
★ ΑΠΟΛΥΤΗ ★
STARLET

ΑΠΟΛΥΤΗ STARLET

Συγγραφείς :
ΣΩΤΗΡΙΟΣ ΜΠΑΛΑΤΣΙΑΣ
ΣΩΤΗΡΙΑ ΚΑΣΤΑΝΑ

Εκδόσεις ----

ΠΕΡΙΕΧΟΜΕΝΑ

☆ STARLET ☆

Η Starlet είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου η οποία έχει αναπτυχθεί για επιστημονικούς σκοπούς . Πιο συγκεκριμένα ο λόγος δημιουργίας της είναι ο προγραμματισμός του ρομποτικού μη επανδρωμένου σκάφους StarLetGo-07 , του οποίου σκοπός είναι η αποτύπωση των πρώτων πανοραμικών φωτογραφιών του BPM 37093* .

Οι προγραμματιστικές της ικανότητες είναι πολλαπλές και τα προγραμματιστικά εργαλεία που παρέχει είναι τόσα ώστε να απλώνεται ένα εύρος δυνατοτήτων στον προγραμματιστή , καθώς περιέχει δύο προτώτυπες πολύ ισχυρές προγραμματιστικές εντολές οι οποίες κάνουν την υπέρβαση στο κόσμο του προγραμματισμού. Η Starlet επίσης υποστηρίζει συναρτήσεις, μετάδοση παραμέτρων με αναφορά, τιμή και αντιγραφή, αναδρομικές κλήσεις και άλλες ενδιαφέρουσες δομές . Επίσης επιτρέπει φώλιασμα στη δήλωση συναρτήσεων κάτι που λίγες γλώσσες υποστηρίζουν .

Στην συνέχεια αναφέρονται αναλυτικά η περιγραφή της γλώσσας και ο μεταγλωττιστής , ο οποίος και παρουσιάζει ιδιαίτερο και αξιοσημείωτο ενδιαφέρον.

---

* Ο **BPM 37093** είναι ένας «διαμαντένιος»** λευκός νάνος, με δομή που προκύπτει από τις αναπάλσεις του.

**Η αρχική θερμότητα που κληρονόμησαν οι λευκοί νάνοι προέρχεται όχι μόνο από εκείνη που παρέμεινε από την παραγωγή ενέργειας με θερμοπυρηνικές αντιδράσεις στον προγεννήτορά τους αστέρα, αλλά κυρίως από τη βαρυτική δυναμική ενέργεια που μετατράπηκε σε θερμότητα κατά τη βαρυτική κατάρρευση των κεντρικών περιοχών του παλαιού αστέρα. Τη θερμότητα αυτή ξοδεύουν οι λευκοί νάνοι ακτινοβολώντας φως και λοιπή ηλεκτρομαγνητική ακτινοβολία στο διάστημα σύμφωνα με τους γνωστούς νόμους της ακτινοβολίας μέλανος σώματος, του Στεφάν, κλπ., χωρίς να μπορούν να την αναπληρώσουν στο παραμικρό αφού είναι αδρανή, νεκρά σώματα. Επειδή όμως η έκταση της ακτινοβολούσας επιφάνειάς τους είναι ελάχιστη σε σχέση με τη μάζα τους, παραμένουν υπέρθερμοι επί πολλά δισεκατομμύρια χρόνια. Υπάρχουν ενδείξεις ότι το εσωτερικό τους κρυσταλλώνεται αργά καθώς ψύχονται και τελικώς καταλήγει σε κάτι που μοιάζει με τον κρύσταλλο ενός διαμαντιού.

1. ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΓΛΩΣΣΑΣ STARLET

Αρχικά παρουσιάζονται οι **λεκτικές μονάδες** της Starlet :

- Μικρά και κεφαλαία γράμματα του λατινικού αλφαβήτου(“A”…”Z” και “a”…”z”)
- Αριθμητικά ψηφία(“0”,…,”9”)
- Σύμβολα αριθμητικών πράξεων(”+”,”-”,”*”,”/”)
- Τελεστές συσχέτισης(“<”,”>”,”=”,”<=”,”>=”,”<>”)
- Σύμβολο ανάθεσης(“:=”)
- Διαχωριστές(“;”,”,”,”:”)
- Σύμβολα ομαδοποίησης(“(”,”)” στις αριθμητικές παραστάσεις και ”[”,”]” στις λογικές παραστάσεις)
- Σύμβολα διαχωρισμού σχολίων(“/*”,”*/”,”//”)

Κάποιες **δεσμευμένες** λέξεις, οι οποίες δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές :

- program, endprogram
- declare
- if then else endif
- while endwhile, dowhile endwhile
- loop endloop, exit
- forcase endforase, incase endcase, when , default enddefault
- function endfuntion, return, in, out, inandout
- and, or, not
- input, print

Οι **σταθερές** της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία από αριθμητικά ψηφία .

Τα **αναγνωριστικά** της γλώσσας αποτελούνται από γράμματα και ψηφία, αρχίζοντας μόνο από γράμμα, συνθέτοντας μια συμβολοσειρά .

Ο μεταγλωττιστής λαμβάνει υπόψιν του μόνο τα τριάντα πρώτα γράμματα.

Οι **λευκοί χαρακτήρες(**tab, space, return) αν δεν βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά και σταθερές, καθώς αγνοούνται, μπορούν να χρησιμοποιηθούν με οποιοδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή.

Το ίδιο ισχύει και για τα **σχόλια** (“/* …. */” ή “//”) τα οποία απαγορεύεται να ανοίξουν δύο φορές, πριν τα πρώτα κλείσουν και δεν υποστηρίζονται τα εμφωλευμένα σχόλια .

Ο μοναδικός **τύπος δεδομένων** που υποστηρίζει η Starlet είναι οι ακέραιοι αριθμοί οι οποίοι το εύρος τους κυμαίνεται στο διάστημα [-32767,32767] .

Η **δήλωση μεταβλητών** γίνεται με την εντολή declarations και το τέλος της αναγνωρίζεται με το “;” . Τα ονόματα των αναγνωριστικών που ακολουθούν δεν χρειάζονται κάποια άλλη δήλωση καθώς γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και δεν είναι απαραίτητο να βρίσκονται στην ίδια γραμμή . Επιτρέπεται να έχουμε παραπάνω από μια συνεχόμενες χρήσεις της declarations .

Στην συνέχεια παρουσιάζεται η προτεραιότητα των **τελεστών** από την μεγαλύτερη προς την μικρότερη :

- Λογικό “not”
- Πολλαπλασιαστικοί: “*”, “/”
- Μοναδιαίοι προσθετικοί : “+”, “-“
- Δυαδικοί προσθετικοί: “+”, “-“
- Σχεσιακοί: “=”, “<”, “>”, “<>”, “<=”, “>=”
- Λογικό “and”
- Λογικό “or”

Μορφή Προγράμματος:

program id
declarations
subprograms
statements
endprogram

Δομές της Starlet:

Εκχώρηση:
id := expression

Απόφαση if:
if (condition) then
statements
[else
statements]
endif

Επανάληψη while:
while (condition)
statements
endwhile

Επανάληψη dowhile-endowhile:

dowhile

statements

enddowhile (condition)

Επανάληψη loop:

loop

Statements

endloop

Επανάληψη forcase:

forcase

*(when (condition): statements)**

default: statements enddeafult

endforcase

η “forcase” είναι η πρωτη πρωτότυπη πολύ ισχυρή δομή επανάληψης που υποστηρίζει η Starlet. Η forcase ελέγχει τις condition που βρίσκονται μετά τα when . Μόλις μια από αυτές βρεθεί αληθής, τότε εκτελούνται οι statements που ακολουθούν. Μετά ο έλεγχος μεταβαίνει έξω από την forcase. Αν καμία από τις when δεν ισχύει , τότε ο έλεγχος μεταβαίνει στη default και εκτελούνται οι αντίστοιχες statements. Στη συνέχεια ο έλεγχος μεταβαίνει στην αρχή της forcase .

Επανάληψη incase:

incase

*(when (condition): statements)**

endcase

η άλλη πρωτότυπη δομή επανάληψης είναι η “incase” η οποία ελέγχει τις condition που βρίσκονται μετά τα when, εξετάζοντας τες κατά σειρά. Για κάθε μία από αυτές που η αντίστοιχη condition 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 εξεταστούν όλες οι when ο έλεγχος μεταβαίνει έξω από την δομή incase εάν καμία από τις statements δεν έχει εκτελεστεί ή μεταβαίνει στην αρχή της incase, εάν έστω και μια από τις statements έχει εκτελεστεί .

Επιστροφή τιμής:

return expression

Έξοδος:

print expression

Είσοδος:

Input id

Μορφή Υποπρογράμματος:

function id (formal_pars)

declarations

subprograms

statements

endfunction

όπου «formal_pars» είναι η λίστα των τυπικών παραμέτρων.

Οι συναρτήσεις μπορούν να φωλιάσουν η μια μέσα στην άλλη.

Η επιστροφή συνάρτησης γίνεται με το return.

Η Starlet υποστηρίζει τρείς τρόπους **μετάδοσης παραμέτρων**:

- με σταθερή τιμή, όπου δηλώνεται με την λεκτική μονάδα «**in**». Αλλαγές στην τιμή της δεν επιστρέφονται στο πρόγραμμα που κάλεσε την συνάρτηση.
- με αναφορά, όπου δηλώνεται με την λεκτική μονάδα «**inout**». Κάθε αλλαγή στη τιμή της μεταφέρεται αμέσως στο πρόγραμμα που κάλεσε την συνάρτηση.
- με αντιγραφή, όπου δηλώνεται με την λεκτική μονάδα «**inandout**» . Κάθε αλλαγή στη τιμή της μεταφέρεται στο πρόγραμμα που κάλεσε την συνάρτηση, όταν ολοκληρώνεται η εκτέλεση της συνάρτησης .

