**Μεταφραστές:**

**Τελική αναφορά εργαστηριακής άσκησης**

*Ονόματα μελών ομάδας:*

Γούλας Χρήστος Α.Μ. 2677

Μουχάνη Ισμήνη Α.Μ. 2761

**ΠΕΡΙΕΧΟΜΕΝΑ :**

1. Εισαγωγή:

1.1. Στόχος εργαστηριακής άσκησης.

1.2. Η γλώσσα ΕΕL.

1.3. Συντακτικό της γλώσσας.

1.4. Γραμματική της γλώσσας.

2. Παραγωγή Μεταφραστή για τη γλώσσα EEL:

2.1. Λεκτικός Αναλυτής.

2.2. Συντακτικός Αναλυτής.

2.3. Παραγωγή Ενδιάμεσου Κώδικα.

2.4. Πίνακας Συμβόλων.

2.5. Παραγωγή Τελικού κώδικα.

2.6. Σημασιολογική ανάλυση.

3.Εκτέλεση κώδικα:

3.1.Παραγωγή πεντάδων ενδιάμεσου κώδικα.

3.2.Παραγωγή αντίστοιχου κώδικα σε C.

3.3.Παραγωγή τελικού κώδικα Assembly.

4.Παρατηρήσεις-Σημειώσεις.

**Εισαγωγή:**

* 1. Ο αντικειμενικός σκοπός αυτή της εργασίας είναι η κατανόηση των εννοιών, η πρακτική εφαρμογή αυτών καθώς και η εξοικείωση με τη διαδικασία παραγωγής ενός μεταφραστή της γλώσσας EEL . Η διαδικασία αυτή διακρίνεται σε πέντε στάδια: στην παραγωγή του λεκτικού και του συντακτικού αναλυτή, του ενδιάμεσου κώδικα, του πίνακα συμβόλων και του τελικού κώδικα τα οποία θα αναλυθούν εκτενέστερα στα επόμενα κεφάλαια.
  2. Η EEL (Early Experimental Language) είναι μια μικρή γλώσσα προγραμματισμού. Yποστηρίζει συναρτήσεις και διαδικασίες, μετάδοση παραμέτρων με αναφορά και τιμή, αναδρομικές κλήσεις, και άλλες δομές. Eπίσης επιτρέπει φώλιασμα στη δήλωση συναρτήσεων και διαδικασιών.Δεν υποστηρίζει βασικά προγραμματιστικά εργαλεία όπως η δομή for, ή τύπους δεδομένων όπως οι πραγματικοί αριθμοί και οι συμβολοσειρές.
  3. Το αλφάβητο της EEL αποτελείται από:

• τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου («Α»,…,«Ζ» 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 μεσα 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.

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.

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»

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.

* 1. Η γραμματική της γλώσσας περιγράφεται από τους παρακάτω κανόνες:

<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> ::= exit

<while-stat> ::= while <condition><statements>endwhile

<switch-stat> ::= switch<expression>(case <expression>:<statements>)+endswitch

<forcase-stat> ::= forcase ( when <condition>: <statements>)+ endforcase

<call-stat> ::= call id <actualpars>

<return-stat> ::= return <expression>

<print-stat> ::= print <expression>

<input-statt> ::= input id

<actualpars> ::= ( <actualparlist>)

<actualparlis> ::= <actualparitem> ( , <actualparitem> )\* | ε

<actualparitemm> ::= in <expression> | inout id

<return-statt> ::= return<expression>

<condition> ::= <boolterm>(or <boolterm>)\*

<boolterm> ::= <boolfactor> (and <boolfactor>)\*

<boolfactorr> ::=not [<conditionn>] | [<conditionn>] | <expressionn> <relational-oper> <expression> |true | false

<expression> ::= <optional-sign> <term> ( <add-operr> <term>)\*

<term> ::= <factor> (<mul-oper> <factor>)\*

<factor> ::= constant | (<expression>) | id <idtail>

<idtail> ::= ε | <actualpars>

<relational-oper> ::= = | <= |>= | > | < | <>

<add-oper> ::= + | -

<mul-oper> ::= \* | /

<optional-sign> ::= ε | <add-oper>

**Παραγωγή Μεταφραστή για τη γλώσσα EEL:**

**2.1.** Το πρώτο στάδιο στη δημιουργία του μεταφραστή είναι η υλοποίηση του λεκτικού αναλυτή. Ο Λεκτικός αναλυτής αποτελείται από μία συνάρτηση που καλείται από το συντακτικό αναλυτή (βλέπε κεφάλαιο 2.2) , διαβάζει γράμμα-γράμμα τον πηγαίο κώδικα και επιστρέφει στο συντακτικό αναλυτή έναν πίνακα δύο θέσεων που στην πρώτη του θέση περιέχει τη λεκτική μονάδα την οποία διάβασε και στη δεύτερη θέση το αναγνωριστικό αυτής. Το αναγνωριστικό αυτό στην περίπτωση που δεν έχουμε δεσμευμένη λέξη είναι ένα αλφαριθμητικό το οποίο επιλέξαμε εμείς και το οποίο χαρακτηρίζει την λεκτική μονάδα. Στην αντίθετη περίπτωση είναι η ίδια δεσμευμένη λέξη με την κατάληξη 'tk' στο τέλος.

Συγκεκριμένα για να παραχθεί αυτός ο πίνακας για κάθε λεκτική μονάδα καλείται η συνάρτηση lektikos\_analutis() ,η οποία ουσιαστικά εσωτερικά λειτουργεί σαν ένα αυτόματο καταστάσεων, το οποίο ξεκινά από μια αρχική κατάσταση με είσοδο ένα χαρακτήρα και αλλάζει καταστάσεις μέχρι να συναντήσει μια τελική κατάσταση και η οποία ακολουθεί την εξής διαδικασία:

Αρχικά, χρησιμοποιώντας την μεταβλητή position η οποία αρχικοποιείται στο μηδέν και στην οποία κρατάμε την θέση στην οποία βρισκόμαστε κάθε φορά μέσα στο αρχείο το οποίο διαβάζουμε, πρώτα κάνουμε έναν έλεγχο για την περίπτωση που έχει ξεπεράσει το μέγεθος του αρχείου, ο οποίος αν ισχύει επιστρέφει τον πινάκα με στοιχεία [eof, eoftk] και μετά τρέχουμε ένα while loop το οποίο εκτελείται όσο αυτή είναι μικρότερη από το μέγεθος του αρχείου(δηλαδή όσο υπάρχουν ακόμα λεκτικές μονάδες για διάβασμα).Μέσα σε αυτό υπάρχει ένα πλήθος από if καθένα από τα οποία χρησιμοποιείται για τις διαφορετικές λεκτικές μονάδες που μπορούμε να έχουμε. Αναλυτικά:

1)Στην περίπτωση που συναντήσουμε '+' (σύμβολο πρόσθεσης) και βρισκόμαστε στην κατάσταση μηδέν ,πράγμα το οποίο μας δείχνει η μεταβλητή state στην οποία κρατάμε κάθε φορά την κατάσταση στην οποία βρισκόμαστε και η οποία αρχικά είναι και αυτή μηδέν, επιστρέφουμε τον πίνακα με στοιχεία ['+','plustk'] ,αυξάνουμε το position κατά ένα πάντα και βγαίνουμε από το loop.

2)Στην περίπτωση που συναντήσουμε '\t' ή κενό απλά αυξάνουμε το position και βγαίνουμε από το loop, αλλιώς αν βρούμε '\n'(δηλαδή αλλαγή γραμμής) πρώτα αυξάνουμε την μεταβλητή lines, η οποία κρατάει τον αριθμό των γραμμών που έχουμε και η οποία αρχικοποιείται στο μηδέν, και μετά κάνουμε και τα βήματα που ακολουθήσαμε στο '\t' και το κενό.

3)Στην περίπτωση που συναντήσουμε '-' (σύμβολο αφαίρεσης) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία ['-','minustk'], αυξάνουμε το position και βγαίνουμε από το loop.

4)Στην περίπτωση που συναντήσουμε '\*'(σύμβολο πολ/μου) και βρισκόμαστε στην κατάσταση μηδεν, επιστρέφουμε τον πίνακα με στοιχεία ['\*','mulstk'], αυξάνουμε το position και βγαίνουμε απο το loop.

5)Στην περίπτωση που συναντήσουμε '/' και βρισκόμαστε στην κατάσταση μηδέν, αλλάζουμε κατάσταση από μηδέν σε έξι, αυξάνουμε το position και ελέγχουμε την επόμενη λεκτική μονάδα γιατί υπάρχουν παραπάνω από μια περιπτώσεις τις οποίες μπορούμε να έχουμε. Συγκεκριμένα:

5.1)Αν ξανασυναντήσουμε '/' και βρισκόμαστε στην κατάσταση έξι ,σημαίνει οτι έχουμε βρεί σχόλιο, επιστρέφουμε τον πίνακα με στοιχεία['//','commenttk'],και αυξάνουμε το position.

5.2)Αν συναντήσουμε '\*' και βρισκόμαστε στην κατάσταση έξι ,σημαίνει οτι έχουμε βρεί άνοιγμα σχόλιου, επιστρέφουμε τον πίνακα με στοιχεία['/\*','commentοtk'], κάνουμε την κατάσταση επτά και αυξάνουμε το position.

5.3)Αν δεν συναντήσουμε τίποτα από τα παραπάνω σημαίνει πως έχουμε βρει το σύμβολο της διαίρεσης, επιστρέφουμε τον πίνακα με στοιχεία['/','divtk'] ,επαναφέρουμε την κατάσταση στο μηδέν για να πάμε πάλι από την αρχή και βγαίνουμε από το loop.

6)Στην περίπτωση που δεν συναντήσουμε '\n' και βρισκόμαστε στην κατάσταση έξι σημαίνει ότι βρισκόμαστε μέσα σε σχόλιο οπότε αγνοούμε τα περιεχόμενά του, τα οποία δεν μας χρειάζονται στην μετάφραση και αυξάνουμε απλά το position.

7)Στην περίπτωση που δεν συναντήσουμε '\*' και βρισκόμαστε στην κατάσταση επτά( δηλαδή είμαστε μέσα σε σχόλια) ,αρχικά κάνουμε έναν έλεγχο για το αν βρισκόμαστε στο τέλος του αρχείου ο οποίος αν ισχύει επιστρέφει μήνυμα λάθους διότι τα σχόλια δεν κλείνουν ποτέ και βγαίνουμε από το loop αλλιώς αυξάνουμε απλά το position.

8)Στην περίπτωση που συναντήσουμε '\*' και βρισκόμαστε στην κατάσταση επτά(δηλαδή είμαστε μέσα σε σχόλια) , αυξάνουμε το position αγνοώντας τα περιεχόμενα του σχόλιου μέχρι να συναντήσουμε '/',που σημαίνει ότι φτάσαμε στο τέλος του σχόλιου, όπου επαναφέρουμε την κατάσταση στο μηδέν, επιστρέφουμε τον πίνακα με στοιχεία['\*/','commentctk'] και αυξάνουμε το position.

9)Στην περίπτωση που συναντήσουμε '\n' και βρισκόμαστε στην κατάσταση έξι σημαίνει ότι φτάσαμε στο τέλος του σχόλιου, όπου αυξάνουμε τις γραμμές κατά ένα ,επαναφέρουμε την κατάσταση στο μηδέν και αυξάνουμε το position.

10)Στην περίπτωση που συναντήσουμε '='( ισότητα) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία ['=','equalstk'], αυξάνουμε το position και βγαίνουμε από το loop.

11)Στην περίπτωση που συναντήσουμε '(' (άνοιγμα παρένθεσης) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία ['(','openbtk'], αυξάνουμε το position και βγαίνουμε από το loop.

12)Στην περίπτωση που συναντήσουμε ')' (κλείσιμο παρένθεσης) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία [')','closeptk'], αυξάνουμε το position και βγαίνουμε από το loop.

