*Πανεπιστήμιο Ιωαννίνων*

*Πολυτεχνική Σχολή*

*Τμήμα Μηχανικών Η/Υ και Πληροφορικής*

**ΜΥΥ802: Μεταφραστές**

Διδάσκων: Γ. Μανής

Ακαδημαϊκό έτος: 2018-2019

Καρέτση Φωτεινή Α.Μ.: 2990

Μπελλά Ελευθερία Α.Μ.: 3039

**Πίνακας περιεχομένων**

## 1. Λεκτικός Αναλυτής3

## 2. Συντακτικός Αναλυτής7

## 3. Ενδιάμεσος Κώδικας8

## 3.1. Μεταβλητές8

## 3.2. Συναρτήσεις8

## 3.3. Λειτουργία ενδιάμεσου κώδικα9

## 3.4. Παραγωγή ισοδύναμου κώδικα σε C11

## 4. Πίνακας Συμβόλων13

## 5. Σημασιολογική Ανάλυση16

## 6. Τελικός Κώδικας18

1. **Λεκτικός αναλυτής**

Ο λεκτικός αναλυτής της γλώσσας Starlet είναι μια συνάρτηση που διαβάζεις γράμμα-γράμμα το πηγαίο πρόγραμμα και κάθε φορά επιστρέφει την αμέσως επόμενη λεκτική μονάδα. Συγκεκριμένα, ο λεκτικός αναλυτής (lex) επιστρέφει δύο τιμές, το string token, που είναι η λεκτική μονάδα που έχει αναγνωρίσει, και το string tokenID, που αποτελεί τον τύπο της λεκτικής μονάδας που αναγνωρίστηκε. Πιο αναλυτικά, μπορεί να πάρει μία από της ακόλουθες τιμές:

* Δεσμευμένες λέξεις, της έχουν οριστεί στη γλώσσα της Starlet (program, if, while, forcase, endforcase κτλ).
* Την τιμή id, που αντιστοιχεί σε αναγνωριστικό (μη δεσμευμένη λέξη). Το αναγνωριστικό αυτό μπορεί να αποτελεί όνομα μεταβλητής, συνάρτησης ή σχόλιο.
* Την τιμή constant, που αντιστοιχεί σε αριθμητική σταθερά.
* Έναν εκ των παρακάτω χαρακτήρων που επιτρέπονται στη γλώσσα, δηλαδή τα σύμβολα αριθμητικών πράξεων *(«+», «-», «\*», «/»),* της τελεστές συσχέτισης *(«<», «>», «=», «<=», «>=», «<>»)*, το σύμβολο ανάθεσης *(«:=»)*, της *διαχωριστές («;», «,», «:»)*, τα σύμβολα ομαδοποίησης *(«(»,«)»,«[»,«]»)* και διαχωρισμού σχολίων *(«/\*»,«\*/»,«//»)*. Για καθεμία από της παραπάνω περιπτώσεις, η τιμή του token ταυτίζεται με την τιμή του tokenID.

Ο λεκτικός αναλυτής, όσον αφορά την εσωτερική λειτουργία του, λειτουργεί σαν ένα αυτόματο καταστάσεων, το οποίο ξεκινά από μια αρχική κατάσταση, ανάλογα με το χαρακτήρα που διαβάζει μεταβαίνει σε κάποια άλλη κατάσταση και τελικά καταλήγει σε μία τελική. Σε περίπτωση μη επιτρεπτής λειτουργίας, της αυτές έχουν οριστεί από τη γλώσσα, ο αναλυτής **εμφανίζει κατάλληλο μήνυμα λάθους** και τερματίζει τη λειτουργία του (υλοποίηση συνάρτησης displayError).

Επιπλέον, ο λεκτικός αναλυτής διατηρεί ενημερωμένη την τρέχουσα γραμμή του αρχείου που διαβάζει, κάθε φορά που συναντά τον χαρακτήρα «\n».

*Οι καταστάσεις της οποίες μπορεί να μεταβεί ο lex() είναι οι εξής:*

1. Κατάσταση 0: Αρχική κατάσταση λεκτικού αναλυτή
2. Κατάσταση 1: Αναγνώριση λεκτικού (δεσμευμένης λέξης ή αναγνωριστικού id)
3. Κατάσταση 2: Αναγνώριση αριθμητικής σταθεράς
4. Κατάσταση 3: Αναγνώριση /
5. Κατάσταση 4: Αναγνώριση σχολίου μιας γραμμής
6. Κατάσταση 5: Αναγνώριση έναρξης σχολίων πολλαπλών γραμμών
7. Κατάσταση 6,11: Αναγνώριση εμφωλευμένων σχολίων
8. Κατάσταση 7: Αναγνώριση τερματισμού σχολίων πολλαπλών γραμμών
9. Κατάσταση 8: Αναγνώριση τελεστών που περιλαμβάνουν το χαρακτήρα <
10. Κατάσταση 9: Αναγνώριση τελεστών που περιλαμβάνουν το χαρακτήρα >
11. Κατάσταση 10: Αναγνώριση τελεστών που περιλαμβάνουν το χαρακτήρα :
12. Κατάσταση 100: Τελική κατάσταση

Της περιπτώσεις που κάποιος χαρακτήρας αναγνωρίζεται απευθείας χωρίς να εξετάσουμε κάποιο άλλο ενδεχόμενο, μεταβαίνει αμέσως σε τελική κατάσταση.