Τέλος, τα αρχεία της Starlet έχουν κατάληξη .**stl**

Η **Γραμματική της Starlet** είναι η εξής:

<program>	::= program id <block> endprogram
<block>	::= <declarations> <subprograms> <statements>
<declarations>	::= (declare <varlist>;)*
<varlist>	::= ε | id (, id)*
<subprograms>	::= (<subprogram>)*
<subprogram>	::= function id <funcbody> endfunction
<funcbody>	::= <formalpars> <block>
<formalpars>	::= (<formalparlist>)
<formalparlist>	::= <formalparitem> (, <formalparitem>)* | ε
<formalparitem>	::= in id | inout id | inandout id

```
<statements>          ::= <statement> ( ; <statement> ) *
<statement>           ::= ε  | <assignment=stat>  |  <if-stat>  |
                            <while-stat> | <do-while-stat>|
                          <forcase-stat> |  <incase-stat>  |
                               <loop-stat>   |  <exit-stat> |
                      <return-stat> | <input-stat>|<print-stat>
<assignment-stat>     ::= id := <expression>
<if-stat>             ::= if (<condition>) then <statements> <elsepart> endif
<elsepart>            ::= ε | else <statements>
<while-stat>          ::= while (<condition>) <statements> endwhile
<do-while-stat>       ::= dowhile <statements> enddowhile (<condition>)
<loop=stat>           ::= loop <statements> endloop
<exit-stat>           ::= exit
<for-case>            ::= forcase
                            ( when (<condition>) : <statements> ) *
                            default : <statements> enddefault
                          endforcase
<incase-stat>         ::= incase
                            ( when (<condition>) : <statements> ) *
                          endincase
<return-stat>         ::= return <expression>
<print-stat>          ::= print <expression>
<input-stat>          ::= input id
<actualpars>          ::= ( <actualparlist> )
<actualparlist>       ::= <actualparitem> ( , <actualparitem> ) * | ε
<actualparitem>       ::= in <expression> | inout id | inandout id
<condition>           ::= <boolterm> ( or <boolterm> ) *
<boolterm>            ::= <boolfactor> ( and <boolfactor> ) *
```

<boolfactor>	::= not [<condition>] | [<condition>] |
	<expression> <relational-oper> <expression>
<expression>	::= <optional-sign> <term> (<add-oper> <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>

Παρακάτω παρατίθεται ένα απλό παράδειγμα προγράμματος γραμμένο σε Starlet.

```
program examle
   declare c,a,b,t;
   a:=1;
   while (a+b<1 and b<5)
      if (t=1)then
         c:=2
      else
         if (t=2) then
            c:=4
         else
            c:=0
         endif
      endif;
      while (a<1)
          if (a=2) then
              while(b=1)
                    c:=2;
              endwhile
         endif
    endwhile
  endwhile
endprogram
```

2. ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΜΕΤΑΓΛΩΤΤΙΣΤΗ ΤΗΣ STARLET

Ο Μεταγλωττιστής μας (starletc.py) ο οποίος είναι υλοποιημένος σε γλώσσα Python παίρνει ως είσοδο ένα αρχικό πρόγραμμα γραμμένο σε γλώσσα Starlet (.stl) και παράγει σαν έξοδο ένα ισοδύναμο τελικό πρόγραμμα γραμμένο σε assembly (.asm) για τον επεξεργαστή MIPS.

Οι **φάσεις της Μεταγλώττισης** είναι οι εξής:

1. λεκτική ανάλυση
2. συντακτική ανάλυση
3. σημασιολογική ανάλυση
4. παραγωγή ενδιάμεσου κώδικα
5. παραγωγή τελικού κώδικα

Αρχικά έχουμε τον λεκτικό αναλυτή ο οποίος διαβάζει χαρακτήρες και επιστρέφει λεκτικές μονάδες του προγράμματος. Αυτός καλείται από μια ομάδα συναρτήσεων οι οποίες είναι υπεύθυνες για την συντακτική ανάλυση του προγράμματος . Επίσης έχουμε μια ακόμη ομάδα συναρτήσεων οι οποίες με την χρήση ενός Πίνακα Συμβόλων είναι υπεύθυνες για την σημασιολογική ανάλυση . Μέχρι και το σημείο αυτό εμφανίζουμε διαγνωστικά μηνύματα , μηνύματα λάθους ή μηνύματα προειδοποιήσεων (τα οποία εφόσον αυτά υπάρχουν θα ξαναεμφανιστούν κατά την εκτέλεση του προγράμματος στον MIPS). Συγχρόνως δημιουργούμε δύο αρχεία το ένα με τον ενδιάμεσο κώδικα (.int) και το άλλο με τον τελικό κώδικα (.asm) .

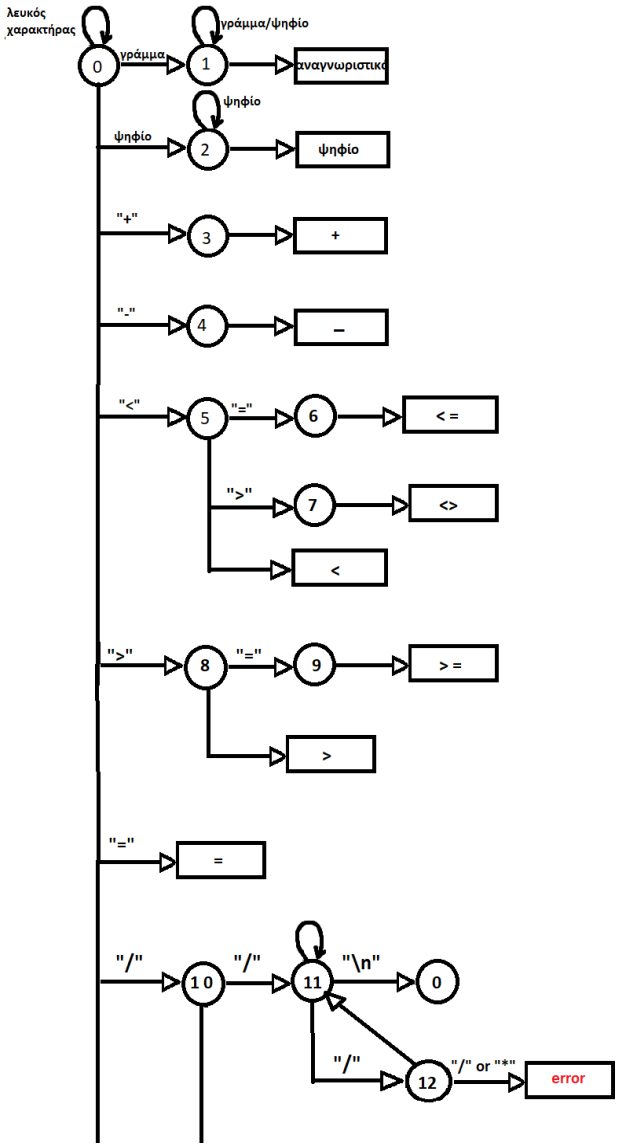
2.1. ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

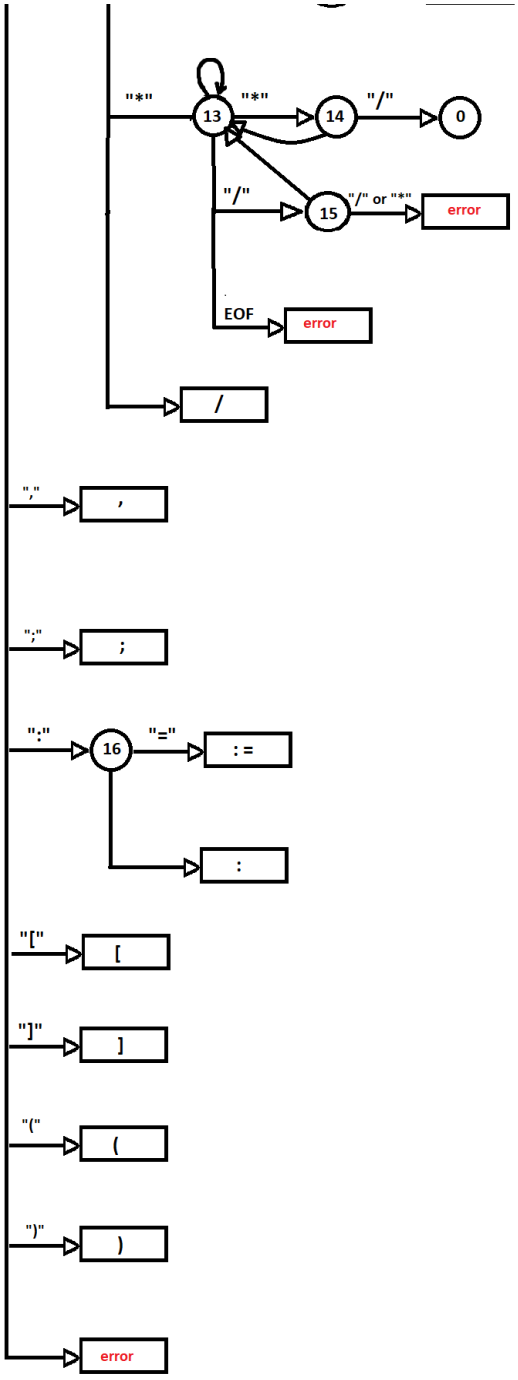
Ο λεκτικός αναλυτής αποτελείται από μια συνάρτηση (lex()) η οποία καλείται από τον συντακτικό αναλυτή. Ο lex() διαβάζει γράμμα-γράμμα το πρόγραμμα Starlet και όταν βρει την επόμενη λεκτική μονάδα ενημερώνει τις global μεταβλητές “tokenID” και “token” με ένα ID που χαρακτηρίζει την λεκτική μονάδα και την λεκτική μονάδα αντίστοιχα .

ΑΥΤΟΜΑΤΟ ΚΑΤΑΣΤΑΣΕΩΝ

Ο lex() εσωτερικά λειτουργεί σαν ένα αυτόματο καταστάσεων το οποίο ξεκινά από μια αρχική κατάσταση και με την είσοδο κάθε χαρακτήρα αλλάζει κατάσταση έως ότου συναντήσει μια τελική κατάσταση, δηλαδή ολοκληρωθεί μια λεκτική μονάδα. Η λεκτική μονάδα που αναγνωρίζεται μπορεί να είναι μια από τις δεσμευμένες λέξεις της Starlet, κάποιο από τα σύμβολα αυτής, αναγνωριστικά ή σταθερές. Τέλος ο lex() τερματίζει το πρόγραμμα σε περίπτωση λάθους με το αντίστοιχο μήνυμα (όπως πχ μη επιτρεπτός χαρακτήρας, κλείσιμο σχολίων χωρίς να έχουν ανοίξει προηγουμένως, εμφωλευμένα σχόλια κλπ).