13)Στην περίπτωση που συναντήσουμε '[' (άνοιγμα αγκύλης) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία ['[','openptk'], αυξάνουμε το position και βγαίνουμε από το loop.

14)Στην περίπτωση που συναντήσουμε ']' (κλείσιμο αγκύλης) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία [']','closeptk'], αυξάνουμε το position και βγαίνουμε από το loop.

15)Στην περίπτωση που συναντήσουμε ',' (κόμμα) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία [',' ,'commatk'], αυξάνουμε το position και βγαίνουμε από το loop.

16)Στην περίπτωση που συναντήσουμε ';' (διαχωριστής) και βρισκόμαστε στην κατάσταση μηδέν, επιστρέφουμε τον πίνακα με στοιχεία [';' ,'questiontk'], αυξάνουμε το position και βγαίνουμε από το loop.

17)Στην περίπτωση που συναντήσουμε '<' και βρισκόμαστε στην κατάσταση μηδέν, αλλάζουμε κατάσταση από μηδέν σε τρία, αυξάνουμε το position και ελέγχουμε την επόμενη λεκτική μονάδα γιατί υπάρχουν παραπάνω από μια περιπτώσεις τις οποίες μπορούμε να έχουμε. Συγκεκριμένα:

17.1)Αν συναντήσουμε '>' και βρισκόμαστε στην κατάσταση τρία ,σημαίνει ότι έχουμε βρει το σύμβολο του διάφορου, επιστρέφουμε τον πίνακα με στοιχεία['<>','notequalstk'], επαναφέρουμε την κατάσταση στο μηδέν ,αυξάνουμε το position και βγαίνουμε από το loop.

17.2)Αν συναντήσουμε '=' και βρισκόμαστε στην κατάσταση τρία ,σημαίνει ότι έχουμε βρει το σύμβολο του μικρότερου ή ίσου, επιστρέφουμε τον πίνακα με στοιχεία['<=','minequalstk'], επαναφέρουμε την κατάσταση στο μηδέν, αυξάνουμε το position και βγαίνουμε από το το loop.

17.3)Αν δεν συναντήσουμε τίποτα από τα παραπάνω σημαίνει πως έχουμε βρει το σύμβολο του μικρότερου, επιστρέφουμε τον πίνακα με στοιχεία['<','mintk'] ,επαναφέρουμε την κατάσταση στο μηδέν για να πάμε πάλι από την αρχή και βγαίνουμε από το loop.

18)Στην περίπτωση που συναντήσουμε '>' και βρισκόμαστε στην κατάσταση μηδέν, αλλάζουμε κατάσταση από μηδέν σε τέσσερα, αυξάνουμε το position και ελέγχουμε την επόμενη λεκτική μονάδα γιατί υπάρχουν παραπάνω από μια περιπτώσεις τις οποίες μπορούμε να έχουμε. Συγκεκριμένα:

18.1)Αν συναντήσουμε '=' και βρισκόμαστε στην κατάσταση τέσσερα ,σημαίνει ότι έχουμε βρει το σύμβολο του μεγαλύτερου ή ίσου, επιστρέφουμε τον πίνακα με στοιχεία['>=','maxequalstk'], επαναφέρουμε την κατάσταση στο μηδέν ,αυξάνουμε το position και βγαίνουμε από το loop.

18.2)Αν δεν συναντήσουμε το παραπάνω σημαίνει πως έχουμε βρει το σύμβολο του μεγαλύτερου, επιστρέφουμε τον πίνακα με στοιχεία ['>','maxtk'] ,επαναφέρουμε την κατάσταση στο μηδέν για να πάμε πάλι από την αρχή και βγαίνουμε από το loop.

19)Στην περίπτωση που συναντήσουμε ':' (διαχωριστής) και βρισκόμαστε στην κατάσταση μηδέν, αλλάζουμε κατάσταση από μηδέν σε πέντε, αυξάνουμε το position και ελέγχουμε την επόμενη λεκτική μονάδα γιατί υπάρχουν παραπάνω από μια περιπτώσεις τις οποίες μπορούμε να έχουμε. Συγκεκριμένα:

19.1)Αν ξανασυναντήσουμε '=' και βρισκόμαστε στην κατάσταση πέντε ,σημαίνει οτι έχουμε βρει ανάθεση, επιστρέφουμε τον πίνακα με στοιχεία[':=','assigmenttk'], αυξάνουμε το position,επαναφέρουμε την κατάσταση στο μηδέν και βγαίνουμε από το loop.

19.2)Αν δεν συναντήσουμε το παραπάνω σημαίνει πως έχουμε βρει διαχωριστή, επιστρέφουμε τον πίνακα με στοιχεία [':','doubledottk'], επαναφέρουμε την κατάσταση στο μηδέν και βγαίνουμε από το loop.

20)Στην περίπτωση που συναντήσουμε αλφαριθμητικό και βρισκόμαστε στην κατάσταση μηδέν, το κρατάμε στην μεταβλητή token διότι θέλουμε να ελέγξουμε και αυτά που θα ακολουθήσουν ,αλλάζουμε κατάσταση από μηδέν σε ένα, και αυξάνουμε το position.

21)Στην περίπτωση που συναντήσουμε αλφαριθμητικό ή αριθμό και βρισκόμαστε στην κατάσταση ένα(δηλαδή έχουμε βρει αλφαριθμητικό ήδη),το προσθέτουμε στην μεταβλητή token διότι συνεχίζουμε να βρίσκουμε γράμματα του αλφαριθμητικού και αυξάνουμε το position.

22)Στην περίπτωση που δεν συναντήσουμε αλφαριθμητικό ή αριθμό και βρισκόμαστε στην κατάσταση ένα(δηλαδή έχουμε βρει αλφαριθμητικό ήδη),επαναφέρουμε την κατάσταση στο μηδέν διότι τελείωσε η λέξη που βρήκαμε, καλούμε την συνάρτηση is\_bounded(token) με όρισμα αυτή την λέξη για να διαπιστώσουμε αν έχουμε κάποιο όνομα μεταβλητής ή κάποια δεσμευμένη λέξη, και βγαίνουμε από το loop.Συγκεκριμένα η συνάρτηση is\_bounded(token) διατρέχει τον πίνακα boundedwords, ο οποίος περιέχει όλες τις δεσμευμένες λέξεις της γλώσσας, ψάχνοντας το αλφαριθμητικό που πήρε σαν όρισμα όταν καλέστηκε. Αν το βρει επιστρέφει τον πίνακα που στην πρώτη θέση του έχει το ίδιο το αλφαριθμητικό και την δεύτερη θέση του έχει το αλφαριθμητικό με την λέξη 'tk' στο τέλος αλλιώς επιστρέφει τον πίνακα που στην πρώτη θέση του έχει το ίδιο το αλφαριθμητικό και την δεύτερη θέση του έχει την λέξη 'idtk' .

23)Στην περίπτωση που συναντήσουμε αριθμό και βρισκόμαστε στην κατάσταση μηδέν(δηλαδή δεν έχουμε βρει κανέναν αριθμό ακόμα),το κρατάμε στην μεταβλητή token διότι μπορεί να βρούμε κι άλλους στη συνέχεια, αλλάζουμε την κατάσταση από μηδέν σε δύο και αυξάνουμε το position.

24)Στην περίπτωση που συναντήσουμε αριθμό και βρισκόμαστε στην κατάσταση δύο(δηλαδή έχουμε βρει κάποιον αριθμό ήδη),το προσθέτουμε στην μεταβλητή token και αυξάνουμε το position για να ελέγξουμε το επόμενο.

25)Στην περίπτωση που συναντήσουμε αλφαριθμητικό και βρισκόμαστε στην κατάσταση δύο(δηλαδή έχουμε βρει κάποιον αριθμό ήδη), εμφανίζουμε μήνυμα λάθους διότι δεν μπορούμε να έχουμε αλφαριθμητικό μετά από αριθμό σύμφωνα με τους κανόνες της γλώσσας και βγαίνουμε από το πρόγραμμα .

26)Στην περίπτωση δεν που συναντήσουμε αριθμό και βρισκόμαστε στην κατάσταση δύο(δηλαδή έχουμε βρει κάποιον αριθμό ήδη), καλούμε την συνάρτηση check\_num(token) η οποία ελέγχει αν ο αριθμός βρίσκεται μέσα στα όρια που επιτρέπει η γλώσσα μας (int(token)<32767) και (int(token)>-32767)) .Αν δεν είναι τότε εμφανίζουμε μήνυμα λάθους και βγαίνουμε από το πρόγραμμα αλλιώς επιστρέφουμε πίνακα ο οποίος στην πρώτη του θέση έχει τον αριθμό και στην δεύτερη του θέση έχει την λέξη 'consttk'. Μετά την επιστροφή της συνάρτησης επαναφέρουμε την κατάσταση στο μηδέν και βγαίνουμε από το loop.

27)Η τελευταία περίπτωση είναι αυτή που δεν συναντήσουμε τίποτα από τα παραπάνω, δηλαδή έχουμε βρει κάτι που δεν ανήκει στην γλώσσα μας, στην οποία εμφανίζουμε αντίστοιχο μήνυμα λάθους και βγαίνουμε από το πρόγραμμα.

**2.2** Αναφέραμε ήδη ότι ο λεκτικός αναλυτής καλείται από το συντακτικό αναλυτή. Τι είναι όμως ο συντακτικός αναλυτής; Ο συντακτικός αναλυτής είναι ένα σύνολο από συναρτήσεις που υλοποιούν τη γραμματική της γλώσσας. Από το σύνολο αυτών το συναρτήσεων η πρώτη που καλείται είναι η program(), η οποία με τη σειρά της καλεί και τις υπόλοιπες. Γενικότερα η ιδέα για την αλληλεπίδραση όλων αυτών των συναρτήσεων που απαρτίζουν τον συντακτικό αναλυτή είναι πως ανάλογα με την λεκτική μονάδα που επιστρέφεται, από το συντακτικό αναλυτή, γίνονται οι ανάλογες κλήσεις στις υπόλοιπες συναρτήσεις.

Πιο συγκεκριμένα:

Η συνάρτηση program() είναι η πρώτη συνάρτηση που εκτελείται. Πριν από την κλήση αυτής, έχουμε διαβάσει μέσω του λεκτικού αναλυτή την πρώτη λεκτική μονάδα και έχουμε αποθηκεύσει αυτή και το αναγνωριστικό της στις δύο θέσεις του πίνακα tokenboard. Αν το αναγνωριστικό που βρίσκεται στη δεύτερη θέση του πίνακα (tokenboard[1]) ταυτίζεται με το programtk τότε η εκτέλεση του κώδικα μπορεί να συνεχίσει κανονικά όπως ορίζει και η γραμματική μας. Σε περίπτωση αποτυχίας της ταύτισης αυτής εμφανίζεται το αντίστοιχο μήνυμα λάθους.

Στη συνέχεια καλεί το λεκτικό αναλυτή για να πάρει την επόμενη λεκτική μονάδα όπου το αποτέλεσμα που θα επιστραφεί θα αποθηκευτεί όπως και πριν στις 2 θέσεις του πίνακα tokenboard. Αν το αναγνωριστικό τώρα ταυτίζεται με idtk που σημαίνει ότι αυτό που πήραμε δεν είναι δεσμευμένη λέξη, αριθμός ή κάποια αριθμητική πράξη, τότε η λεκτική μονάδα που έχουμε στον πίνακα είναι το όνομα του προγράμματος και η εκτέλεση του προγράμματος συνεχίζεται κανονικά ειδάλλως θα πάρουμε το αντίστοιχο μήνυμα λάθους.

Καλούμε πάλι το λεκτικό αναλυτή και σειρά έχει η block() να εκτελεστεί στην οποία περνάμε και ως όρισμα το όνομα του προγράμματος. Στη block καλούνται οι εξής συναρτήσεις με την αντίστοιχη σειρά: declarations(),subprograms(),statements(). Αφού εκτελεστούν αυτές επιστρέφει στην program().

