include <stdio.h>

int main()
{
    int v,T_0,v,a,T_1,a,k,T_2,q,T_3,z,x,T_4,w,T_5,w,x,T_6;
    L_0:
    L_1: if (v!=0) goto L_2;
    L_2: goto L_6;
    L_3: T_0=z+b;
    L_4: v=T_0;
    L_5: a=1;
    L_6: goto L_8;
    L_7: T_1=v/b;
    L_8: a=T_1;
    L_9: k=x;
    L_10: T_2=y+w;
    L_11: q=T_2;
    L_12: T_3=q*x;
    L_13: z=T_3;
    L_14: if (x<y) goto L_22;
    L_15: goto L_25;
    L_16: T_4=x+y;
    L_17: w=T_4;
    L_18: goto L_27;
    L_19: T_5=x*y;
    L_20: w=T_5;
    L_21: x=1;
    L_22: y=2;
    L_23: T_6=x+y;
    L_24: {}
}
```



## Κώδικας σε .asm αρχείο:

```

L0:
L1:      j Lmain
p3:
L2:      sw $ra, ($sp)
L3:      lw $t0, -4($sp)
L4:      lw $t0, -4($t0)
L5:      addi $t0, $t0, -20
L6:      lw $t0, ($t0)
L7:      lw $t1, ($t0)
L8:      li $t2, 0
L9:      bne $t1, $t2, L4
L10:     j L8
L11:     lw $t0, -4($sp)
L12:     addi $t0, $t0, -12
L13:     lw $t0, ($t0)
L14:     lw $t1, ($t0)
L15:     lw $t0, -16($sp)
L16:     lw $t2, ($t0)
L17:     add $t1, $t1, $t2
L18:     sw $t1, -24($sp)
L19:     lw $t1, -24($sp)
L20:     lw $t0, -12($sp)
L21:     sw $t1, ($t0)
L22:     lw $t1, -24($sp)
L23:     lw $t0, -12($sp)
L24:     sw $t1, ($t0)
L25:     addi $fp, $sp, 32
L26:     addi $t0, $sp, -16
L27:     sw $t0, -12($fp)
L28:     li $t1, 1
L29:     sw $t1, -12($s0)
L30:     li $t1, 2
L31:     sw $t1, -16($s0)
L32:     lw $t1, -12($s0)
L33:     lw $t2, -16($s0)
L34:     add $t1, $t1, $t2
L35:     sw $t1, -24($s0)
L36:     addi $fp, $sp, 36
L37:     lw $t0, -24($s0)
L38:     sw $t0, -12($fp)
L39:     addi $t0, $sp, -20
L40:     sw $t0, -16($fp)
L41:     addi $t0, $sp, -20
L42:     sw $t0, -20($fp)
L43:     lw $t0, -4($sp)
L44:     sw $t0, -4($fp)
L45:     addi $sp $sp, 36
L46:     jal pl
L47:     addi $sp $sp, -36
L48:

```