Η αναπαράσταση του Αυτόματου Καταστάσεων για την γλώσσα Starlet είναι το εξής:


λευκός χαρακτήρας
γράμμα/ψηφίο
γράμμα
αναγνωριστικό
ψηφίο
ψηφίο
ψηφίο
"+"
+
"-"
–
"<"
"="
< =
">"
<>
<
">"
"="
> =
>
"="
=
"/"
"/"
"\n"
"/"
"/" or "*"
error


"*"
13
"*"
14
"/"
0
"/"
15
"/" or "*"
error
EOF
error
/
","
,
";"
;
":"
16
"="
: =
:
"["
[
"]"
]
"("
(
")"
)
error

Η υλοποίηση του lex() πραγματοποιείται ως εξής :

Αρχικά η κατάσταση “0” ως μια while(λευκος χαρακτήρας) η οποία προσπερνάει όλους τους λευκούς χαρακτήρες που υπάρχουν στην αρχή

Στην συνέχεια για κάθε άλλο χαρακτήρα μπαίνουμε σε μια σειρά εντολών απόφασης όπου κάθε μια εντολή από αυτές αναλαμβάνει να συμπληρώσει την λεκτική μονάδα που ξεκίνησε και να ενημερώσει σύμφωνα με αυτήν τις global μεταβλητές “tokenID” και “token” . Πιο συγκεκριμένα :

- Αν ο χαρακτήρας που διαβάστηκε είναι γράμμα (κατάσταση “1”)τότε θα μπει σε μια εσωτερική while() στην οποία θα διαβάσει όλα τα επόμενα γράμματα ή ψηφία έως ότου ολοκληρωθεί το αναγνωριστικό με το οποίο και θα ενημερώσει τις global μεταβλητές . Σημειώνεται ότι εδώ γίνεται ο έλεγχος για το αν το αναγνωριστικό αποτελεί μια από τις δεσμευμένες λέξεις του προγράμματος και αν δεν είναι φροντίζει να κρατήσει μέχρι και τους 30 πρώτους χαρακτήρες και τους υπόλοιπους απλά τους καταναλώνει .
- Αν ο χαρακτήρας που διαβάστηκε είναι ψηφίο (κατάσταση “2”) τότε παρόμοια με πριν μπαίνει σε μια εσωτερική while() η οποία θα διαβάσει ολόκληρο τον φυσικό αριθμό . Τέλος ελέγχει αν είναι μέσα στα όρια [-32767, 32767] και αν τα ξεπερνάει τερματίζεται η μεταγλώττιση με μήνυμα λάθους διαφορετικά ενημερώνονται οι global μεταβλητές .
- Αν ξεκινήσουν σχόλια τότε αυτά καταναλώνονται με μια while() η οποία και σταματάει όταν τα σχόλια τερματιστούν. Δηλαδή αν τα σχόλια ξεκινήσουν με “//” τότε η while() θα τερματίσει με την αλλαγή γραμμής ενώ αν ξεκινήσουν με “*/” η while() θα τερματίσει μόνο όταν διαβαστεί “/*” . Εδώ είναι το σημείο στο οποίο ελέγχουμε για εμφωλευμένα σχόλια ή σχόλια που ανοίγουν και δεν κλείνουν ποτέ. Σ΄αυτές τις περιπτώσεις τερματίζουμε τον μεταγλωττιστή με μήνυμα λάθους . Διαφορετικά ξανακαλείται ο lex() για να ξεκινήσουμε από την κατάσταση “0” .
- Οι υπόλοιπες εντολές απόφασης αναλαμβάνουν κάθε μια και από ένα σύμβολο της γλώσσας Starlet ενημερώνοντας τις global μεταβλητές .

Η ανάγνωση των χαρακτήρων από το αρχείο γίνεται με μια βοηθητική συνάρτηση my_read() η οποία όταν καλείται παίρνει ως παράμετρο το 1 και

το -1 . Όταν θέλουμε να διαβάσουμε τον επόμενο χαρακτήρα την καλούμε ως my_read(1) . Όμως στις περιπτώσεις στις οποίες για να αναγνωρίσουμε μια λεκτική μονάδα χρειάζεται να κοιτάξουμε έναν χαρακτήρα πιο μπροστά για παράδειγμα αν διαβάσουμε το “>” τότε για να ξεχωρίσουμε αν αυτό είναι το “>=” ή το σκέτο “>” πρέπει να διαβάσουμε τον επόμενο χαρακτήρα. Αν ο επόμενος χαρακτήρας είναι το “=” τότε απλά επιστρέφουμε το “>=” αλλιώς όμως πρέπει να “γυρίσουμε πίσω” την my_read() ώστε την επόμενη φορά να μην χάσουμε χαρακτήρα . Αυτή η λειτουργία (“γυρίζουμε πίσω” την my_read) επιτυγχάνεται καλώντας την ως my_read(-1) .

2.2. ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

Κατά την Συντακτική Ανάλυση ελέγχουμε εάν η δομή του προγράμματος ανήκει στην γλώσσα Starlet . Αυτό επιτυγχάνεται χρησιμοποιώντας αναδρομική κατάβαση και βασιζόμενοι σε γραμματική LL(1).

Πιο αναλυτικά για κάθε έναν από τους κανόνες της γραμματικής Starlet υλοποιούμε και την αντίστοιχη συνάρτηση . Κάθε συνάρτηση όταν συναντάει μη τερματικό σύμβολο καλεί και την αντίστοιχη συνάρτηση και όταν συναντήσει τερματικό σύμβολο ελέγχει για το αν αυτό ταιριάζει με αυτό της γραμματικής και συνεχίζει, διαφορετικά τερματίζεται η μεταγλώττιση με μήνυμα λάθους. Τέλος η συντακτική ανάλυση τερματίζει επιτυχώς όταν η τελευταία λέξη που αναγνωριστεί είναι το “endprogram” .

Για παράδειγμα για τον πρώτο κανόνα :

«<program>::= program id <block> endprogram»

έχουμε την αντίστοιχη συνάρτηση

```
function program(){
        lex() //αρχικοποίηση του token
        if (token == “programtk”){
                lex()
```

```
            if (token == “IDtk”) {
                lex()
                block()
                if (token == “endprogramtk”) {
                    // επιτυχής μεταγλώττιση του προγράμματος
                } else {
                    System.error(“expected endprogram”)
                }
            }else {
                System.error(“expected program name”)
            }
        }else {
            System.error(“expected program”)
        }
}
```

Παρόμοια για τον δεύτερο κανόνα:
«<block>::= <declarations> <subprograms> <statements>»

έχουμε την εξής συνάρτηση:

```
function block() {
    declarations()
    subprograms()
    statements()
}
```

Αντίστοιχα δουλεύουμε με κάθε έναν από τους υπόλοιπους κανόνες .

2.3. ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΛΥΣΗ

Στη Σημασιολογική Ανάλυση σκοπός είναι να ελέγξουμε και να αποτρέψουμε την χρήση κοινών ονομάτων σε συναρτήσεις και μεταβλητές για δεύτερη φορά στο ίδιο επίπεδο, κάθε μεταβλητή ή συνάρτηση που χρησιμοποιείται να έχει δηλωθεί και κατά τη κλήση συναρτήσεων τον έλεγχο για το αν οι παράμετροι είναι συμβατοί με τα ορίσματα. Τα παραπάνω ελέγχονται με την βοήθεια του Πίνακα Συμβόλων , ο οποίος είναι καρπός δημιουργίας της Συντακτικής Ανάλυσης και θα αναλυθεί στην συνέχεια. Επίσης στην Σημασιολογική Ανάλυση ελέγχουμε για τον αν κάθε συνάρτηση έχει ένα return και αποτρέπουμε την χρήση του έξω από συνάρτηση . Τέλος η εντολή exit πρέπει να χρησιμοποιείται μόνο μέσα σε βρόχους loop-endloop.

1. Ο έλεγχος και η αποτροπή του να χρησιμοποιήσουμε ίδιο όνομα μεταβλητής ή συνάρτησης στο ίδιο επίπεδο επιτυγχάνεται με την συνάρτηση search_if_repeat() της οποίας η λειτουργία θα αναλυθεί αργότερα
2. Ο έλεγχος για το αν οι μεταβλητές/συναρτήσεις που χρησιμοποιούνται είναι ήδη δηλωμένες επιτυγχάνεται με την συνάρτηση search_for() της οποίας η λειτουργία θα αναλυθεί αργότερα
3. Ο έλεγχος για το αν οι παράμετροι με τις οποίες καλείται μια συνάρτηση είναι ακριβώς αυτές που έχουν δηλωθεί και με την σωστή σειρά πραγματοποιείται με την συνάρτηση CheckForPar() της οποίας η λειτουργία θα αναλυθεί αργότερα
4. Ο έλεγχος για το αν μια συνάρτηση έχει τουλάχιστον ένα return γίνεται με την εξής λειτουργία:
   Έχουμε μια global λίστα με όνομα returnList .
   - Στην αρχή της δήλωσης μιας συνάρτησης (στην συνάρτηση subprogram() (line: 929)) προσθέτουμε στην returnList μια κενή λίστα και στην συνέχεια αναλύουμε τα περιεχόμενα της συνάρτησης.
   - Στα περιεχόμενα αυτά για κάθε return που θα βρίσουμε θα μεταβαίνουμε στην return_stat() (line: 1364) η οποία θα ενημερώσει την λίστα την οποία πρόσθεσε στην returnList η συνάρτηση subprogram()
   - Και στο τέλος της συνάρτησης subprogram() ελέγχουμε για το αν η λίστα την οποία μόλις προσθέσαμε στην returnList είναι άδεια . Αν είναι άδεια σημαίνει ότι η λίστα δεν έχει ενημερωθεί από

κανένα return και συνεπώς έχει τελειώσει μια συνάρτηση χωρίς την παρουσία κάποιου return, για αυτό τερματίζουμε την μεταγλώττιση με μήνυμα λάθους.