Στη declarations εξετάζουμε το ενδεχόμενο να έχει δηλώσει κάτι ο χρήστης στο πρόγραμμα. Ο έλεγχος αυτός γίνεται με το αντίστοιχο αναγνωριστικό declaretk. Εδώ δεν χρειάζεται να πετάξουμε μήνυμα λάθους στο χρήστη σε περίπτωση που δεν συναντήσουμε το αναγνωριστικό αφού υπάρχει το ενδεχόμενο ο χρήστης να μην έχει δηλώσει τίποτα στο πρόγραμμα που έχει γράψει και συνεπώς να μην έχει χρησιμοποιήσει καθόλου τη λέξεις declare και το αντίστοιχο αναγνωριστικό . Αν όμως συναντήσουμε το αναγνωριστικό αυτό τότε καλούμε το λεκτικό αναλυτή και τη συνάρτηση varlist(). Όταν επιστρέψει από τη varlist() θα εκτελεστεί ένας έλεγχος διότι σύμφωνα με τη γραμματική μας η δήλωση μεταβλητών τελειώνει με τη δεσμευμένη λέξη enddeclare η οποία πρέπει οπωσδήποτε να συναντηθεί μετά τις μεταβλητές που δηλώνει ο χρήστης. Αν δε βρούμε το αντίστοιχο αναγνωριστικό τότε εμφανίζουμε στο χρήστη το αντίστοιχο μήνυμα λάθους.

Στη varlist εξετάζουμε αν το αναγνωριστικό που πήραμε είναι τύπου idtk. Στη συνέχεια καταναλώνουμε την επόμενη λεκτική μονάδα και κοιτάζουμε αν είναι κόμμα (commatk ως αναγνωριστικό),γιατί υπάρχει το ενδεχόμενο ο χρήστης να έχει δηλώσει πολλές μεταβλητές χωρισμένες με κόμματα. Για το λόγο αυτό και στον έλεγχο του commatk χρησιμοποιήσαμε τη δομή while. Φυσικά έχουμε και ένα μήνυμα λάθους το οποίο θα εμφανιστεί στο χρήστη αν μέσα στις δηλώσεις που κάνει περάσει κάτι το οποίο θα γυρίσει ένα αναγνωριστικό διαφορετικό του idtk. Κάτι θα μπορούσε να συμβεί αν για παράδειγμα ο χρήστης δηλώσει ως όνομα μεταβλητής μία δεσμευμένη λέξη καθώς αυτές όπως έχουμε ήδη αναφέρει στο λεκτικό αναλυτή έχουν διαφορετικό αναγνωριστικό.

Επιστρέφοντας από τη declarations() (στη block) σειρά έχει να εκτελεστεί η subprograms().Στη συνάρτηση αυτή εξετάζουμε αν το αναγνωριστικό που πήραμε είναι συνάρτησης ή διαδικασίας. Ο έλεγχος αυτός γίνεται μέσα σε δομή while γιατί μπορεί να βρούμε πολλαπλές συναρτήσεις ή διαδικασίες. Αν το αναγνωριστικό είναι ίδιο με proceduretk ή functiontk τότε πηγαίνουμε στην procorfunc().

Στην procorfunc() διακρίνουμε τρεις περιπτώσεις: αν είναι αναγνωριστικό συνάρτησης ή διαδικασίας αυτό που πήραμε, ή κάτι διαφορετικό από αυτά τα δύο οπότε θα πρέπει να εμφανίσουμε μήνυμα λάθους. Αν το αναγνωριστικό που συναντήσαμε είναι procedurtk ή functiontk, καταναλώνουμε με σκοπό να πάρουμε στην επόμενη λεκτική μονάδα που θα μας επιστραφεί το όνομα της συνάρτησης ή της διαδικασίας. Φυσικά κάνουμε έναν έλεγχο σε περίπτωση που ο χρήστης δεν έγραψε όνομα στη συνάρτηση ή έβαλε κάποια δεσμευμένη λέξη ως όνομα. Αν δεν είναι idtk αυτό που μας επιστράφηκε φυσικά πετάμε μήνυμα λάθους, ειδάλλως καταναλώνουμε και πάλι και εκτελούμε την procorfuncbody() με όρισμα το όνομα της συνάρτησης ή διαδικασίας αντίστοιχα.

H procorfuncbody() καλεί με την εξής σειρά άλλες 2 συναρτήσεις τις formalpars() και block() με το όρισμα που μόλις πήρε από αυτήν που την κάλεσε (procorfunc). Έχουμε ήδη εξηγήσει πως λειτουργεί η block οπότε όταν γυρίσει από την formalpars() θα γίνει η ίδια διαδικασία.

Στη formalpars() ελέγχουμε αν το αναγνωριστικό που πήραμε από το λεκτικό αναλυτή είναι αυτό της αριστερής παρένθεσης ‘’openbtk’’.Αυτό γίνεται γιατί μετά από το όνομα της συνάρτησης ή διαδικασίας πρέπει να ακολουθήσουν παρενθέσεις οι οποίες θα είναι κενές αν δεν έχει κανένα όρισμα η συνάρτηση-διαδικασία ειδάλλως να περιέχει τον τρόπο περάσματος παραμέτρων και στη συνέχεια το όνομα αυτών .Καταναλώνουμε την επόμενη λεκτική μονάδα και στη συνέχεια εξετάζουμε τι είδους πέρασμα παραμέτρων έχουμε, με τιμή ή με αναφορά και εκτελείται η formalparlist. Όταν επιστρέψουμε από τη formalparlist ελέγχουμε αν συναντάμε δεξιά παρένθεση. Τα μηνύματα λάθους σε αυτή τη συνάρτηση είναι να μην βρούμε αριστερή παρένθεση μετά το όνομα της συνάρτησης ή διαδικασίας. Να μην βρούμε δεξιά παρένθεση μετά την επιστροφή από τη formalparlist() και να μας έρθει ως αναγνωριστικό κάτι διαφορετικό από τις δύο δεσμευμένες λέξεις in inout.

Στη formalparlist() καλούμε τη formalparitem() και όταν επιστρέψουμε από αυτή έχουμε μια δομή while για να βρίσκει αν έχουμε πολλές παραμέτρους χωρισμένες με κόμματα. Αυτό γίνεται ελέγχοντας αν συναντάμε το αναγνωριστικό του κόμματος ‘’commatk’’. Οπότε όσο είμαστε μέσα στο while καταναλώνουμε την επόμενη λεκτική μονάδα και καλούμε πάλι τη formalparitem.

Στη formalparitem() κάνουμε τους εξής ελέγχους: Αv έχουμε πέρασμα παραμέτρων με τιμή, αν έχουμε πέρασμα παραμέτρων με αναφορά ή τίποτα από τα δύο, συνεπώς θα πρέπει να επιστρέψουμε στο χρήστη μήνυμα λάθους. Αν τελικά έχουμε βρει πέρασμα με τιμή ή αναφορά στη συνέχεια καλούμε το λεκτικό αναλυτή για να πάρουμε το όνομα της παραμέτρου. Αν στον έλεγχο το αναγνωριστικό που πήραμε από το λεκτικό αναλυτή δεν είναι idtk τότε εμφανίζουμε το αντίστοιχο μήνυμα λάθους.

Επιστρέφοντας πίσω στη block είναι η σειρά της statements() να εκτελεστεί. Στη statements(), στο βρόχο while ελέγχουμε αν βρήκαμε το αναγνωριστικό που αντιστοιχεί στο ερωτηματικό, ‘’questiontk’’, καλούμε το λεκτικό αναλυτή και εκτελούμε τη statement.

Στη statement ελέγχουμε τη λεκτική μονάδα που μας επέστρεψε ο λεκτικός αναλυτής και διακρίνουμε τις εξής περιπτώσεις:

* Aν το αναγνωριστικό είναι idtk καλούμε τη συνάρτηση assignment\_stat().
* Aν το αναγνωριστικό είναι iftk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης if, καλούμε τη συνάρτηση if\_stat().
* Aν το αναγνωριστικό είναι whiletk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης while, καλούμε τη συνάρτηση while\_stat().
* Aν το αναγνωριστικό είναι repeattk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης repeat, καλούμε τη συνάρτηση repeat\_stat().
* Aν το αναγνωριστικό είναι exittk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης exit, καλούμε τη συνάρτηση exit\_stat().
* Aν το αναγνωριστικό είναι switchtk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης switch, καλούμε τη συνάρτηση switch\_stat().
* Aν το αναγνωριστικό είναι forcasetk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης for, καλούμε τη συνάρτηση forcase\_stat().
* Aν το αναγνωριστικό είναι calltk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης call, καλούμε τη συνάρτηση call\_stat().
* Aν το αναγνωριστικό είναι returntk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης return, καλούμε τη συνάρτηση return\_stat().
* Aν το αναγνωριστικό είναι inputtk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης input, καλούμε τη συνάρτηση input\_stat().
* Aν το αναγνωριστικό είναι printtk, που αντιστοιχεί στο αναγνωριστικό της δεσμευμένης λέξης print, καλούμε τη συνάρτηση print\_stat().

Στη assignment\_stat() ελέγχουμε το αναγνωριστικό αν είναι idtk.Αν δεν είναι πετάμε μήνυμα λάθους στο χρήστη. Αν είναι καλούμε το λεκτικό αναλυτή και ελέγχουμε αν αυτό που μας επέστρεψε είναι το αναγνωριστικό της ανάθεσης. Αν είναι καλούμε το λεκτικό αναλυτή, τη συνάρτηση expression και επιστρέφουμε στη συνάρτηση που μας κάλεσε. Αν δεν είναι ενημερώνουμε το χρήστη με μήνυμα λάθους.

Στην if\_stat() ελέγχουμε αν το αναγνωριστικό στον πίνακα μας είναι το αναγνωριστικό του βρόγχου if. Αν δεν είναι ενημερώνουμε το χρήστη με μήνυμα λάθους αλλιώς καλούμε το λεκτικό αναλυτή, τη συνάρτηση condition() και εξετάζουμε αν αυτό που μας επέστρεψε είναι το αναγνωριστικό του then,αν δεν είναι πετάμε μήνυμα λάθους στο χρήστη. Αν είναι καλούμε και πάλι το λεκτικό αναλυτή και στη συνέχεια τη statements(). Μετά τη statements καλούμε την else\_part() και ακολουθεί ένας έλεγχος για να ελέγξουμε αν βρήκαμε το αναγνωριστικό που τερματίζει το βρόγχο if,δηλαδή το αναγνωριστικό της δεσμευμένης λέξης endif. Αν δεν το βρούμε τυπώνεται το αντίστοιχο μήνυμα λάθους.

Στην else\_part() ελέγχουμε αν εμφανίζεται το αναγνωριστικό του else και αν ναι τότε καλούμε τον λεκτικό αναλυτή και στη συνέχεια εκτελούμε τη statements().

Στη repeat\_stat() ελέγχουμε αν βρήκαμε το αναγνωριστικό του βρόγχου repeat. Αν όχι τυπώνουμε ένα μήνυμα λάθους. Ειδάλλως καλούμε το λεκτικό αναλυτή και εκτελούμε τη συνάρτηση statements(). Στη συνέχεια ελέγχουμε αν η λεκτική μονάδα που πήραμε είναι το αναγνωριστικό του τέλος του βρόγχου της repeat, δηλαδή το αναγνωριστικό της δεσμευμένη λέξης endrepeat. Αν είναι επιστρέφουμε στη statement αλλιώς τυπώνουμε μήνυμα λάθους και επιστρέφουμε.

Στην exit\_stat() ελέγχουμε αν το αναγνωριστικό που έχουμε στην δεύετερη θέση του πίνακα μας είναι το αναγνωριστικό της δεσμευμένης λέξης if. Αν είναι καλούμε το λεκτικό αναλυτή. Αν όχι τότε εμφανίζουμε το αντίστοιχο μήνυμα λάθους.

Στη while\_stat() ελέγχουμε αν το αναγνωριστικό στον πίνακα μας είναι το αντίστοιχο της δεσμευμένης λέξης while. Αν είναι προχωράμε και καλούμε το λεκτικό αναλυτή για να πάρουμε την επόμενη λεκτική μονάδα, καλούμε τη συνάρτηση condition() και εκτελούμε τη statements(). Αν όχι εμφανίζουμε μήνυμα λάθους. Εφόσον έχουμε βρει το αναγνωριστικό του while και έχουμε καλέσει το λεκτικό αναλυτή και τη statements() στη συνέχεια κάνουμε έναν έλεγχο αν βρήκαμε το αναγνωριστικό της δεσμευμένης λέξης endwhile. Αν όχι εμφανίζουμε μήνυμα λάθους, καθώς σύμφωνα με τη γλώσσα μας αν βρούμε while πρέπει να συναντήσουμε το endwhile που σηματοδοτεί το τέλος του βρόγχου.

Στη switch\_stat() ελέγχουμε αν βρήκαμε το αναγνωριστικό της δεσμευμένης λέξης switch,αν δεν το βρούμε εμφανίζουμε μήνυμα λάθους. Καλούμε το λεκτικό αναλυτή, καλούμε τη συνάρτηση expression() και στη συνέχεια ελέγχουμε αν αυτό που μας επέστρεψε είναι το αναγνωριστικό που της δεσμευμένης λέξης case,αν όχι ενημερώνουμε το χρήστη με μήνυμα λάθους. Καλούμε πάλι το λεκτικό αναλυτή και τη συνάρτηση expression(). Ελέγχουμε αν βρήκαμε το αναγνωριστικό του συμβόλου άνω-κάτω τελεία. Αν δεν το βρούμε εκτυπώνουμε μήνυμα λάθους. Αν το βρούμε καλούμε το λεκτικό αναλυτή και τη statements(). Στη συνέχεια έχουμε έναν βρόγχο while που ελέγχει αν συναντάμε το αναγνωριστικό της δεσμευμένης λέξης case. Καλούμε το λεκτικό αναλυτή και τη συνάρτηση expression(). Ελέγχουμε και πάλι αν συναντάμε το αναγνωριστικό της άνω-κάτω τελείας, καλούμε το λεκτικό αναλυτή και τη statements(). Έξω από το βρόγχο while έχουμε έναν έλεγχο για τη δεσμευμένη λέξη endswitch. Αν δεν τη συναντήσουμε πετάμε μήνυμα λάθους στο χρήστη. Αν τη συναντήσουμε καλούμε το λεκτικό αναλυτή.

Στη forcase\_stat() ελέγχουμε αν συναντάμε το αναγνωριστικό της δεσμευμένης λέξης for.Αν δεν το συναντήσουμε εκτυπώνουμε μήνυμα λάθους. Αν το συναντήσουμε καλούμε το λεκτικό αναλυτή. Στη συνέχεια ελέγχουμε αν αυτό που πήραμε από το λεκτικό αναλυτή είναι το αναγνωριστικό της λέξης when. Αν δεν είναι εμφανίζουμε μήνυμα λάθους. Αν είναι καλούμε το λεκτικό αναλυτή και τη συνάρτηση condition. Στη συνέχεια κάνουμε έναν έλεγχο για το αναγνωριστικό της άνω-κάτω τελείας. Αν ο λεκτικός αναλυτής δεν μας επέστρεψε αυτό το σύμβολο τότε εμφανίζουμε μήνυμα λάθους. Ειδάλλως καλούμε το λεκτικό αναλυτή και τη συνάρτηση statements() και στη συνέχεια έχουμε ένα βρόγχο while στον οποίο ελέγχουμε αν βρήκαμε το αναγνωριστικό της δεσμευμένης λέξης when. Αν το βρήκαμε μπαίνουμε μέσα στο βρόγχο και καλούμε το λεκτικό αναλυτή και τη συνάρτηση condition(). Στη συνέχεια ελέγχουμε για το αναγνωριστικό του συμβόλου άνω-κάτω τελεία σε ένα βρόχο if και αν το έχουμε πάρει από το λεκτικό αναλυτή τότε καλούμε και πάλι το λεκτικό αναλυτή και τη συνάρτηση statements(). Έξω από τη while έχουμε έναν έλεγχο για τη δεσμευμένη λέξη endfor. Καθώς σύμφωνα με τη γραμματική μας κάθε for case πρέπει στο τέλος του να έχει ένα endfor. Αν δεν το συναντήσουμε σημαίνει ότι είναι λάθος και ενημερώνουμε το χρήστη.

Στη συνάρτηση call-stat() πρώτα ελέγχουμε αν πήραμε από το λεκτικό αναλυτή το αναγνωριστικό της δεσμευμένης λέξης call. Αν όχι εκτυπώνουμε μήνυμα λάθους. Αν το πήραμε τότε καλούμε το λεκτικό αναλυτή και ελέγχουμε αν η νέα λεκτική μονάδα που πήρε έχει αναγνωριστικό idtk . Αν ισχύει η συνθήκη τότε καλούμε το λεκτικό αναλυτή και τη συνάρτηση actualpars(). Αν δεν ισχύει εμφανίζουμε μήνυμα λάθους.

Στη συνάρτηση return\_stat() ελέγχουμε αν η λεκτική μονάδα που συναντήσαμε είναι η δεσμευμένη λέξη return.Αν δεν είναι οδηγούμαστε στο else και εκτυπώνουμε μήνυμα λάθους. Αν είναι καλούμε το λεκτικό αναλυτή και τη συνάρτηση expression().

Στη συνάρτηση print\_stat() ελέγχουμε αν η λεκτική μονάδα που είναι τώρα στον πίνακα tokenboard έχει το αναγνωριστικό της δεσμευμένης λέξη print και αν ισχύει καλούμε το λεκτικό αναλυτή και τη συνάρτηση expression(). Αν δεν ισχύει ενημερώνουμε το χρήστη με μήνυμα λάθους.

Στη συνάρτηση input\_stat() ελέγχουμε αν η λεκτική μονάδα που πήραμε είναι η δεσμευμένη λέξη input. Αν ισχύει η συνθήκη καλούμε το λεκτικό αναλυτή και ελέγχουμε αν αυτό που θα μας επιστρέψει έχει αναγνωριστικό idtk. Αν ισχύει και αυτή η συνθήκη καλούμε το λεκτικό αναλυτή. Σε περίπτωση που δεν ισχύει κάποια από τις συνθήκες ενημερώνουμε το χρήστη με αντίστοιχο μήνυμα.

Στη συνάρτηση actualpars() αρχικά ελέγχουμε αν η λεκτική μονάδα που έχουμε τώρα στον πίνακα είναι αριστερή παρένθεση. Αν δεν είναι ενημερώνουμε το χρήστη με μήνυμα λάθους. Αν είναι καλούμε το λεκτικό αναλυτή και τη συνάρτηση actualparlist(). Όταν επιστρέψουμε από την actualparlist κάνουμε έναν έλεγχο για δεξιά παρένθεση. Αν δεν έχει ο πίνακάς μας δεξιά παρένθεση εμφανίζουμε μήνυμα λάθους. Αν έχει τότε απλά καλούμε το λεκτικό αναλυτή.

Στη συνάρτηση actualparlist() αρχικά καλούμε την actualparitem() ενώ όταν επιστρέψουμε από την actualparitem() έχουμε ένα βρόγχο while που ελέγχει αν ο λεκτικός αναλυτής βρήκε κόμμα και καλεί ξανά το λεκτικό αναλυτή και την actualparitem().

Στη συνάρτηση actualparitem() αρχικά ελέγχουμε αν ο πίνακας μας έχει μέσα τη δεσμευμένη λέξη in ή inout. Αν δεν έχει καμία από αυτές τότε εμφανίζουμε ένα μήνυμα λάθους στο χρήστη. Αν τελικά ο πίνακας μας έχει τη δεσμευμένη λέξη in και το αντίστοιχο αναγνωριστικό τότε καλούμε το λεκτικό αναλυτή και τη συνάρτηση expression(). Αν έχει τη δεσμευμένη λέξη inout και το αντίστοιχο αναγνωριστικό τότε καλούμε το λεκτικό αναλυτή και ελέγχουμε αν στη δεύτερη θέση του πίνακα πέρασε αναγνωριστικό τύπου idtk. Αν πέρασε τέτοιου είδους αναγνωριστικό τότε καλεί πάλι το λεκτικό αναλυτή. Αλλιώς πετάει μήνυμα λάθους στο χρήστη.

Στη συνάρτηση condition() αρχικά καλούμε τη συνάρτηση boolterm(). Στη συνέχεια έχουμε έναν βρόγχο while ο οποίος θα εκτελείται όσο παίρνουμε αναγνωριστικό της δεσμευμένης λέξης or. Μέσα στη while καλείται ο λεκτικός αναλυτής και η boolterm().

Στη συνάρτηση boolterm() καλούμε τη boolfactor(). Μετά την εκτέλεση της boolfactor έχουμε ένα βρόγχο while όπου ελέγχουμε αν στον πίνακα tokenboard έχουμε τη δεσμευμένη λέξη and και το αναγνωριστικό της. Αν ισχύει η συνθήκη καλούμε το λεκτικό αναλυτή και τη συνάρτηση boolfactor().

Στη συνάρτηση boolfactor() έχουμε τις εξής περιπτώσεις:

* Αν στον πίνακα tokenboard έχουμε τη δεσμευμένη λέξη not καλούμε το λεκτικό αναλυτή και ελέγχουμε αν αυτό που μας επέστρεψε είναι αριστερή παρένθεση. Αν δεν είναι αριστερή παρένθεση θα ενημερώσουμε το χρήστη με μήνυμα λάθους. Αν είναι καλούμε το λεκτικό αναλυτή και τη συνάρτηση condition() και στη συνέχεια ελέγχουμε πάλι αν ο λεκτικός αναλυτής μας επέστρεψε αριστερή παρένθεση. Αν δεν μας επέστρεψε αριστερή παρένθεση εμφανίζουμε μήνυμα λάθους.
* Αν στον πίνακα tokenboard έχουμε αριστερή παρένθεση τότε καλούμε το λεκτικό αναλυτή και την condition() .Στη συνέχεια ελέγχουμε αν βρήκαμε δεξιά παρένθεση. Αν δεν βρήκαμε ενημερώνουμε το χρήστη με μήνυμα λάθους.
* Αν στον πίνακα tokenboard έχουμε τη δεσμευμένη λέξη true. Αν ισχύει η συνθήκη καλούμε το λεκτικό αναλυτή.
* Αν στον πίνακα tokenboard έχουμε τη δεσμευμένη λέξη false. Αν ισχύει η συνθήκη καλούμε το λεκτικό αναλυτή.
* Αν δεν ισχύει καμία από τις παραπάνω συνθήκες τότε καλείται η expression(), η relational\_oper() και πάλι η expression().

Στη συνάρτηση expression() αρχικά καλούνται οι συναρτήσεις optional\_sign() και term(). Στη συνέχεια έχουμε ένα βρόγχο while όπου ελέγχουμε αν βρήκαμε το αναγνωριστικό των πράξεων πρόσθεσης και αφαίρεσης. Αν ισχύει η συνθήκη της while καλούνται οι συναρτήσει add\_oper() και term().

Στη συνάρτηση term() καλούμε τη συνάρτηση factor() και έχουμε μία δομή while στην οποία έχουμε θέσει ως συνθήκη αν στη δεύτερη θέση του πίνακα tokenboard έχουμε αναγνωριστικό διαίρεσης ή πολλαπλασιασμού. Αν η συνθήκη ισχύει τότε καλούνται οι εξής συναρτήσεις: mul\_oper() και factor().