5. Ο έλεγχος για το αν υπάρχει κάποιο return έξω από συνάρτηση γίνεται στην return_stat() (line: 1364) στην οποία ελέγχουμε την προαναφερθείσα returnList για το αν είναι άδεια . Αν είναι σημαίνει ότι δεν έχουμε βρει μια συνάρτηση ώστε να κληθεί η subprogram() (η οποία προσθέτει μια λίστα την οποία θα "ενημερώναμε" εδώ) και επομένως τερματίζουμε την μεταγλώττιση με μήνυμα λάθους .
6. Ο έλεγχος για το αν η εντολή exit υπάρχει μόνο στην loop-endloop πραγματοποιείται ως εξής:
    - Στην αρχή της συνάρτησης που αναλύει την λειτουργία της loop-endloop (loop-stat() (line: 1209)) προσθέτουμε στην global λίστα exitList μια κενή λίστα.
    - Στα περιεχόμενα της loop αν βρεθεί exit καλείται η exit_stat() (line: 1230) η οποία θα ενημερώσει την λίστα την οποία μόλις πρόσθεσε η loop_stat() στην exit_list() αφού ελέγξει το μήκος της exitList και δεν την βρει κενή.
    - Αν όμως στον έλεγχο αυτό βρει κενή την exitList σημαίνει πως ο έλεγχος δεν έχει βρει loop_ stat()(η οποία θα πρόσθετε μια λίστα σε αυτήν) άρα η exit αυτή βρίσκεται εκτός loop-endloop και τερματίζουμε την μεταγλώττιση με μήνυμα λάθους.

Πίνακας Συμβόλων

Ο Πίνακας Συμβόλων αποτελείται από τα Scopes, τα Entities και τα Arguments τα οποία περιέχουν:

Scopes:

- Λίστα με entities
- Βάθος Φωλιάσματος
- Βοηθητική μεταβλητή με όνομα "totalOffset" αρχικοποιημένη σε 12 αυξάνεται κατά 4 με κάθε νέα μεταβλητή που προστίθεται στο Scope(συνολική απόσταση από την κορυφή της στοίβας)

Entity:

- Όνομα

- Τύπο
  - Var:
    - offset : απόσταση από την κορυφή της στοίβας
  - tempVar:
    - offset : απόσταση από την κορυφή της στοίβας
  - function:
    - startQuad: το label της πρώτης τετράδας του κώδικα της συνάρτησης
    - framelength : το μήκος εγγραφήματος δραστηριότητας
    - arguments : λίστα με παραμέτρους
  - par:
    - offset: απόσταση από την κορυφή της στοίβας
    - parMode : ο τρόπος περάσματος της μεταβλητής

Argument:

- Τύπο περάσματος του ορίσματος

Τα παραπάνω είναι δομημένα ως εξής:

Αρχικά έχουμε μια global λίστα με όνομα Scope_list η οποία αποτελείται από Scopes . Κάθε ένα από τα Scopes είναι ένα λεξικό με την μορφή :

{"Entity": [], "nestingLevel": level, "totalOffset": 12 (αρχικοποιημένη τιμή)}

Κάθε entity ανάλογα τον τύπο του είναι της μορφής :

- τύπου Var:
  {"name": name, "type": "Var", "offset": offset}
- τύπου tempVar:
  {"name": name, "type": "tempVar", "offset": offset}
- τύπου function:
  {"name": name, "type": "function", "startQuad": None, "framelength": None, "Arguments": []}
- τύπου par:
  {"name": name, "type": "par", "offset": offset, "parMode": parMode}

Όσον αφορά τα Arguments δεν έχουν κάποια δικιά τους δομή, αλλά είναι Strings μέσα στην αντίστοιχη λίστα των function entities.

Επιπλέον για την δημιουργία του Πίνακα Συμβόλων υπάρχουν οι εξής συναρτήσεις:

- addScope(): προσθέτει ένα νέο λεξικό τύπου Scope στην λίστα Scope_list
- addEntityNewVar(name): με παράμετρο το όνομα της μεταβλητής, δημιουργεί ένα λεξικό τύπου entity τύπου “Var“ , στο “offset” βάζει το “totalOffset” του τρέχον Scope και το αυξάνει κατά 4 . Τέλος προσθέτει αυτό το entity στο τελευταίο(αυτό που έχει προστεθεί πιο πρόσφατα) Scope της Scope_list
- addEntityNewTempVar(name): με παράμετρο το όνομα της προσωρινής μεταβλητής, δημιουργεί ένα λεξικό τύπου entity τύπου “tempVar“ , στο “offset” βάζει το “totalOffset” του τρέχον Scope και το αυξάνει κατά 4 . Τέλος προσθέτει αυτό το entity στο τελευταίο(αυτό που έχει προστεθεί πιο πρόσφατα) Scope της Scope_list
- addEntityNewFunction(name): με παράμετρο το όνομα της συνάρτησης, δημιουργεί ένα λεξικό τύπου entity τύπου “function“ , αρχικοποιεί τα “startQuad”,“framelength” σε None,to “Arguments” ως μια κενή λίστα(στην οποία θα προσθέσουμε μετά τον τύπο των παραμέτρων της) και το προσθέτει στο τελευταίο(αυτό που έχει προστεθεί πιο πρόσφατα) Scope της Scope_list
- addEntityNewPar(name, parMode): με παράμετρο το όνομα της παραμέτρου και τον τύπο αυτής, δημιουργεί ένα λεξικό τύπου entity τύπου “par“, στο “offset” βάζει το “totalOffset” του τρέχον Scope και το αυξάνει κατά 8 αν η παράμετρος είναι περασμένη με αντιγραφή , διαφορετικά το αυξάνει κατά 4. Τέλος προσθέτει αυτό το entity στο τελευταίο(αυτό που έχει προστεθεί πιο πρόσφατα) Scope της Scope_list Στο σημείο αυτό να σημειωθεί πως οι παράμετροι οι οποίες είναι περασμένες με αντιγραφή πιάνουν τον διπλάσιο χώρο στην στοίβα, δηλαδή δύο θέσεις (2 x 4 bytes) καθώς χρησιμοποιούμε την πρώτη θέση για να αποθηκεύσουμε την τιμή της μεταβλητής την οποία περνάμε και την δεύτερη για να αποθηκεύσουμε την διεύθυνση αυτής στην στοίβα . Οπότε κάθε φορά που θέλουμε να χρησιμοποιήσουμε (διαβάσουμε/γράψουμε) την παράμετρο αυτή χρησιμοποιούμε την πρώτη θέση (δηλαδή την τιμή της ωστέ να μην επηρεάζεται η μεταβλητή την οποία περάσαμε ως όρισμα) και στο τέλος της συνάρτησης γράφουμε την τιμή αυτή (την πρώτη θέση) στην διεύθυνση που αναγράφεται στη δεύτερη θέση. Με αυτό τον τρόπο μετά το τέλος

μιας συνάρτησης, οι παράμετροι οι οποίες έχουν περαστεί με αντιγραφή ενημερώνουν τις κατάλληλες μεταβλητές.

- addArgument(parMode): με παράμετρο τον τύπο περάσματος της μεταβλητής προσθέτει ένα String στα Arguments του Entity της συνάρτησης (το οποίο βρίσκεται σε ένα προηγούμενο Scope) με τον τύπο περάσματος (parMode).

Οι συναρτήσεις αυτές , καλούνται μέσα στις συναρτήσεις που έχουν δημιουργηθεί για τον Συντακτικό Αναλυτή .

- Αρχικά η δημιουργία των Scopes γίνεται πρώτον στην συνάρτηση program() (εδώ δημιουργείται το Scope της main) και δεύτερον όλα τα υπόλοιπα Scopes δημιουργούνται στην συνάρτηση subprogram() .
- Η δημιουργία των entities γίνεται :
    - entities τύπου “Var” δημιουργούνται όταν δηλώνουμε τις μεταβλητές, δηλαδή μέσα στην varlist() (line: 903)
    - entities τύπου “newTemp” δημιουργούνται την στιγμή που δημιουργείται μια νέα προσωρινή μεταβλητή στην συνάρτηση newTemp() (line: 287) η οποία θα αναλυθεί αργότερα στην παραγωγή ενδιάμεσου κώδικα
    - entities τύπου “function” δημιουργούνται κατά την δήλωση μιας καινούριας συνάρτησης στην συνάρτηση subprogram() (line: 929)
    - entities τύπου “par” δημιουργούνται κατά τον ορισμό παραμέτρων μιας συνάρτησης στην συνάρτηση formalparitem() (line: 1013)
- Τέλος η δημιουργία των Arguments γίνεται κατά τον ορισμό παραμέτρων μιας συνάρτησης στην συνάρτηση formalparitem() (line: 1013)

Επιπλέον έχουμε και τις εξής συναρτήσεις οι οποίες χρησιμοποιούνται για την Σημασιολογική Ανάλυση του κώδικα :

- search_for (name): με παράμετρο ένα όνομα ψάχνει στην Scope_list ξεκινώντας από το τελευταίο (αυτό που έχει προστεθεί πιο πρόσφατα) και αναζητά ένα entity με το ίδιο όνομα. Αν το βρει επιστρέφει το entity αυτό μαζί με το βάθος φωλιάσματος στο οποίο το βρήκε. Αν δεν το βρει ελέγχουμε αν το όνομα με το οποίο έχουμε κάνει την αναζήτηση

μοιάζει με κάποιο από τα ονόματα που χρησιμοποιούμε μέσα στο πρόγραμμα (τα ονόματα όλων των entities) αυτό γίνεται με την βοήθεια της συνάρτησης devenshtein_distance() την οποία και θα αναλύσουμε αργότερα. Και τερματίζει την μεταγλώττιση με κατάλληλο μήνυμα.

- search_if_repeat(name): με παράμετρο ένα όνομα διατρέχει το τελευταίο Scope(αυτό που έχει προστεθεί πιο πρόσφατα) και ελέγχει αν υπάρχει entity με αυτό το όνομα . Αν αυτό υπάρχει τότε τερματίζει την μεταγλώττιση με κατάλληλο μήνυμα, διαφορετικά επιστρέφει True (αυτή η συνάρτηση χρησιμοποιείται για να αποφύγουμε την δήλωση μιας μεταβλητής με όνομα το οποίο χρησιμοποιείται ήδη μέσα στην συνάρτηση που βρίσκεται) .
- CheckForPar(fname, callPar): με ορίσματα το όνομα της συνάρτησης και μια λίστα με τους τύπους παραμέτρων με τους οποίους καλούμε αυτή την συνάρτηση και ελέγχουμε αν αυτοί ταιριάζουν τόσο στο πλήθος όσο και στον τρόπο περάσματος με τις παραμέτρους με τις οποίες έχει οριστεί η συνάρτηση .

Τέλος έχουμε την συνάρτηση delScope() η οποία αν πρόκειται για Scope μιας συνάρτησης ενημερώνει το πεδίο “framelength” του entity του προηγούμενου Scope, την οποία βρίσκει στο πεδίο του “totalOffset” . Και τέλος διαγράφει το Scope από την global λίστα Scope_list.