Στη συνάρτηση factor() ελέγχουμε αν στον πίνακα tokenboard έχουμε αριστερή παρένθεση ή idtk ή σταθερά. Αν δεν έχουμε τίποτα από όλα αυτά τότε ενημερώνουμε το χρήστη με μήνυμα λάθους. Αν έχουμε αριστερή παρένθεση καλούμε το λεκτικό αναλυτή και τη συνάρτηση expression(). Στη συνέχεια ελέγχουμε αν βρήκαμε δεξιά παρένθεση και αν ισχύει η συνθήκη καλούμε το λεκτικό αναλυτή, αλλιώς τυπώνουμε μήνυμα λάθους. Αν αντί της παρενθέσεως βρούμε idtk τότε καλούμε το λεκτικό αναλυτή και τη συνάρτηση idtail(). Αν βρούμε μία σταθερά τότε καλούμε το λεκτικό αναλυτή.

Στην idtail() ελέγχουμε αν ο πίνακας μας έχει μέσα κάτι από τα εξής σύμβολα:

* =
* <=
* >=
* >
* <
* <>

Αν δεν έχει κανένα από αυτά τα σύμβολα ενημερώνουμε το χρήστη εμφανίζοντας μήνυμα λάθους. Αν επιβεβαιώνει οποιαδήποτε από τις συνθήκες τότε καλούμε το λεκτικό αναλυτή.

Στην add\_oper() ελέγχουμε αν στον πίνακα tokenboard έχουμε το σύμβολο + ή - και τα αντίστοιχα αναγνωριστικά. Αν έχει κάποιο από τα δύο σύμβολα τότε καλούμε το λεκτικό αναλυτή αλλιώς τυπώνουμε μήνυμα λάθους.

Στην add\_oper() ελέγχουμε αν στον πίνακα tokenboard έχουμε το σύμβολο \* ή / και τα αντίστοιχα αναγνωριστικά. Αν έχει κάποιο από τα δύο σύμβολα τότε καλούμε το λεκτικό αναλυτή αλλιώς τυπώνουμε μήνυμα λάθους.

Στην optional\_sign() ελέγχουμε αν στον πίνακα tokenboard έχουμε το σύμβολο + ή – και καλούμε τη συνάρτηση add\_oper().

**2.3** Επόμενο στάδιο της κατασκευής του μεταφραστή μας είναι η παραγωγή του ενδιάμεσου κώδικα. Αυτό το στάδιο αποτελείται από την δημιουργία ενός συνόλου από πεντάδες(πίνακες πέντε θέσεων) οι οποίες αποτελούνται από ένα αναγνωριστικό, έναν τελεστή και τρία τελούμενα. Το αναγνωριστικό, το οποίο υπάρχει πάντα στην πρώτη θέση του πίνακα, χαρακτηρίζει την κάθε πεντάδα και είναι ένας αύξων αριθμός. Κάθε φορά που εκτελείται μια πεντάδα, αμέσως μετά εκτελείται αυτή με τον αμέσως μεγαλύτερο αριθμό στο αναγνωριστικό της, εκτός αν υπάρχει κάποια που να περιέχει άλμα σε κάποια άλλη συγκεκριμένη .

Για να υλοποιηθεί όμως αυτή η κατασκευή των πινάκων χρειάστηκε να δημιουργήσουμε κάποιες συναρτήσεις ,οι οποίες καλούνται σε συγκεκριμένα σημεία του συντακτικού αναλυτή, ο οποίος με την σειρά του υπέστη κάποιες μικρές αλλαγές. Συγκεκριμένα:

-Αρχικά οι συναρτήσεις που χρειάστηκαν ήταν:

-H **nextquad()** η οποία δημιουργεί και επιστρέφει ένα νέο αριθμό αυξημένο κατά ένα από αυτόν της προηγούμενης για το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί.

-Η **genquad(quad\_id,op,x,y,z)** η οποία όταν καλείται παίρνει σαν όρισμα τα περιεχόμενα μιας πεντάδας, καλεί την nextquad() για να δώσει νέο αναγνωριστικό σε αυτή και αφού δημιουργεί τον πίνακα με αυτά τα περιεχόμενα τον προσθέτει στον πίνακα quad ο οποίος μετά το τέλος της παραγωγής του ενδιάμεσου κώδικα θα περιέχει όλες τις πεντάδες που έχουν δημιουργηθεί.

-Η **backpatch(lista,z)** η οποία παίρνει σαν όρισμα μία λίστα από αναγνωριστικά πεντάδων των οποίων το τελευταίο τελούμενο στους δεν είναι συμπληρωμένο και ένα τελούμενο προς συμπλήρωση. Ψάχνει τον αρχικό πίνακα quad συγκρίνοντας τα αναγνωριστικά των πεντάδων του με αυτά της λίστας του ορίσματος αν βρει κάποιο ίδιο πάει και προσθέτει το τελούμενο του ορίσματος στην τελευταία του θέση.

-Η **newTemp()** η οποία δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή της μορφής 'T\_(num)' ,όπου num ένας αύξων αριθμός που ξεκινάει από το μηδέν και αυξάνεται κάθε φορά κατά ένα.

-Η **emptylist()** η οποία δημιουργεί και επιστρέφει μία κενή λίστα ετικετών πεντάδων.

-Η **makelist(x) ()** η οποία δημιουργεί και επιστρέφει μία λίστα ετικετών πεντάδων που περιέχει μόνο το στοιχείο του ορίσματος.

-Η **mergelist(list1,list2)** η οποία παίρνει σαν όρισμα δύο λίστες και δημιουργεί μια καινούργια από την συνένωση τους.

Τώρα για την δημιουργία των κατάλληλων πεντάδων κατά την εκτέλεση του προγράμματος αυτές οι συναρτήσεις καλέστηκαν στα κατάλληλα σημεία του κώδικα του συντακτικού αναλυτή. Συγκεκριμένα:

-Στην συνάρτηση **block(name)**, η οποία παίρνει πλέον σαν όρισμα το όνομα που έχει το ίδιο το πρόγραμμα όταν καλείται στην **program()** καθώς και κάθε συνάρτηση ή διαδικασία που περιλαμβάνει όταν καλείται στην **procorfuncbody(name)** και με βάση αυτό αρχικά δημιουργεί καλώντας την συνάρτηση genquad(quad\_id,op,x,y,z), μια πεντάδα της μορφής (quad\_id,'begin\_block',name,' ',' ') που δηλώνει την αρχή της συνάρτησης ή του προγράμματος και αμέσως μετά την επιστροφή από τις συναρτήσεις που καλούνται για την δόμη τους δημιουργεί καλώντας ξανά την genquad(quad\_id,op,x,y,z), μια πεντάδα της μορφής (quad\_id,'end\_block',name,' ',' ') που δηλώνει το τέλος της/του. Στην περίπτωση του κύριου προγράμματος πριν από την δημιουργία της πεντάδας που δηλώνει το τέλος του δημιουργεί μια ακόμα της μορφής (quad\_id,'halt',' ',' ',' ') ξανά με την genquad(quad\_id,op,x,y,z).

-Στην συνάρτηση **assigment\_stat(),** η οποία καλώντας την genquad(quad\_id,op,x,y,z) δημιουργεί μια πεντάδα της μορφής (quad\_id,k,eplace,'',t) όπου k ο τελεστής της ανάθεσης ,eplace το αποτέλεσμα που επιστρέφει πλέον η expression() διότι μπορεί να έχουμε παραπάνω από μια διαφορετικές περιπτώσεις ανάθεσης και t το τελούμενο στο οποίο βάζουμε το αποτέλεσμα αυτής.

-Στην συνάρτηση **if\_stat(),** η οποία καλώντας την condition() λαμβάνει πλέον δυο λίστες με αναγνωριστικά, μια για την περίπτωση που ισχύει η συνθήκη της και μία για την αντίθετη και με βάση αυτές στην πρώτη περίπτωση καλεί την backpatch(condtrue,nextquad()) με όρισμα την πρώτη λίστα και το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί μέσα από την statements() και μετά δημιουργεί μια λίστα που περιέχει μόνο το αναγνωριστικό της επόμενης πεντάδας με την makelist(nextquad()).Επίσης δημιουργεί ένα τρύπιο 'jump' με την genquad(quad\_id,'jump','','','') και μετά συμπληρώνει τα κενά των πεντάδων με αναγνωριστικά ίδια με αυτά τις δεύτερης λίστας που έλαβε, με το αναγνωριστικό της επόμενης λίστας που θα δημιουργηθεί μέσα στο else\_part().Τέλος συνενώνει τις δύο λίστες με που πλέον λαμβάνει από την statements() και else\_part() σε μια καινούργια με την mergelist(t1,t2) την οποία επιστρέφει κιόλας και συμπληρώνει τα τρύπια 'jump' με την backpatch(iflist,nextquad()).

-Στην συνάρτηση **repeat\_stat()**, η οποία δημιουργεί ένα 'jump' στην επόμενη πεντάδα που θα δημιουργηθεί μέσα από την statements() με την genquad(quad\_id,'jump','','',sQuad) όπου sQuad=nextquad() . Επίσης δημιουργεί μια λίστα από την συνένωση της ίδιας με αυτήν που έλαβε από την statements() (exitlist=mergelist(exitlist,t)) για την έξοδο και μόλις τελειώσει το repeat συμπληρώνει τα κενά 'jump' των πεντάδων της εξόδου με την επόμενη πεντάδα που θα δημιουργηθεί (backpatch(exitlist,nextquad())).

-Στην συνάρτηση **exit\_stat()**, η οποία δημιουργεί μια λίστα με μόνο στοιχείο το αναγνωριστικό την επόμενης πεντάδας που θα δημιουργηθεί με την t=makelist(nextquad()) την οποία επιστρέφει κιόλας και αμέσως μετά δημιουργεί ένα τρύπιο 'jump' το οποίο θα συμπληρωθεί από εκεί που καλέστηκε αργότερα.

-Στην συνάρτηση **while\_stat()**, η οποία δημιουργεί μια λίστα που περιέχει μόνο το αναγνωριστικό της επόμενης πεντάδας με την makelist(nextquad()) για να ξέρει που να γυρίζει κάθε φορά. Καλεί την condition() και λαμβάνει δυο λίστες με αναγνωριστικά, μια για την περίπτωση που ισχύει η συνθήκη της και μία για την αντίθετη και με βάση αυτές στην πρώτη περίπτωση καλεί την backpatch(condtrue,nextquad()) με όρισμα την πρώτη λίστα και το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί μέσα από την statements() και δημιουργεί ένα τρύπιο 'jump' με την genquad(quad\_id,'jump','','',''). Μετά συμπληρώνει τα κενά των πεντάδων με αναγνωριστικά ίδια με αυτά τις δεύτερης λίστας που έλαβε, με το αναγνωριστικό της επόμενης λίστας που θα δημιουργηθεί .Τέλος επιστρέφει την λίστα που έλαβε από την statements() για να την χρησιμοποιήσει στην συνέχεια όπου χρειαστεί.