Λεξικογραφικός Έλεγχος Ονομάτων

Αρχικά έχουμε την συνάρτηση levenshtein(seq1, seq2) με ορίσματα δύο λέξεις επιστρέφει σε ποσοστό επί τοις εκατό το πόσο αυτές διαφέρουν λεξικογραφικά (100% -> εάν αυτές είναι εντελώς διαφορετικές και 0% -> εάν αυτές ταυτίζονται).

Στη συνέχεια η συνάρτηση devenshtein_distance(id,list) με βάση το όρισμα id και μια λίστα με ονόματα επιστρέφει το όνομα μέσα από την λίστα με το οποίο μοιάζει το iD περισσότερο . Όταν λέμε περισσότερο εννοούμε την λέξη με την όποια έχει την μικρότερη λεξικογραφική διαφορά . Παρ’όλα αυτά αν αυτή η διαφορά του id με όλες τις λέξεις από την λίστα είναι μεγαλύτερη της τάξης του 50% η συνάρτηση επιστρέφει False.

2.4. ΠΑΡΑΓΩΓΗ ΕΝΔΙΑΜΕΣΟΥ ΚΩΔΙΚΑ

Ο Ενδιάμεσος Κώδικας είναι ένα σύνολο τετράδων οι οποίες αποτελούνται από έναν τελεστή και τρία τελούμενα . (πχ +, a, b, c) . Η κάθε τετράδα συνοδεύεται με έναν αριθμό ο οποίος είναι μοναδικός για την καθεμία και σύμφωνα με αυτόν εκτελούνται οι τετράδες – εντολές , πρώτα με τον μικρότερο και μετά αυτή με τον αμέσως μεγαλύτερο εκτός και αν η τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό (πχ jump, _, _, 100) .

Οι τετράδες χωρίζονται στις εξής κατηγορίες:

- Πρώτον έχουμε τις τετράδες της μορφής «op, a, b, c»
  όπου το op μπορεί να είναι ένα εκ των : +, -, *, / και η εκτέλεση της τετράδας αυτής είναι να γίνει η πράξη μεταξύ των a και b (μεταβλητές/αριθμητικές σταθερές)και το αποτέλεσμα να εκχωρηθεί στην μεταβλητή c.

πχ: Η πράξη c = a + b αντιστοιχεί στην τετράδα +, a, b, c

- Δεύτερον έχουμε τετράδες της μορφής «:=, a, _, b»
  της οποίας η εκτέλεση είναι ότι η τιμή του τελούμενου a (μεταβλητή/αριθμητική σταθερά) εκχωρείται στην μεταβλητή b.

πχ: Η εκχώρηση x:= y + 5 αντιστοιχεί στις τετράδες : 100: +, y, 5, T_1

101: :=, T_1, _, x

- Τρίτον τετράδες της μορφής «jump, _, _, a»
  της οποίας η εκτέλεση είναι η μεταπήδηση χωρίς όρους στην τετράδα με αριθμό a και να συνεχιστεί η εκτέλεση από εκεί και κάτω .
- Τέταρτον τετράδες της μορφής «relop, a, b, c»
  όπου relop μπορεί να είναι ένα εκ των: =, <, >, <>, <=, >= και η εκτέλεση της τετράδας αυτής είναι η μεταπήδηση στην τετράδα με αριθμό c, εφόσον ισχύει η σχέση “relop” μεταξύ των a και b.

πχ: Η if(x>3) then { statements } αντιστοιχεί στις τετράδες : 100: >,x,3,102

101: jump, _,_, 103

102: {statements}

103: …

- Πέμπτον έχουμε τετράδες οι οποίες υποδηλώνουν την αρχή και το τέλος μιας ενότητας :
  → «begin_block, name, _, _»
  η οποία δηλώνει την αρχή του προγράμματος ή μιας συνάρτησης με το όνομα name
  → «end_bloc, name, _, _»
  η οποία δηλώνει το τέλος του προγράμματος ή μιας συνάρτησης με το όνομα name
  → «halt, _, _, _»
  η οποία δηλώνει τον τερματισμό του προγράμματος

πχ: το πρόγραμμα program p1{
function f1(){
x = x +1
}
}
θα γίνει: 100: begin_block, p1, _, _
101: begin_block, f1, _, _
102: := x, 1, x
103: end_block, f1, _, _
104: halt, _, _, _
105: end_block, p1, _, _

- Και τέλος έκτον έχουμε τετράδες που αφορούν συναρτήσεις:
  → «par, x, m, _»
  με το x να είναι η παράμετρος συνάρτησης και m ο τρόπος μετάδοσης :
  CV : μετάδοση με τιμή
  REF: μετάδοση με αναφορά
  CVREF: μετάδοση με αντιγραφή
  RET: επιστροφή τιμής συνάρτησης
  → «call, name, _, _»
  κλήση συνάρτησης με όνομα name
  → «RETV, x, _, _»

επιστροφή τιμής συνάρτησης

πχ: η κλήση της συνάρτησης f1(in a, inout b) θα γίνει: 100: par, a, CV, _
101: par, b, REF, _
102: par, T_1, RET, _
103: call, f1, _, _

Για την παραγωγή του Ενδιάμεσου Κώδικα έχουμε τις εξής βοηθητικές συναρτήσεις:

- genquad(a,b,c,d) : δημιουργεί μια λίστα πέντε θέσεων . Στην πρώτη θέση βάζει τον αριθμό της επόμενης τετράδας (καλώντας την nextquad()) και στις υπόλοιπες τέσσερεις το a, b, c, d . Στη συνέχεια προσθέτει αυτή την λίστα σε μια global λίστα με όνομα quad_list ,η οποία κρατάει τις λίστες με όλες τις τετράδες του ενδιάμεσου κώδικα.
- nextquad() : επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί
- newTemp() : δημιουργεί και επιστρέφει μια νέα προσωρινή μεταβλητή (οι προσωρινές μεταβλητές έχουν την μορφή T_1, T_2 κ.ο.κ)
- merge(A,B) : με παραμέτρους δύο λίστες A, B συγχωνεύει τα στοιχεία τους και τα επιστρέφει όλα μαζί σε μια νέα λίστα
- emptylist(): δημιουργεί και επιστρέφει μια κενή λίστα
- makelist(a): δημιουργεί και επιστρέφει μια νέα λίστα με το στοιχείο a
- backpatch(list,label): διατρέχοντας όλες τις τετράδες από την global quad_list , πηγαίνει σε αυτές οι οποίες είναι σημαδεμένες μέσα στην list και τις ενημερώνει με την ετικέτα label

Τις παραπάνω συναρτήσεις τις χρησιμοποιούμε στις συναρτήσεις που έχουμε δημιουργήσει για την Συντακτική Ανάλυση . Για να γίνει κατανοητός ο τρόπος με τον οποίο τις χρησιμοποιούμε παρατίθεται ένα παράδειγμα .

Έστω το παρακάτω κομμάτι κώδικα γραμμένο σε γλώσσα Starlet:

```
incase
        when (a > 0) : c := 2
endincase
```

και ο αντίστοιχος ενδιάμεσος κώδικας είναι ο :

```
100: :=, 0, _, T_1
101: >, a, 0, 103
102: jump, _, _, 105
103: :=, 1, _, T_1
104: :=, c, _, 2
105: =, 1, T_1, 100
```

Αρχικά θα αναλύσουμε τα βήματα με τα οποία προκύπτει ο ενδιάμεσος κώδικας:

- Με την χρήση της newTemp() δημιουργούμε μια τυχαία μεταβλητή (στο παράδειγμα μας η T_1)
- Σημαδεύουμε την επόμενη τετράδα(με την χρήση της nextquad()) που θα δημιουργηθεί την οποία θα χρησιμοποιήσουμε για να πετύχουμε την λειτουργία επανάληψης της εντολής
  (στο παράδειγμα μας θα σημαδέψουμε το quad με label = 100)
- με την χρήση της genquad() δημιουργούμε την τετράδα «:=, 0, _, T_1» η οποία αποθηκεύεται με label 100 (η μεταβλητή T_1 θα γίνει ίση με 1 αν πραγματοποιηθεί μια από τις statements μέσα στις when που ακολουθούν)
- Στην συνέχεια θα εκτελεστεί η condition() η οποία θα δημιουργήσει τις τετράδες που αντιστοιχούν στην συνθήκη «a > 0»
  (101: >, a, 0, _)
  (102: jump, _, _, _)
- Με την χρήση της backpatch() ενημερώνουμε τις τετράδες σύμφωνα με την λίστα True(η οποία περιέχει την 101) που επέστρεψε η condition()

να δείχνουν στην επόμενη τετράδα που θα δημιουργηθεί, την 103 (η οποία προκύπτει με την nextquad())

- με την χρήση της genquad() δημιουργούμε την τετράδα «:=, 1, _, T_1» η οποία αποθηκεύεται με label 103
- Στη συνέχεια εκτελείται η statements() η οποία θα δημιουργήσει τις τετράδες που αντιστοιχούν στο «c := 2»
  (104: :=, c, _, 2)
- Με την χρήση της backpatch() ενημερώνουμε τις τετράδες σύμφωνα με την λίστα False(η οποία περιέχει την 102) που επέστρεψε η condition() να δείχνουν στην επόμενη τετράδα που θα δημιουργηθεί, την 105 (η οποία προκύπτει με την nextquad())
- με την χρήση της genquad() δημιουργούμε την τετράδα «=, 1, T_1, 100» Αν η T_1 ισούται με το 1 σημαίνει πως έχει εκτελεστεί κάποια από τις statements και άρα θα μεταβούμε στην αρχή της incase

2.5. ΠΑΡΑΓΩΓΗ ΤΕΛΙΚΟΥ ΚΩΔΙΚΑ

Η κάθε εντολή του τελικού κώδικα παράγεται από την αντίστοιχη εντολή του ενδιάμεσου κώδικα . Ο τελικός κώδικάς που δημιουργείται είναι σύμφωνος με τον επεξεργαστή MIPS. Αυτή η φάση του μεταγλωττιστή είναι υπεύθυνη για την απεικόνιση των μεταβλητών στην μνήμη (στοίβα) και στον σωστό τρόπο περάσματος παραμέτρων κατά την κλήση συναρτήσεων. Οι εντολές που θα παραχθούν τις αποθηκεύουμε σε μια global λίστα με όνομα asm_list.