-Στην συνάρτηση **switch\_stat()**, η οποία αρχικά καλεί την expression() και αποθηκεύει το αποτέλεσμά της σε μία μεταβλητή eplace. Όταν ξεκινήσουν τα cases ακλουθεί πάλι την ίδια διαδικασία χρησιμοποιώντας τώρα μία δεύτερη μεταβλητή eplace1. Στην συνέχεια δημιουργεί μία λίστα με το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί για την περίπτωση του διάφορου, δηλαδή της genquad(quad\_id,'<>',eplace,eplace1,'') και μια για την περίπτωση του ίσου, δηλαδή της genquad(quad\_id,'=',eplace,eplace1,''). Αμέσως μετά και στην περίπτωση που ισχύει το ίσο συμπληρώνει τις πεντάδα του ίσου με το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί από την statements() που θα το εκτελέσει και μετά το πέρας αυτής θα δημιουργήσει μια λίστα με το αναγνωριστικό της λίστας που θα δημιουργηθεί αμέσως μετά και θα δηλώνει την έξοδο από την διαδικασία (exitlist=makelist(nextquad())), δηλαδή της genquad(quad\_id,'jump','','','') και μετά θα συμπληρώσει τις λίστες του διάφορου για να πάει να ελέγξει και τα υπόλοιπα cases. Την ίδια διαδικασία θα επαναλαμβάνει μέχρι και το τελευταίο case και μετά το πέρας αυτού θα δημιουργήσει μία λίστα από την συνένωση των λιστών της εξόδου από τα cases την οποία θα χρησιμοποιήσει όταν τελικά τελειώσει η switch για να τις συμπληρώσει με την επόμενη πεντάδα που θα δημιουργηθεί αμέσως μετά. Τέλος δημιουργεί και επιστρέφει μια λίστα από την συνένωση των λιστών που προήλθαν από τις statements() που καλέστηκαν μέσα στα cases για να την χρησιμοποιήσει στην συνέχεια εκεί που χρειάζεται.

-Στην συνάρτηση **forcase\_stat(),** η οποία δημιουργεί μια λίστα που περιέχει μόνο το αναγνωριστικό της επόμενης πεντάδας με την makelist(nextquad()) για να ξέρει που να γυρίζει κάθε φορά. Καλεί την condition() και λαμβάνει δυο λίστες με αναγνωριστικά, μια για την περίπτωση που ισχύει η συνθήκη της και μία για την αντίθετη και με βάση αυτές στην πρώτη περίπτωση καλεί την backpatch(condtrue,nextquad()) με όρισμα την πρώτη λίστα και το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί μέσα από την statements() και δημιουργεί ένα 'jump' ξανά στην αρχή με την genquad(quad\_id,'jump','','',fc). Μόλις γυρίσει από εκεί συμπληρώνει τα κενά των πεντάδων με αναγνωριστικά ίδια με αυτά τις δεύτερης λίστας που έλαβε, με το αναγνωριστικό της επόμενης λίστας που θα δημιουργηθεί, δηλαδή αυτής έξω από την επανάληψη. Συνεχίζει την ίδια διαδικασία όσο έχει ακόμα επαναλήψεις και μετά δημιουργεί και επιστρέφει μια λίστα απο την συνένωση των λιστών που προήλθαν από τις statements() που καλέστηκαν για να την χρησιμοποιήσει στην συνέχεια εκεί που χρειάζεται και τέλος μετά το πέρας της τελευταίας επανάληψης βγαίνει έξω από αυτή.

-Στην συνάρτηση **call\_stat()**, η οποία δημιουργεί μία πεντάδα για την κλήση μιας συνάρτησης ( genquad(quad\_id,'call',x,'','') όπου x το όνομα αυτής).

-Στην συνάρτηση **return\_stat()**, η οποία δημιουργεί μία πεντάδα για την οτιδήποτε θέλουμε να επιστρέψουμε, το οποίο προέρχεται μέσα από την expression() ( (genquad(quad\_id,'RET','','',eplace) όπου eplace το αποτέλεσμα της expression).

-Στην συνάρτηση **print\_stat()**, η οποία δημιουργεί μία πεντάδα για αυτό που θέλουμε να τυπώσουμε στην οθόνη το οποίο προέρχεται μέσα από την expression()( genquad(quad\_id,'out',eplace,'','') ) όπου eplace το αποτέλεσμα της expression).

-Στην συνάρτηση **input\_stat()**, η οποία δημιουργεί μία πεντάδα για αυτό που πάρουμε από το πληκτρολόγιο (( genquad(quad\_id,'inp',x,'','') όπου x αυτό που διαβάσαμε.

-Στην συνάρτηση **actualparitem()**, η οποία δημιουργεί τις πεντάδες για τις παραμέτρους των συναρτήσεων και των διαδικασιών. Αν έχουμε πέρασμα παραμέτρου με τιμή τότε χρησιμοποιεί την genquad(quad\_id,'par',x,'CV','') όπου x το όνομα της παραμέτρου, αλλιώς αν έχουμε πέρασμα με αναφορά χρησιμοποιεί την genquad(quad\_id,'par',tokenboard[0],'REF','').

-Στην συνάρτηση **condition()**, η οποία καλεί την boolterm(), η οποία με την σειρά της καλεί την boolfactor() και της επιστρέφει δυο λίστες με αναγνωριστικά, μια για την περίπτωση που ισχύει η συνθήκη της και μία για την αντίθετη. Στην συνέχεια όσο βρίσκει or συμπληρώνει τις πεντάδες που δηλώνουν ότι δεν ισχύει η συνθήκη με την επόμενη πεντάδα που θα δημιουργηθεί, μετά ακολουθεί την ίδια διαδικασία με πριν και στο τέλος δημιουργεί μία λίστα από την συνένωση των λιστών με τα αναγνωριστικά των λιστών για τις συνθήκες που ισχύουν και την επιστρέφει μαζί με την αντίστοιχη για τις συνθήκες που δεν ισχύουν.

-Στην συνάρτηση **boolterm()**, η οποία καλεί την boolfactor() και της επιστρέφει δυο λίστες με αναγνωριστικά, μια για την περίπτωση που ισχύει η συνθήκη της και μία για την αντίθετη. Στην συνέχεια όσο βρίσκει and συμπληρώνει τις πεντάδες που δηλώνουν ότι ισχύει η συνθήκη με την επόμενη πεντάδα που θα δημιουργηθεί, μετά ακολουθεί την ίδια διαδικασία με πριν και στο τέλος δημιουργεί μία λίστα από την συνένωση των λιστών με τα αναγνωριστικά των λιστών για τις συνθήκες που δεν ισχύουν και την επιστρέφει μαζί με την αντίστοιχη για τις συνθήκες που ισχύουν.

-Στην συνάρτηση **boolfactor()**, η οποία ελέγχει μερικές περιπτώσεις σε κάθε μία από τις οποίες κάνει διαφορετικά πράγματα. Δηλαδή:

-Στην περίπτωση που βρει not ή άνοιγμα παρένθεσης καλεί την condition() κρατάει τις δυο λίστες με αναγνωριστικά, μια για την περίπτωση που ισχύει η συνθήκη της και μία για την αντίθετη που αυτή τις επιστρέφει.

-Στην περίπτωση που βρει true ή false δεν επιστρέφει τίποτα.

-Στην περίπτωση που δεν βρει τίποτα από τα παραπάνω καλεί την expression() και κρατάει το αποτέλεσμά της σε μία μεταβλητή, μετά καλεί την relational\_oper() και κρατάει το σύμβολο που της επιστρέφει σε μια μεταβλητή και μετά ξανά την expression() χρησιμοποιώντας μια τρίτη μεταβλητή. Στην συνέχεια δημιουργεί μία λίστα με το αναγνωριστικό της επόμενης πεντάδας που θα δημιουργηθεί για την περίπτωση που ισχύει η σύγκριση Rtrue=makelist(nextquad()), δηλαδή της genquad(quad\_id,relop,eplace1,eplace2,'')και μια για την περίπτωση που δεν ισχύει Rfalse=makelist(nextquad()), δηλαδή της genquad(quad\_id,'jump','','','') οι οποίες συμπληρώνονται στην συνέχεια.Τελικά επιστρέφει αυτές τις δύο λίστες με τα αναγνωριστικά με την σειρά που τις δημιούργησε.

-Στην συνάρτηση **expression()**, η οποία καλεί την term()και κρατάει το σύμβολο που της επιστρέφει σε μία μεταβλητή. Στην συνέχεια όσο βρίσκει + ή - καλεί την relational\_oper() και την factor() και κρατάει το σύμβολο που της επιστρέφει η πρώτη και την έκφραση που της επιστρέφει η δεύτερη σε δύο μεταβλητές. Μετά δημιουργεί μια προσωρινή μεταβλητή με την newTemp() και αμέσως μετά μια πεντάδα genquad(quad\_id,operation,fplace1,fplace2,w) για την αποθήκευση του αποτελέσματος σε αυτή. Τελικά επιστρέφει το τελικό αποτέλεσμα.

-Στην συνάρτηση **factor()** η οποία ελέγχει μερικές περιπτώσεις σε κάθε μία από τις οποίες κάνει διαφορετικά πράγματα. Δηλαδή:

-Στην περίπτωση που βρει άνοιγμα παρένθεσης καλεί την expression() και κρατάει το αποτέλεσμά της σε μία μεταβλητή την οποία στο τέλος επιστρέφει.

-Στην περίπτωση που βρει κάποιο όνομα ελέγχει αν πρόκειται για συνάρτηση και αν ναι δημιουργεί μία πεντάδα της μορφής genquad(quad\_id,'call',fplace,'','') όπου fplace το όνομα της συνάρτησης και μετά επιστρέφει την προσωρινή μεταβλητή με το αποτέλεσμα της.

-Στην περίπτωση που βρει κάποια σταθερά απλά την επιστρέφει.

-Στην συνάρτηση **idtail()**, η οποία δημιουργεί και επιστρέφει μια προσωρινή μεταβλητή για την αποθήκευση του αποτελέσματος από το κάλεσμα μίας συνάρτησης.( w=newTemp() , genquad(quad\_id,'par',w,'RET','')).

**2.4.** Το τέταρτο στάδιο στη δημιουργία ενός μεταφραστή βρίσκεται η υλοποίηση του πίνακα συμβόλων. Ο πίνακας συμβόλων είναι μία δομή δεδομένων που χρησιμοποιείται ώστε να κρατάμε τη δομική πληροφορία του προγράμματος. Η πληροφορία αυτή συλλέγεται από τις φάσεις ανάλυσης ενός μεταφραστή και χρησιμοποιείται από τις φάσεις σύνθεσης για την παραγωγή του τελικού κώδικα.

Όσον αφορά την υλοποίηση αυτής της δομής δεδομένων πρέπει αποσαφηνίσουμε κάποιες έννοιες που είναι αναγκαίες για τη συνέχεια. Όταν αναφερόμαστε σε Scope θα έχουμε στο μυαλό μας μία δομή η οποία κρατάει τις εξής πληροφορίες: μία λίστα από οντότητες (Entities), δείκτης στη στοίβα και βάθος φωλιάσματος. Μία οντότητα (Entity) μπορεί να είναι μία μεταβλητή ή παράμετρος ή συνάρτηση ή προσωρινή μεταβλητή. Για καθεμία από αυτές τις οντότητες έχουμε ένα σύνολο από πληροφορίες που πρέπει να κρατάμε. Επίσης έχουμε και τα περάσματα συναρτήσεων (Arguments) τα οποία πρέπει να κρατάνε τις αντίστοιχες πληροφορίες.

Πιο συγκεκριμένα στην υλοποίηση μας έχουμε ορίσει ένα σύνολο από κλάσεις προκειμένου να είναι πιο ξεκάθαρη η υλοποίηση του πίνακα συμβόλων αλλά και πιο ευκατανόητες οι δομές που χρησιμοποιούμε για να αποθηκεύσουμε τις απαραίτητες πληροφορίες.

Κλάσεις:

Scope(): Έχει τρία πεδία: Entities, nestingLevel,offset.

Entities: Είναι μία λίστα που αποτελείται από αντικείμενα τύπου Entity.

nestingLevel: Ένας ακέραιος αριθμός που δείχνει το βάθος φωλιάσματος στο οποίο βρίσκεται το συγκεκριμένο Scope.

Offset: Απόσταση από την κορυφή της στοίβας.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Scope για να αρχικοποιήσουμε τα πεδία.

Μία μέθοδο EnterEntity η οποία προσθέτει στο τέλος της λίστα Entities ένα αντικείμενο τύπου entity.