Η παραγωγή τελικού κώδικα γίνεται με την βοήθεια της genASM() (line: 332) η οποία καλείται στην delScope() αυτή διαγράψει το Scope. Δηλαδή την στιγμή που έχει παραχθεί ενδιάμεσος κώδικας για μια ολόκληρη συνάρτηση ή της main . Με άλλα λόγια η genASM() ασχολείται με ένα Scope την φορά.

Αρχικά έχουμε τρείς βοηθητικές συναρτήσεις:

- gnlvcode(v): Η συνάρτηση αυτή γράφει στον καταχωρητή $t0 την αναφορά της μη τοπικής μεταβλητής v, η οποία περνιέται ως παράμετρος. Με την βοήθεια της search_for() βρίσκουμε το entity που αντιστοιχεί στην μεταβλητή v από τον πίνακα συμβόλων και το βάθος φωλιάσματος στο οποίο αυτή είναι δηλωμένη.
  Σε μια προσωρινή λίστα προσθέτουμε την «lw $t0, -4($sp)» η οποία μας δίνει την στοίβα του γονέα. Στην συνέχεια μέσα σε μια δομή επανάληψης προσθέτουμε την «lw $t0, -4($t0)» όσες φορές χρειαστεί ώστε να φτάσει στην στοίβα του προγόνου στον οποίο ορίζεται η v .
  Στη συνέχεια προσθέτουμε στην προσωρινή λίστα την «addi $t0, $t0, -offset» η οποία προσθέτει στον $t0 το offset της μεταβλητής v (το offset το βρίσκουμε από το entity το οποίο επέστρεψε η search_for()).
  Τέλος επιστρέφουμε την λίστα αυτή.
- loadvr(v,r): Η συνάρτηση αυτή γράφει στον καταχωρητή r την τιμή της v. Για να βρούμε την τιμή της v παίρνουμε το entity που αντιστοιχεί σε αυτή μέσω της search_for() και διακρίνουμε τις εξής περιπτώσεις:
  - αν το entity είναι αριθμητική σταθερά προσθέτουμε σε μια προσωρινή λίστα την εντολή «li $tr, v»
  - αν το entity είναι καθολική μεταβλητή τότε προσθέτουμε στην προσωρινή λίστα την εντολή «lw $tr, -offset($s0)»
  - αν το entity είναι τοπική μεταβλητή (δηλαδή το βάθος φωλιάσματος στο οποίο βρίσκεται το entity είναι ίδιο με το βάθος φωλιάσματος το οποίο αναλύουμε τώρα) ή τυπική παράμετρος που περνιέται με τιμή και βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή τότε προσθέτουμε σε μια προσωρινή λίστα την εντολή «lw $tr, -offset($sp)»
  - αν το entity είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον τότε προσθέτουμε σε μια προσωρινή λίστα τις εξής εντολές:
    «lw $t0, -offset($sp)»
    «lw $tr, ($t0)»
  - αν το entity είναι τοπική μεταβλητή ή τυπική παράμετρος που περνιέται με τιμή και βάθος φωλιάσματος μικρότερο απο το τρέχον τότε προσθέτουμε σε μια προσωρινή λίστα τις εξής εντολές:
    gnlvcode()

«lw $tr, ($t0)»

- αν το entity είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον τότε προσθέτουμε σε μια προσωρινή λίστα τις εξής εντολές:
  gnlvcode()
  «lw $t0, ($t0)»
  «lw $tr, ($t0)»

Τέλος επιστρέφουμε την λίστα αυτή.

- storerv(r,v): Η συνάρτηση αυτή μεταφέρει δεδομένα από τον καταχωρητή r στην μνήμη (μεταβλητή v)
  - αν το entity είναι καθολική μεταβλητή τότε προσθέτουμε στην προσωρινή λίστα την εντολή «sw $tr, -offset($s0)»
  - αν το entity είναι τοπική μεταβλητή ή τυπική παράμετρος που περνιέται με τιμή και βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή τότε προσθέτουμε σε μια προσωρινή λίστα την εντολή «sw $tr, -offset($sp)»
  - αν το entity είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον τότε προσθέτουμε σε μια προσωρινή λίστα τις εξής εντολές:
    «lw $t0, -offset($sp)»
    «sw $tr, ($t0)»
  - αν το entity είναι τοπική μεταβλητή ή τυπική παράμετρος που περνιέται με τιμή και βάθος φωλιάσματος μικρότερο απο το τρέχον τότε προσθέτουμε σε μια προσωρινή λίστα τις εξής εντολές:
    gnlvcode(v)
    «sw $tr, ($t0)»
  - αν το entity είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον τότε προσθέτουμε σε μια προσωρινή λίστα τις εξής εντολές:
    gnlvcode(v)
    «lw $t0, ($t0)»
    «sw $tr, ($t0)»

Τέλος επιστρέφουμε την λίστα αυτή.

Η **genASM(main)** λειτουργεί ως εξής:

Αρχικά αν η παράμετρος main έχει την τιμή True σημαίνει πως τώρα θα αναλύσουμε το Scope της main και άρα προσθέτουμε στην global λίστα quad_list(η λίστα με τις τετράδες του ενδιάμεσου κώδικα) το label “Lmain” και τις δύο εντολές «add $sp,$sp,main-framelength», «move $s0,$sp» , όπου το main-framelength το παίρνουμε από το πεδίο “totalOffset” του Scope που διατρέχουμε .

Στη συνέχεια διαβάζει από την quad_list κάθε τετράδα. Και σύμφωνα με το πρώτο στοιχείο της τετράδας καλεί και την αντίστοιχη συνάρτηση. Οι παρακάτω συναρτήσεις επιστρέφουν μια λίστα με εντολές τελικού κώδικα την οποία και προσθέτουμε στην global λίστα asm_list .

Στην περίπτωση που το πρώτο μέρος της τετράδας είναι :

- begin_block : δηλώνεται η αρχή μιας συνάρτησης, άρα αποθηκεύουμε στη πρώτη θέση του εγγραφήματος δραστηριοποίησης την πρώτη θέση την οποία έχει τοποθετήσει στον $ra η jal και επιστρέφει μια προσωρινή λίστα με την εντολή «sw $ra, ($sp)»
- end_block: δηλώνεται το τέλος μιας συνάρτησης, παίρνουμε παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της συνάρτησης και την βάζουμε πάλι στον $ra. Μέσω του $ra επιστρέφουμε στην καλούσα
  «lw $ra,($sp)»
  « jr $ra»
- “:=” : προσθέτει στην προσωρινή λίστα τις λίστες loadvr(x,1) ,storerv(1,z). Όπου x η δεύτερη θέση της τετράδας και z η τέταρτη και επιστρέφουμε την λίστα αυτή
- ‘’=’’,’’<>’’,’’<’’,’’>’,’’<=’’,’’>=’ :
  προσθέτει στην προσωρινή λίστα τις λίστες loavr(x,1) , loadvr(y,2) και την εντολή «branch(?) , $t1, $t2, z» . Όπου x η δεύτερη θέση της τετράδας ,y η τρίτη , z η τέταρτη και όπου branch(?) ένα από τα: ‘’beq’’,’’bne’’,’’bgt’’,’’blt’’,’’bge’’,’’ble’’ και επιστρέφουμε την λίστα αυτή
- “+”, “-”,”*”,”/”:
  προσθέτει στην προσωρινή λίστα τις λίστες loavr(x,1) , loadvr(y,2), την εντολή «op $t1,$t1,$t2» και την λίστα store(1,z) .

Όπου x η δεύτερη θέση της τετράδας ,y η τρίτη , z η τέταρτη και όπου op ένα από τα: “add’’,’’sub’’,’’mul’’,’’div’’ και επιστρέφουμε την λίστα αυτή

- “jump” : προσθέτει στην προσωρινή λίστα την εντολή «j label» όπου label η τέταρτη θέση της τετράδας και επιστρέφουμε την λίστα αυτή
- “in” : προσθέτει στην προσωρινή λίστα τις εντολές «li $v0,10» , «syscall» και επιστρέφουμε την λίστα αυτή
- “out” : προσθέτει στην προσωρινή λίστα την εντολή «li $v0,1» , την λίστα loadvr(a,0) και τις εντολές «move $a0, $t0», «syscall» όπου a η τέταρτη θέση της τετράδας και επιστρέφουμε την λίστα αυτή
- “par”: εδώ και μόνο εδώ αυξάνεται ένας μετρητής ο οποίος μετρά τον αριθμό των παραμέτρων που συναντά και τον χρησιμοποιούμε ώστε να βάλουμε σωστά το offset (σε όλες τις υπόλοιπες συναρτήσεις της genASM() αυτός ο μετρητής μηδενίζεται). Εάν αυτός ο μετρητής είναι ίσος με το μηδέν σημαίνει πως είμαστε στην πρώτη παράμετρο και πρέπει να μετακινήσουμε τον $fp σύμφωνα με το framelength της συνάρτησης που θα κληθεί. Όμως σε αυτό το σημείο δεν ξέρουμε την συνάρτηση που θα κληθεί οπότε προσθέτουμε ένα κενό στην λίστα asm_list και σημαδεύουμε την θέση αυτής ώστε να μπορούμε να γυρίσουμε και να την συμπληρώσουμε όταν θα γνωρίζουμε το όνομα της μεταβλητής (στην call)
  Αυξάνουμε τον μετρητή και έχουμε τις εξής περιπτώσεις:
  - **πέρασμα παραμέτρου με τιμή:**
    προσθέτει στην προσωρινή λίστα τη λίστα load(x,0) και την εντολή «sw $t0, -(12+4i)($fp)» όπου x η δεύτερη θέση της τετράδας και i ο μετρητής που προαναφέραμε και επιστρέφουμε την λίστα αυτή.
  - **πέρασμα παραμέτρου με αναφορά:**
    - αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή,
      προσθέτει στην προσωρινή λίστα τις εντολές «add $t0, $sp, -offset», «sw $t0, -(12+4i)($fp)» όπου το x

είναι η δεύτερη θέση της τετράδας, το offset αυτό που αναγράφεται στο entity της παραμέτρου x και i ο μετρητής που προαναφέραμε και επιστρέφουμε την λίστα αυτή.

- αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά, προσθέτει στην προσωρινή λίστα τις εντολές «lw $t0,-offset($sp)» , «sw $t0,-(12+4i)($fp)» όπου το x είναι η δεύτερη θέση της τετράδας, το offset αυτό που αναγράφεται στο entity της παραμέτρου x και i ο μετρητής που προαναφέραμε και επιστρέφουμε την λίστα αυτή.
- αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή, προσθέτει στην προσωρινή λίστα τις εντολές : gnlvcode(x), «sw $t0,-(12+4i)($fp)» και i ο μετρητής που προαναφέραμε και επιστρέφουμε την λίστα αυτή.
- αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά, προσθέτει στην προσωρινή λίστα τις εντολές : gnlvcode(x), « lw $t0,($t0)», «sw $t0,-(12+4i)($fp)» και i ο μετρητής που προαναφέραμε και επιστρέφουμε την λίστα αυτή.

- **πέρασμα παραμέτρου με αντιγραφή:**
  όπως έχουμε αναφέρει και στον πίνακα συμβόλων η παράμετρος η οποία περνάει με αντιγραφή πιάνει δύο θέσεις στην στοίβα (2x4 bytes). Στη πρώτη θέση εκτελούμε ακριβώς τις εντολές που εκτελέσαμε και για το πέρασμα παραμέτρων και με τιμή , έτσι στην πρώτη θέση αποθηκεύτηκε η τιμή της μεταβλητής . Στη συνέχεια για την σωστή λειτουργία του προγράμματος αυξάνουμε τον μετρητή που έχουμε προαναφέρει κατά ένα και εκτελούμε τις εντολές που αφορούν το πέρασμα παραμέτρου με

αναφορά, με την διαφορά ότι τώρα η αναφορά της τιμής θα γραφτεί στη δεύτερη θέση της παραμέτρου, που είναι περασμένη με αντιγραφή, και όχι σε καινούρια παράμετρο. Με αποτέλεσμα η πρώτη θέση της παραμέτρου να περιέχει την τιμή της μεταβλητής και η δεύτερη την αναφορά αυτής .

- **επιστροφή συνάρτησης:**
  γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή προσθέτει στην προσωρινή λίστα τις εντολές «add $t0,$sp,-offset» και « sw $t0,-8($fp)» και επιστρέφουμε την λίστα αυτή.

- "call": αρχικά γεμίζουμε το 2ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με την διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα της, ώστε η κληθείσα να γνωρίζει που να κοιτάξει αν χρειαστεί να προσπελάσει μία μεταβλητή την οποία έχει δικαίωμα να προσπελάσει, αλλά δεν της ανήκει
  - αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα και προσθέτουμε στην προσωρινή λίστα τις εντολές : «lw $t0,-4($sp)», «sw $t0,-4($fp)»
  - αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας και προσθέτουμε στην προσωρινή λίστα την εντολή: «sw $sp,-4($fp)»
  - στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα και προσθέτουμε στην προσωρινή λίστα την εντολή : «add $sp,$sp,framelength»
  - καλούμε τη συνάρτηση και προσθέτουμε στην προσωρινή λίστα την εντολή : «jal f»
    και όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα και προσθέτουμε στην προσωρινή λίστα την εντολή: «add $sp,$sp,-framelength» και επιστρέφουμε την λίστα αυτή.

- “RETV”: προσθέτουμε στην προσωρινή λίστα την λίστα loadvr(x,1) και τις εντολές «lw $t0,-8($sp)», «sw $t1,($t0)» . Σε αυτό το σημείο αποθηκεύεται ο x στη διεύθυνση που είναι αποθηκευμένη στην 3η θέση του εγγραφήματος δραστηριοποίησης, όπου x η τέταρτη θέση της τετράδας και επιστρέφουμε την λίστα αυτή .¨

  Στη συνέχεια σε μια δομή επανάληψης ξεκινώντας από το πρώτο entity του Scope στο οποίο βρισκόμαστε εντοπίζουμε (αν υπάρχουν) τις παραμέτρους οι οποίες έχουν περαστεί με αντιγραφή και για κάθε μια διαβάζουμε την τιμή που είναι γραμμένη στην πρώτη εκ των δύο θέσεων και την γράφουμε στην αναφορά η οποία είναι γραμμένη στην δεύτερη εκ των δύο θέσεων . Η διαδικασία αυτή με την προσθήκη των εντολών που ακολουθούν για κάθε μια από τις παραμέτρους . Οι εντολές αυτές είναι : «lw $t1, -offset($sp)», ($1^{η}$ θέση : $sp + offset)
  «lw $t0, -offset+4($sp)», ($2^{η}$ θέση : $sp + offset +4)
  «sw $t1, ($t0)»
  όπου offset το offset της καθεμιάς παραμέτρου το οποίο έχουμε βρει από το entity το οποίο αντιστοιχεί σε αυτή

  και επιστρέφουμε την λίστα αυτή .
- “halt” : προσθέτουμε στην προσωρινή λίστα τις εντολές : «li $v0,10» , «syscall» με τις οποίες το πρόγραμμα τερματίζει και επιστρέφουμε την λίστα αυτή.

Αφού τελειώσουμε με όλες τις εντολές έχουμε μια λίστα με εντολές τελικού κώδικα , οπότε στην asm_list προσθέτουμε ένα label με το όνομα της τετράδας , την οποία μόλις αναλύσαμε, και στην συνέχεια τις εντολές που υπάρχουν μέσα στην προαναφερθείσα λίστα και ο αλγόριθμος συνεχίζει παίρνοντας την επόμενη τετράδα.

3. ΠΑΡΑΔΕΙΓΜΑΤΑ

~ Ακολουθούν δύο παραδείγματα προγραμμάτων σε Starlet, στις τετράδες που προκύπτουν από αυτό (Ενδιάμεσος Κώδικας) , Τελικός Κώδικας καθώς και ολόκληρος ο πίνακας Συμβόλων .

Παράδειγμα 1

Στο παρακάτω πρόγραμμα έχουμε έναν συνδυασμό σχεδόν όλων των εντολών της Starlet χωρίς συναρτήσεις .

το πρόγραμμα σε γλώσσα Starlet:

```
program bigtest
        declare a,b,c,x,y;
        //user gives 3 values for a, b, c from keyboard
        input a;
        input b;
        input c;
        x := 0;
        y := 0;
        if (a=1) then    //if-case
                while (b<c)  //while-case
                        if (b<5) then
                                dowhile //dowhile-case
                                        b := b + a;
                                        x := x + 1
                                enddowhile(x=10)
                        else
                                forcase  //forcase-case
                                        when (b=5) : b := b * 2
                                        when (b=6) : b := b * 4
                                        when (b=7) : b := b * 6
                                        default: b := b / 2 enddefault
```

```
                endforcase
            endif
        endwhile
    else
        loop  //loop-case
            if(c<5) then
                dowhile  //dowhile-case
                    c := c + a;
                    y := y + 2
                enddowhile(y=12);
                exit
            else
                incase  //incase-case
                    when (c=5) : c := a*2
                    when (c=6) : c := c/2
                    when (c>6) : exit //exit from loop
                endincase
            endif
        endloop
    endif
endprogram
```

ο Ενδιάμεσος κώδικας που προκύπτει από αυτό στην μορφή τετράδων:

```
0: begin_block bigtest
1: in   a
2: in   b
3: in   c
4: := 0  x
5: := 0  y
6: = a 1 8
7: jump   39
8: < b c 10
9: jump   38
10: < b 5 12
11: jump   19
```

```
12: + b a T_1
13: := T_1  b
14: + x 1 T_2
15: := T_2  x
16: = x 10 12
17: jump   18
18: jump   37
19: = b 5 21
20: jump   24
21: * b 2 T_3
22: := T_3  b
23: jump   37
24: = b 6 26
25: jump   29
26: * b 4 T_4
27: := T_4  b
28: jump   37
29: = b 7 31
30: jump   34
31: * b 6 T_5
32: := T_5  b
33: jump   37
34: / b 2 T_6
35: := T_6  b
36: jump   19
37: jump   8
38: jump   65
39: < c 5 41
40: jump   49
41: + c a T_7
42: := T_7  c
43: + y 2 T_8
44: := T_8  y
45: = y 12 41
46: jump   47
47: jump   65
48: jump   65
```

```
49: := 0  T_9
50: = c 5 52
51: jump   55
52: := 1  T_9
53: * a 2 T_10
54: := T_10  c
55: = c 6 57
56: jump   60
57: := 1  T_9
58: / c 2 T_11
59: := T_11  c
60: > c 6 62
61: jump   64
62: := 1  T_9
63: jump   65
64: = 1 T_9 49
65: halt
66: end_block bigtest
```