Μία μέθοδο get\_offset η οποία επιστρέφει το offset του αντικειμένου Scope και το αυξάνει κατά τέσσερα.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

Entity(): Έχει τρία πεδία: name, typeof, nextEntity.

Name: Το όνομα της οντότητας.

typeof: Τύπος οντότητας, ένα String. Μπορεί να είναι var για μεταβλητές, par για παραμέτρους,func για συναρτήσεις,temp για προσωρινές μεταβλητές.

nextEntity: Ένας δείκτης στην επόμενη οντότητα.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Entity για να αρχικοποιήσουμε τα πεδία.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

Variable(Entity): Έχει ένα πεδίο offset και αυτά που κληρονομεί από

τη γονική κλάση Entity

Offset:Απόσταση από την κορυφή της στοίβας.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Variable για να αρχικοποιήσουμε τα πεδία. Μέσα σε αυτό τον Constructor καλείται και ο constructor της γονική κλάσης με τα απαραίτητα πεδία name και typeof= 'var'.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

Parameter (Entity): Έχει ένα πεδίο offset και ένα parMode και αυτά που

κληρονομεί από τη γονική κλάση Entity.

Offset: Απόσταση από την κορυφή της στοίβας.

ParMode: Κρατάει τον τρόπο περάσματος της παραμέτρου. Με τιμή ή αναφορά.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Parameter για να αρχικοποιήσουμε τα πεδία. Μέσα σε αυτό τον Constructor καλείται και ο constructor της γονική κλάσης με τα απαραίτητα πεδία name και typeof= 'par'.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

Function (Entity): Έχει τρία πεδία: FL, SQ, arglist και αυτά που

κληρονομεί από τη γονική κλάση Entity.

FL: Αριθμός ,κρατάει την τιμή του framelength.

SQ: Όπου κρατάει την πεντάδα από την οποία ξεκινάει να εκτελείται η συνάρτηση.

arglist: Η λίστα των παραμέτρων της συνάρτησης.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Function για να αρχικοποιήσουμε τα πεδία. Μέσα σε αυτό τον Constructor καλείται και ο constructor της γονική κλάσης με τα απαραίτητα πεδία name και typeof= ‘func’.

Mία μέθοδο enterArg η οποία παίρνει ως όρισμα ένα αντικείμενο τύπου arg και το προσθέτει στο τέλος της λίστας arglist.

Μία συνάρτηση setFL η οποία παίρνει ένα όρισμα FL και το θέτει στο πεδίο FL

Mία μέθοδο setSQ η οποία παίρνει σαν όρισμα ένα quad\_id, μία πεντάδα, και το θέτει στο πεδίο SQ.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

TempVar (Entity): Έχει ένα πεδίο offset και αυτά που κληρονομεί από τη

γονική κλάση Entity.

Offset: Απόσταση από την κορυφή της στοίβας.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Parameter για να αρχικοποιήσουμε τα πεδία. Μέσα σε αυτό τον Constructor καλείται και ο constructor της γονική κλάσης με τα απαραίτητα πεδία name και typeof=’ temp’.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

Argument (): Έχει ένα πεδίο parMode και ένα Argnext

Argnext: Δείκτης στο επόμενο αντικείμενο τύπου Argument.

ParMode: Κρατάει τον τρόπο περάσματος της παραμέτρου. Με τιμή ή αναφορά.

Έναν constructor \_\_init\_\_ τον οποίο χρησιμοποιούμε όταν φτιάχνουμε ένα νέο αντικείμενο τύπου Parameter για να αρχικοποιήσουμε τα πεδία. Μέσα σε αυτό τον Constructor καλείται και ο constructor της γονική κλάσης με τα απαραίτητα πεδία name και typeof= 'par'.

Μία μέθοδο Set\_next που παίρνει σαν όρισμα ένα αντικείμενο τύπου Argument και το θέτει στο πεδίο Argnext.

Μία μέθοδο \_\_str\_\_ η οποία επιστρέφει όλες τις πληροφορίες του αντικειμένου σε Sting για να μπορούμε να τις εκτυπώσουμε εύκολα. Ουσιαστικά είναι όπως η toString σε java υλοποιημένη σε python. Δεν ήταν σε καμία περίπτωση αναγκαία απλά την υλοποιήσαμε για δική μας ευκολία στο debugging.

Επίσης για ευκολία στην υλοποίηση του πίνακα συμβόλων υλοποίησαμε και ένα σύνολο συναρτήσεων:

addNewScope(): Δημιουργεί ένα νέο αντικείμενο τύπου Scope και το προσθέτει στο τέλος της λίστας με όλα τα Scope, δηλαδή τη scopes.

deleteScope(): Αφαιρεί το τελευταίο στοιχείο της λίστας scopes.

addEntity(): Παίρνει ως παραμέτρους ένα όνομα (name), έναν τύπο (typeof) και ένα τρόπο περάσματος παραμέτρου (parMode) και ανάλογα με το τι είναι το typeof δημιουργεί ένα αντικείμενο τύπου Entity με offset ίσο με το offset του τελευταίου entity,του τελευταίου Scope στη λίστα scopes και καλεί την EnterEntity με αυτό το αντικείμενο που δημιούργησε ώστε να το προσθέσει στο τέλος της λίστας με τα Entities του τελευταίου Scope.

addArgument(): Παίρνει ως όρισμα ένα όνομα και ένα τύπο τρόπο περάσματος παραμέτρου και ανάλογα με τον τρόπο περάσματος παραμέτρου δημιουργεί ένα αντικείμενο τύπου Argument καλεί τη SearchEntity ώστε να βρει αν υπάρχει ήδη κάπου δηλωμένο ξανά το ίδιο αντικείμενο και το προσθέτει στο συγκεκριμένο entity που θα γυρίσει η SearchΕntity το καινούριο αντικείμενο που δημιούργησε.

SearchEntity():Παίρνει ένα όνομα (name) και έναν τύπο (typeof) και ελέγχει σε όλα τα Entities όλων των scopes στη λίστα scope αν υπάρχει κάποια οντότητα με ίδιο όνομα και τύπο. Αν υπάρχει επιστρέφει πίσω το entity και το βάθος φωλιάσματος που είναι το entity. Αν δεν υπάρχει πουθενά ενημερώνει το χρήστη με μήνυμα λάθους.

SearchEntity\_1(): Παίρνει ένα όνομα (name) και ψάχνει σε όλα τα Entities όλων των Scope στη λίστα scopes από το Scope με το μικρότερο βάθος φωλιάσματος προς το μεγαλύτερο. Αν βρει οντότητα με ίδιο όνομα την επιστρέφει και μαζί επιστρέφει και το nestingLevel. Αν δεν βρει τότε τυπώνει μήνυμα λάθους στο χρήστη.

addFL():Παίρνει σαν όρισμα ένα όνομα και έναν FL (αριθμός) και αν το όνομα είναι ίδιο με το όνομα του προγράμματος θέτει τη μεταβλητή mainFL, που είναι το framelength της main,ίσο με το FL. Αλλιώς καλεί τη SearchEntity και θέτει το FL του entity που θα επιστρέψει αυτή.

AddSQ(): Παίρνει σαν όρισα ένα όνομα και ένα αναγνωριστικό πεντάδας και αν το όνομα είναι το όνομα της main τότε θέτει τη μεταβλητή mainSQ, που είναι το η πεντάδα που ξεκινάει η main, ίσο με quad\_id. Αλλιώς καλεί τη SearchEntity και θέτει το SQ του entity που θα επιστρέψει αυτή.

Τώρα που έχουμε υλοποιήσει το σύνολο όλων αυτών το συναρτήσεων και κλάσεων πρέπει να τις καλέσουμε στα κατάλληλα σημεία μέσα στο συντακτικό αναλυτή ώστε να δημιουργηθεί σωστά ο πίνακάς μας.

Στην program όταν πλέον βρούμε και το όνομα του προγράμματός μας καλούμε τη newScope και την προσθέτουμε στην λίστα μας με τα Scopes που ονομάζεται scopes και είναι global.

Στη block αφού έχουμε δημιουργήσει την πεντάδα που συμβολίζει την αρχή του block καλούμε την AddSQ().Μετά τη statements() έχουμε έναν έλεγχο αν το name που πήραμε ως όρισα στη block είναι ίδιο με το όνομα της main μας (το όνομα της main είναι δηλωμένο global οπότε έχουμε πρόσβαση σε αυτό οποιαδήποτε στιγμή).Στον έλεγχο αυτό αν αληθεύει η συνθήκη τότε σημαίνει πως τελειώσαμε και με τη main οπότε θέλουμε να υπολογίσουμε το τελικό FL γι’ αυτό και καλούμε την addFL(). Αν δεν αληθεύει η συνθήκη τότε θα καλέσουμε την addFL και στη συνέχεια θα διαγράψουμε το συγκεκριμένο Scope από τη scopes.

Στη varlist αφού βρούμε το idtk της μεταβλητής καλούμε τη συνάρτηση addEntity για να προσθέσουμε την οντότητα. Το ίδιο γίνεται και μέσα στο while αφού βρούμε το αναγνωριστικό για το κόμμα και το idtk.

Στην procorfunc αφού βρούμε το αναγνωριστικό διαδικασίας ή συνάρτησης και το idtk για το όνομα, καλούμε την addEntity και την AddNewScope.

Στη formalparitem αφού βρούμε τον τύπο περάσματος της παραμέτρου και το όνομα της καλούμε την addEntity και την addArgument.

Επίσης χρειάστηκε να προσθέσουμε τη συνάρτηση addEntity στη συνάρτηση newTemp που είχαμε φτιάξει για τον ενδιάμεσο κώδικα. Την προσθέσαμε πριν το return για να φτιάχνει για τις προσωρινές μεταβλητές entity.

**2.5** Πλέον είμαστε έτοιμοι να περάσουμε στη φάση σύνθεσης και παραγωγής τελικού κώδικα. Για να το κάνουμε αυτό κρίνεται αναγκαίο να υλοποιήσουμε κάποιες νέες συναρτήσεις που θα χρησιμοποιήσουμε αποκλειστικά για αυτό το στάδιο. Οι νέες συναρτήσεις που δημιουργήσαμε είναι οι εξής: ASMcode(), gnvlcode(), loadvr(), storerv().

Οι gnvlcode(), loadvr() και storerv() καλούνται μέσα στη συνάρτηση ASMcode. Ας τα πάρουμε όμως από την αρχή.

Η ΑSMcode καλείται στο τέλος της block ακριβώς πριν διαγράψουμε το Scope της. Της περνάμε ως παράμετρο την λίστα quad και το όνομα του block. Το quad περιέχει τις πεντάδες που έχουν ήδη δημιουργηθεί μέχρι εκείνη τη στιγμή. Στην ΑSMcode αρχικά ελέγχουμε αν το όνομα που περάσαμε ως όρισμα είναι ίδιο με το όνομα του προγράμματός μας. Αν ισχύει η συνθήκη τότε θέτει τη μεταβλητή sq ίσο με το mainSQ οπου sq το Start Quad, αλλιώς καλούμε τη SearchEntity με ορίσματα name και func. To όρισμα func το περνάμε γιατί ενδιαφερόμαστε μόνο για συναρτήσεις ή διαδικασίες. Παίρνουμε από τη SearhEntity το όνομα του entity και με αυτό θέτουμε τα sq ίσο με to SQ του Entity. Για κάθε πεντάδα που υπάρχει στον πίνακα από τη θέση του sq της συνάρτησης ή της διαδικασίας -1, είτε αυτή είναι η main είτε όχι, μέχρι το τέλος του πίνακα εκείνη τη στιγμή, εκτελούμε τα παρακάτω:

Αρχικά γράφουμε στην πρώτη γραμμή του αρχείου μας το Label με το αναγνωριστικό της κάθε πεντάδας. Μετά κάνουμε τους παρακάτω ελέγχους

* Αν έχουμε jump: γράφουμε στο αρχείο το άλμα προς το Label που θέλουμε να πάμε. Για να το κάνουμε αυτό χρησιμοποιήσαμε το τελευταίο στοιχείο της πεντάδας που έχουμε.
* Αν έχουμε ισότητα: καλούμε μία φορά τη συνάρτηση loadvr() με ορίσματα το ένα τελούμενο της πεντάδας που βρίσκεται την θέση δύο με τον καταχωρητή 1 στον οποίο φορτώνουμε την τιμή του και άλλη μία την ίδια με ορίσματα το επόμενο τελούμενο που βρίσκεται στη θέση τρία και τον καταχωρητή 2 επίσης στον οποίο φορτώνουμε την τιμή του . Τέλος γράφουμε στο αρχείο την αντίστοιχη εντολή ελέγχου ισότητας σε Assembly.
* Αν έχουμε διάφορο: ακολουθούμε την ίδια διαδικασία που ακολουθήσαμε στην ισότητα με την διαφορά ότι στο τέλος γράφουμε στο αρχείο την αντίστοιχη εντολή ελέγχου μη ισότητας σε Assembly.
* Αν έχουμε ανισότητα (> , < ,>= , <=): ακολουθούμε πάλι την ίδια διαδικασία με την ισότητα απλά για κάθε μία από αυτές γράφουμε στο αρχείο την αντίστοιχη εντολή της σε Assembly.
* Αν έχουμε ανάθεση : καλούμε τη συνάρτηση loadvr() με ορίσματα το ένα τελούμενο της πεντάδας που βρίσκεται την θέση δύο με τον καταχωρητή 1 στον οποίο φορτώνουμε την τιμή του και αμέσως μετά καλούμε την συνάρτηση storerv() () με ορίσματα τον καταχωρητή 2 στον οποίο περνάμε το αποτέλεσμα και το τελούμενο που βρίσκεται στη θέση τρία της πεντάδας.
* Αν έχουμε αφαίρεση, πρόσθεση, πολ/μο ή διαίρεση τότε καλούμε μία φορά τη συνάρτηση loadvr() με ορίσματα το ένα τελούμενο της πεντάδας που βρίσκεται την θέση δύο με τον καταχωρητή 1 στον οποίο φορτώνουμε την τιμή του και άλλη μία την ίδια με ορίσματα το επόμενο τελούμενο που βρίσκεται στη θέση 3 και τον καταχωρητή 2 επίσης στον οποίο φορτώνουμε την τιμή του .Μετά γράφουμε στο αρχείο την αντίστοιχη εντολή ελέγχου της κάθε μιας σε Assembly και τέλος καλούμε την συνάρτηση storerv() με ορίσματα τον καταχωρητή 2 στον οποίο περνάμε το αποτέλεσμα και το τελούμενο που βρίσκεται στη θέση τρία της πεντάδας.
* Αν έχουμε print: γράφουμε την εντολή με την οποία φορτώνουμε την σταθερά 1 στον καταχωρητή ν0 στο αρχείο, καλούμε την συνάρτηση loadvr() με ορίσματα το τελούμενο της πεντάδας που βρίσκεται την θέση δύο και τον καταχωρητή 1 στον οποίο φορτώνουμε την τιμή του, αμέσως μετά περνάμε την τιμή του στον καταχωρητή α0 και κάνουμε κλήση συστήματος.
* Αν έχουμε input: γράφουμε την εντολή με την οποία φορτώνουμε την σταθερά 5 στον καταχωρητή ν0 στο αρχείο, και αμέσως μετά κάνουμε κλήση συστήματος.
* Αν έχουμε return: καλούμε την συνάρτηση loadvr() με ορίσματα το τελούμενο της πεντάδας που βρίσκεται την θέση τέσσερα και τον καταχωρητή 1 στον οποίο φορτώνουμε την τιμή του. Αμέσως μετά γράφουμε την εντολή με την οποία φορτώνουμε το περιεχόμενο της μνήμης που βρίσκεται οχτώ θέσεις από τον stack pointer( δηλαδή την διεύθυνση επιστροφής που πρέπει να γράψω) στον καταχωρητή t0 στο αρχείο και την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 1 σε αυτή τη θέση μνήμης που περιέχει ο t0 .
* Αν έχουμε παράμετρο: Αν βρισκόμαστε στο αρχικό πρόγραμμα τότε γράφουμε στο αρχείο την εντολή η οποία μετακινεί τον stack pointer κατά το FL της main και αποθηκεύει το αποτέλεσμα στον frame pointer και κάνουμε το level=0 αλλιώς καλούμε την συνάρτηση SearchEntity() με ορίσματα το όνομα της συνάρτησης στην οποία βρισκόμαστε και τύπο ‘συνάρτηση΄ και κρατάμε σε μια μεταβλητή το FL και σε άλλη μια το level της τα οποία μας επιστρέφει και κάνουμε το level ίσο με το επίπεδό της. Μετά ελέγχουμε αν έχουμε παραμέτρους σε αυτή και αν όχι γράφουμε στο αρχείο την εντολή η οποία μετακινεί τον stack pointer κατά το FL της συνάρτησης αυτής και αποθηκεύει το αποτέλεσμα στον frame pointer . Στην συνέχεια βάζουμε την πεντάδα της παραμέτρου στον πίνακα τον παραμέτρων και υπολογίζουμε την απόσταση στην οποία βρισκόμαστε από την αρχή της στοίβας. Αμέσως μετά ελέγχουμε αν έχουμε πέρασμα με αναφορά ή τιμή. Στο πέρασμα με τιμή καλούμε την συνάρτηση loadvr() με ορίσματα το τελούμενο της πεντάδας που βρίσκεται την θέση δύο και τον καταχωρητή 0 στον οποίο φορτώνουμε την τιμή του και μετά γράφουμε στο αρχείο την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση όση η απόσταση που υπολογίσαμε πριν λίγο. Στο πέρασμα με αναφορά αφού πρώτα ψάξουμε την παράμετρο στον πίνακα με την SearchEntity\_1() κάνουμε τους παρακάτω ελέγχους

-Αν έχω μεταβλητή και η καλούσα συνάρτηση και αυτή έχουν το ίδιο βάθος φωλιάσματος τότε γράφουμε στο αρχείο την εντολή με την οποία προσθέτουμε στον καταχωρητή 0 το περιεχόμενο του sp μείον το offset της μεταβλητής. Στην συνέχεια γράφουμε στο αρχείο την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση όση η απόσταση που υπολογίσαμε πριν λίγο.

-Αν έχουμε παράμετρο που έχει περαστεί με τιμή και η καλούσα συνάρτηση και αυτή έχουν το ίδιο βάθος φωλιάσματος τότε γράφουμε στο αρχείο την εντολή με την οποία προσθέτουμε στον καταχωρητή 0 το περιεχόμενο του sp μείον το offset της μεταβλητής. Στην συνέχεια γράφουμε στο αρχείο την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση όση η απόσταση που υπολογίσαμε πριν λίγο.

--Αν έχουμε παράμετρο που έχει περαστεί με αναφορά και η καλούσα συνάρτηση και αυτή έχουν το ίδιο βάθος φωλιάσματος τότε γράφουμε στο αρχείο την εντολή με την οποία φορτώνουμε στον καταχωρητή 0 το περιεχόμενο της μνήμης που απέχει από τον sp απόσταση offset και την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση όση η απόσταση που υπολογίσαμε πριν λίγο.

-Αν έχουμε παράμετρο που έχει περαστεί με τιμή και η καλούσα συνάρτηση και αυτή έχουν διαφορετικό βάθος φωλιάσματος ή έχουμε μεταβλητή τότε καλούμε την gnvlcode() με όρισμα το όνομα της και γράφουμε στο αρχείο την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση όση η απόσταση που υπολογίσαμε πριν λίγο.

- Αν έχουμε παράμετρο που έχει περαστεί με αναφορά και η καλούσα συνάρτηση και αυτή έχουν διαφορετικό βάθος φωλιάσματος τότε καλούμε την gnvlcode() με όρισμα το όνομα της γράφουμε στο αρχείο την εντολή με την οποία φορτώνουμε στον καταχωρητή 0 το περιεχόμενο της θέσης μνήμης στην οποία βρίσκεται ο καταχωρητής 0 και την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση όση η απόσταση που υπολογίσαμε πριν λίγο.

Αν έχουμε προσωρινή μεταβλητή καλούμε την συνάρτηση SearchEntity\_1() για να την βρούμε στον πίνακα ,γράφουμε στο αρχείο την εντολή με την οποία προσθέτουμε στον καταχωρητή 0 το περιεχόμενο του sp μείον το offset της μεταβλητής και την εντολή γράφουμε το περιεχόμενο του καταχωρητη 0 σε αυτή τη θέση μνήμης που απέχει από τον fp απόσταση οχτώ θέσεων.

* Αν έχουμε κλήση συνάρτησης: Αν βρισκόμαστε στο αρχικό πρόγραμμα τότε γράφουμε στο αρχείο την εντολή η οποία μετακινεί τον stack pointer κατά το FL της main και αποθηκεύει το αποτέλεσμα στον frame pointer και κάνουμε το level=0 αλλιώς καλούμε την συνάρτηση SearchEntity() με ορίσματα το όνομα της συνάρτησης στην οποία βρισκόμαστε και τύπο ‘συνάρτηση΄ και κρατάμε σε μια μεταβλητή το FL και σε άλλη μια το level της τα οποία μας επιστρέφει και κάνουμε το level ίσο με το επίπεδό της, καλούμε την συνάρτηση SearchEntity για να βρούμε την συνάρτηση στον πίνακα και μετά ελέγχουμε αν είναι στο ίδιο επίπεδο με την καλούσα. Αν ναι τότε γράφουμε στο αρχείο την εντολή με την οποία φορτώνουμε στον καταχωρητή 0 το περιεχόμενο της μνήμης που απέχει από τον stack pointer απόσταση τέσσερα και μετά την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητή 0 στην θέση μνήμης που απέχει από τον frame pointer απόσταση τέσσερα αλλιώς γράφουμε στο αρχείο την εντολή την οποία γράφουμε το περιεχόμενο του καταχωρητή 0 στην θέση μνήμης που απέχει από τον frame pointer απόσταση τέσσερα, μετά αυτή με την οποία προσθέτουμε στον καταχωρητή sp το περιεχόμενο του συν το fl της συνάρτησης, μετά αυτή με την οποία κάνουμε άλμα στο label της και μετά ξανά αυτή την οποία προσθέτουμε στον καταχωρητή sp το περιεχόμενο του μείον το fl της συνάρτησης για να τον επαναφέρουμε.
* Αν έχουμε ξεκίνημα συνάρτησης ή διαδικασίας τότε γράφουμε στο αρχείο την εντολή με την οποία γράφουμε το περιεχόμενο του καταχωρητή ra στην θέση μνήμης στην οποία βρίσκεται ο stack pointer.
* Αν έχουμε το ξεκίνημα του κύριου προγράμματος τότε γράφουμε στο αρχείο την εντολή με την οποία προσθέτουμε στον καταχωρητή sp το περιεχόμενο του συν το fl της main και σημειώνουμε στον $s0 το εγγράφημα δραστηριοποίησης της main ώστε να έχουμε εύκολη πρόσβαση στις global μεταβλητές.
* Τέλος αν έχουμε τελείωμα συνάρτησης ή διαδικασίας τότε γράφουμε στο αρχείο την εντολή με την οποία φορτώνουμε στον καταχωρητή ra το περιεχόμενο της μνήμης στην οποία βρίσκεται ο stack pointer και μετά αυτή με την οποία κάνουμε άλμα στον ra.

**4.** Στην condition() στη γραμμή 1049 αντί να γράψουμε lektikos\_analutis() γράφουμε lektiοkos\_analutis().Πράγμα που εμείς δεν αντιληφθήκαμε διότι δεν είχαμε παράδειγμα με or.