ο Τελικός Κώδικας :

```
L:
        j Lmain
Lmain:
        add $sp,$sp,76
        move $s0,$sp
L0:
        sw $ra,($sp)
L1:
        li $v0,5
        syscall
L2:
        li $v0,5
        syscall
L3:
        li $v0,5
        syscall
```

```
L4:
        li $t1,0
        sw $t1,-24($s0)
L5:
        li $t1,0
        sw $t1,-28($s0)
L6:
        lw $t1,-12($s0)
        li $t2,1
        beq $t1,$t2,L8
L7:
        j L39
L8:
        lw $t1,-16($s0)
        lw $t2,-20($s0)
        blt $t1,$t2,L10
L9:
        j L38
L10:
        lw $t1,-16($s0)
        li $t2,5
        blt $t1,$t2,L12
L11:
        j L19
L12:
        lw $t1,-16($s0)
        lw $t2,-12($s0)
        add $t1,$t1,$t2
        sw $t1,-32($sp)
L13:
        lw $t1,-32($sp)
        sw $t1,-16($s0)
L14:
        lw $t1,-24($s0)
        li $t2,1
        add $t1,$t1,$t2
        sw $t1,-36($sp)
```

```
L15:
        lw $t1,-36($sp)
        sw $t1,-24($s0)
L16:
        lw $t1,-24($s0)
        li $t2,10
        beq $t1,$t2,L12
L17:
        j L18
L18:
        j L37
L19:
        lw $t1,-16($s0)
        li $t2,5
        beq $t1,$t2,L21
L20:
        j L24
L21:
        lw $t1,-16($s0)
        li $t2,2
        mul $t1,$t1,$t2
        sw $t1,-40($sp)
L22:
        lw $t1,-40($sp)
        sw $t1,-16($s0)
L23:
        j L37
L24:
        lw $t1,-16($s0)
        li $t2,6
        beq $t1,$t2,L26
L25:
        j L29
L26:
        lw $t1,-16($s0)
        li $t2,4
        mul $t1,$t1,$t2
```

```
        sw $t1,-44($sp)
L27:
        lw $t1,-44($sp)
        sw $t1,-16($s0)
L28:
        j L37
L29:
        lw $t1,-16($s0)
        li $t2,7
        beq $t1,$t2,L31
L30:
        j L34
L31:
        lw $t1,-16($s0)
        li $t2,6
        mul $t1,$t1,$t2
        sw $t1,-48($sp)
L32:
        lw $t1,-48($sp)
        sw $t1,-16($s0)
L33:
        j L37
L34:
        lw $t1,-16($s0)
        li $t2,2
        div $t1,$t1,$t2
        sw $t1,-52($sp)
L35:
        lw $t1,-52($sp)
        sw $t1,-16($s0)
L36:
        j L19
L37:
        j L8
L38:
        j L65
L39:
```

```
        lw $t1,-20($s0)
        li $t2,5
        blt $t1,$t2,L41
L40:
        j L49
L41:
        lw $t1,-20($s0)
        lw $t2,-12($s0)
        add $t1,$t1,$t2
        sw $t1,-56($sp)
L42:
        lw $t1,-56($sp)
        sw $t1,-20($s0)
L43:
        lw $t1,-28($s0)
        li $t2,2
        add $t1,$t1,$t2
        sw $t1,-60($sp)
L44:
        lw $t1,-60($sp)
        sw $t1,-28($s0)
L45:
        lw $t1,-28($s0)
        li $t2,12
        beq $t1,$t2,L41
L46:
        j L47
L47:
        j L65
L48:
        j L65
L49:
        li $t1,0
        sw $t1,-64($sp)
L50:
        lw $t1,-20($s0)
        li $t2,5
```

```
        beq $t1,$t2,L52
L51:
        j L55
L52:
        li $t1,1
        sw $t1,-64($sp)
L53:
        lw $t1,-12($s0)
        li $t2,2
        mul $t1,$t1,$t2
        sw $t1,-68($sp)
L54:
        lw $t1,-68($sp)
        sw $t1,-20($s0)
L55:
        lw $t1,-20($s0)
        li $t2,6
        beq $t1,$t2,L57
L56:
        j L60
L57:
        li $t1,1
        sw $t1,-64($sp)
L58:
        lw $t1,-20($s0)
        li $t2,2
        div $t1,$t1,$t2
        sw $t1,-72($sp)
L59:
        lw $t1,-72($sp)
        sw $t1,-20($s0)
L60:
        lw $t1,-20($s0)
        li $t2,6
        bgt $t1,$t2,L62
L61:
        j L64
```

```
L62:
        li $t1,1
        sw $t1,-64($sp)
L63:
        j L65
L64:
        li $t1,1
        lw $t2,-64($sp)
        beq $t1,$t2,L49
L65:
        li $v0,10
        syscall
L66:
        lw $ra,($sp)
        jr $ra
```

και τέλος ο Πίνακας Συμβόλων:

```
=  {'name': 'a', 'type': 'Var', 'offset': 12, 'global': 'True'}
=  {'name': 'b', 'type': 'Var', 'offset': 16, 'global': 'True'}
=  {'name': 'c', 'type': 'Var', 'offset': 20, 'global': 'True'}
=  {'name': 'x', 'type': 'Var', 'offset': 24, 'global': 'True'}
=  {'name': 'y', 'type': 'Var', 'offset': 28, 'global': 'True'}
=  {'name': 'T_1', 'type': 'tempVar', 'offset': 32}
=  {'name': 'T_2', 'type': 'tempVar', 'offset': 36}
=  {'name': 'T_3', 'type': 'tempVar', 'offset': 40}
=  {'name': 'T_4', 'type': 'tempVar', 'offset': 44}
=  {'name': 'T_5', 'type': 'tempVar', 'offset': 48}
=  {'name': 'T_6', 'type': 'tempVar', 'offset': 52}
=  {'name': 'T_7', 'type': 'tempVar', 'offset': 56}
=  {'name': 'T_8', 'type': 'tempVar', 'offset': 60}
=  {'name': 'T_9', 'type': 'tempVar', 'offset': 64}
=  {'name': 'T_10', 'type': 'tempVar', 'offset': 68}
=  {'name': 'T_11', 'type': 'tempVar', 'offset': 72}
=----------------------------------------------------------------------------
= nesting Level : 0
```

Παράδειγμα 2

Στο παρακάτω πρόγραμμα έχουμε κλήση συναρτήσεων και την υλοποίηση των τριών διαφορετικών περάσματων παραμέτρων

το πρόγραμμα σε γλώσσα Starlet:

```
program testPar
	declare a,y;
	function parIn(in x)
		x := 5;
		return x
	endfunction

	function parInout(inout x)
		x := 5;
		return x
	endfunction

	function parInandout(inandout x)
		x := 5;
		return x
	endfunction

	a := 1;
	y := parIn(in a);
	print a;

	a := 1;
	y := parInout(inout a);
	print a;

	a := 1;
	y := parInandout(inandout a);
	print a;
endprogram
```

ο Ενδιάμεσος κώδικας που προκύπτει από αυτό στην μορφή τετράδων:

```
0: begin_block parIn
1: := 5  x
2: RETV   x
3: jump   4
4: end_block parIn
5: begin_block parInout
6: := 5  x
7: RETV   x
8: jump   9
9: end_block parInout
10: begin_block parInandout
11: := 5  x
12: RETV   x
13: jump   14
14: end_block parInandout
15: begin_block testPar
16: := 1  a
17: par a CV
18: par T_1 RET
19: call parIn
20: := T_1  y
21: out   a
22: := 1  a
23: par a REF
24: par T_2 RET
25: call parInout
26: := T_2  y
27: out   a
28: := 1  a
29: par a CVREF
30: par T_3 RET
31: call parInandout
32: := T_3  y
33: out   a
34: halt
35: end_block testPar
```

ο Τελικός Κώδικας:

```
L:
	j Lmain
L0:
	sw $ra,($sp)
L1:
	li $t1,5
	sw $t1,-12($sp)
L2:
	lw $t1,-12($sp)
	lw $t0,-8($sp)
	sw $t1,($t0)
L3:
	j L4
L4:
	lw $ra,($sp)
	jr $ra
L5:
	sw $ra,($sp)
L6:
	li $t1,5
	lw $t0,-12($sp)
	sw $t1,($t0)
L7:
	lw $t0,-12($sp)
	lw $t1,($t0)
	lw $t0,-8($sp)
	sw $t1,($t0)
L8:
	j L9
L9:
	lw $ra,($sp)
	jr $ra
L10:
	sw $ra,($sp)
```

```
L11:
        li $t1,5
        sw $t1,-12($sp)
L12:
        lw $t1,-12($sp)
        lw $t0,-8($sp)
        sw $t1,($t0)
        lw $t1,-12($sp)
        lw $t0,-16($sp)
        sw $t1,($t0)
L13:
        j L14
L14:
        lw $ra,($sp)
        jr $ra
Lmain:
        add $sp,$sp,32
        move $s0,$sp
L15:
        sw $ra,($sp)
L16:
        li $t1,1
        sw $t1,-12($s0)
L17:
        addi $fp, $sp,16
        lw $t0,-12($s0)
        sw $t0,-12($fp)
L18:
        add $t0,$sp,-20
        sw $t0,-8($fp)
L19:
        sw $sp,-4($fp)
        addi $sp,$sp,16
        jal L0
        addi $sp,$sp,-16
L20:
        lw $t1,-20($sp)
```

```
        sw $t1,-16($s0)
L21:
        li $v0,1
        lw $t0,-12($s0)
        move $a0, $t0
        syscall
L22:
        li $t1,1
        sw $t1,-12($s0)
L23:
        addi $fp, $sp,16
        addi $t0,$sp,-12
        sw $t0,-12($fp)
L24:
        add $t0,$sp,-24
        sw $t0,-8($fp)
L25:
        sw $sp,-4($fp)
        addi $sp,$sp,16
        jal L5
        addi $sp,$sp,-16
L26:
        lw $t1,-24($sp)
        sw $t1,-16($s0)
L27:
        li $v0,1
        lw $t0,-12($s0)
        move $a0, $t0
        syscall
L28:
        li $t1,1
        sw $t1,-12($s0)
L29:
        addi $fp, $sp,20
        lw $t0,-12($s0)
        sw $t0,-12($fp)
        addi $t0,$sp,-12
```

```
        sw $t0,-16($fp)
L30:
        add $t0,$sp,-28
        sw $t0,-8($fp)
L31:
        sw $sp,-4($fp)
        addi $sp,$sp,20
        jal L10
        addi $sp,$sp,-20
L32:
        lw $t1,-28($sp)
        sw $t1,-16($s0)
L33:
        li $v0,1
        lw $t0,-12($s0)
        move $a0, $t0
        syscall
L34:
        li $v0,10
        syscall
L35:
        lw $ra,($sp)
        jr $ra
```

και τέλος ο Πίνακας Συμβόλων:

```
=  {'name': 'x', 'type': 'par', 'offset': 12, 'parMode': 'in'}
=----------------------------------------------------------------------------------------------------
= nesting Level : 1
==================================================================
=  {'name': 'x', 'type': 'par', 'offset': 12, 'parMode': 'inout'}
=----------------------------------------------------------------------------------------------------
= nesting Level : 1
==================================================================
=  {'name': 'x', 'type': 'par', 'offset': 12, 'parMode': 'inandout'}
=----------------------------------------------------------------------------------------------------
```

```
= nesting Level : 1
==================================================================
=  {'name': 'a', 'type': 'Var', 'offset': 12, 'global': 'True'}
=  {'name': 'y', 'type': 'Var', 'offset': 16, 'global': 'True'}
=  {'name': 'parIn', 'type': 'function', 'startQuad': 0, 'framelength': 16,
'Arguments': ['in']}
=  {'name': 'parInout', 'type': 'function', 'startQuad': 5, 'framelength': 16,
'Arguments': ['inout']}
=  {'name': 'parInandout', 'type': 'function', 'startQuad': 10, 'framelength': 20,
'Arguments': ['inandout']}
=  {'name': 'T_1', 'type': 'tempVar', 'offset': 20}
=  {'name': 'T_2', 'type': 'tempVar', 'offset': 24}
=  {'name': 'T_3', 'type': 'tempVar', 'offset': 28}
=----------------------------------------------------------------------------------------------------
= nesting Level : 0
==================================================================
```

Τέλοοος!!! ^_^