Κάθε φορά που καλείται ο lex() ξεκινά από την **κατάσταση 0.** Εάν διαβάσει κάποιο λευκό χαρακτήρα παραμένει σε αυτή την κατάσταση, ενώ αν διαβάσει κάποιο κεφαλαίο ή μικρό γράμμα μεταβαίνει στην κατάσταση 1. Όμοια, αν διαβάσει ψηφίο μεταβαίνει στην κατάσταση 2 και με «/» στην κατάσταση 3. Εάν διαβάσει «<», «>», «:» πηγαίνει της καταστάσεις 8,9 και 10 αντίστοιχα, ενώ αν συναντήσει κάποιον από της χαρακτήρες «+»,«-»,«\*», «,», «;», «=», «(», «)», «[», «]» ή τέλος αρχείου (EOF), μεταβαίνει στην κατάσταση 100 και επιστρέφει τα token και tokenID. Εάν συναντήσει χαρακτήρα που δεν ανήκει στο λεξιλόγιο της Starlet εμφανίζει κατάλληλο μήνυμα λάθους.

Στην **κατάσταση 1** παραμένει όσο διαβάζει γράμματα ή ψηφία. Διαφορετικά μεταβαίνει στην κατάσταση 100. Της, έχει χρησιμοποιηθεί της counter που μετρά το πλήθος των χαρακτήρων του αλφαριθμητικού και επιστρέφει της 30 πρώτους εάν το πλήθος αυτό υπερβεί την τιμή 30.

Στην **κατάσταση 2** παραμένει όσο διαβάζει ψηφία. Εάν διαβάσει γράμμα επιστρέφει μήνυμα λάθους, καθώς δεν είναι αποδεκτά ονόματα μεταβλητών που ξεκινούν από αριθμό. Επιπλέον, γίνεται έλεγχος ώστε η απόλυτη τιμή της σταθεράς να μην ξεπερνά το 32767, διαφορετικά εμφανίζεται κατάλληλο μήνυμα λάθους.

Από την **κατάσταση 3** και διαβάζοντας τον χαρακτήρα «\*» μεταβαίνει στην κατάσταση 5 και με τον χαρακτήρα «/» στην κατάσταση 4. Διαφορετικά μεταβαίνει σε τελική κατάσταση και επιστρέφει το χαρακτήρα «/».

Από την **κατάσταση 4** επιστρέφει στην αρχική όταν συναντήσει το χαρακτήρα αλλαγής γραμμής και τα σχόλια που είχαν αναγνωριστεί αγνοούνται. Εάν διαβαστεί το κενό (τέλος αρχείου) o lex() τερματίζει πρόωρα τη λειτουργία του. Διαβάζοντας το χαρακτήρα «/» μεταβαίνει στην κατάσταση 11 και από εκεί, εάν αναγνωριστεί εμφωλευμένο σχόλιο τυπώνεται μήνυμα λάθους και τερματίζει, αλλιώς επιστρέφει στην κατάσταση 4.

Από την **κατάσταση 5** μεταβαίνει στην κατάσταση 6 διαβάζοντας το χαρακτήρα «/», στην 7 αν διαβάσει «\*» ή τερματίζει πρόωρα αν διαβάσει το τέλος αρχείου. Σε οποιαδήποτε άλλη περίπτωση παραμένει στην ίδια κατάσταση.

Από την **κατάσταση 6** επιστρέφει στην κατάσταση 5 σε κάθε περίπτωση πλην του τέλους αρχείου και της περίπτωσης αναγνώρισης εμφωλευμένων σχολίων (διαβάζοντας της χαρακτήρες «/» ή «\*»). Σε αυτές της περιπτώσεις εμφανίζει κατάλληλο μήνυμα λάθους.

Από την **κατάσταση 7** επιστρέφει στην αρχική όταν συναντήσει το χαρακτήρα «/» και τα σχόλια που είχαν αναγνωριστεί αγνοούνται ή στην κατάσταση 5 για οποιοδήποτε άλλο χαρακτήρα.

Από **την κατάσταση 8** θα αναγνωριστούν τα εξής σύμβολα:

* «<>» , με τον χαρακτήρα «>»
* «<=», με τον χαρακτήρα «=»
* «<», ειδάλλως

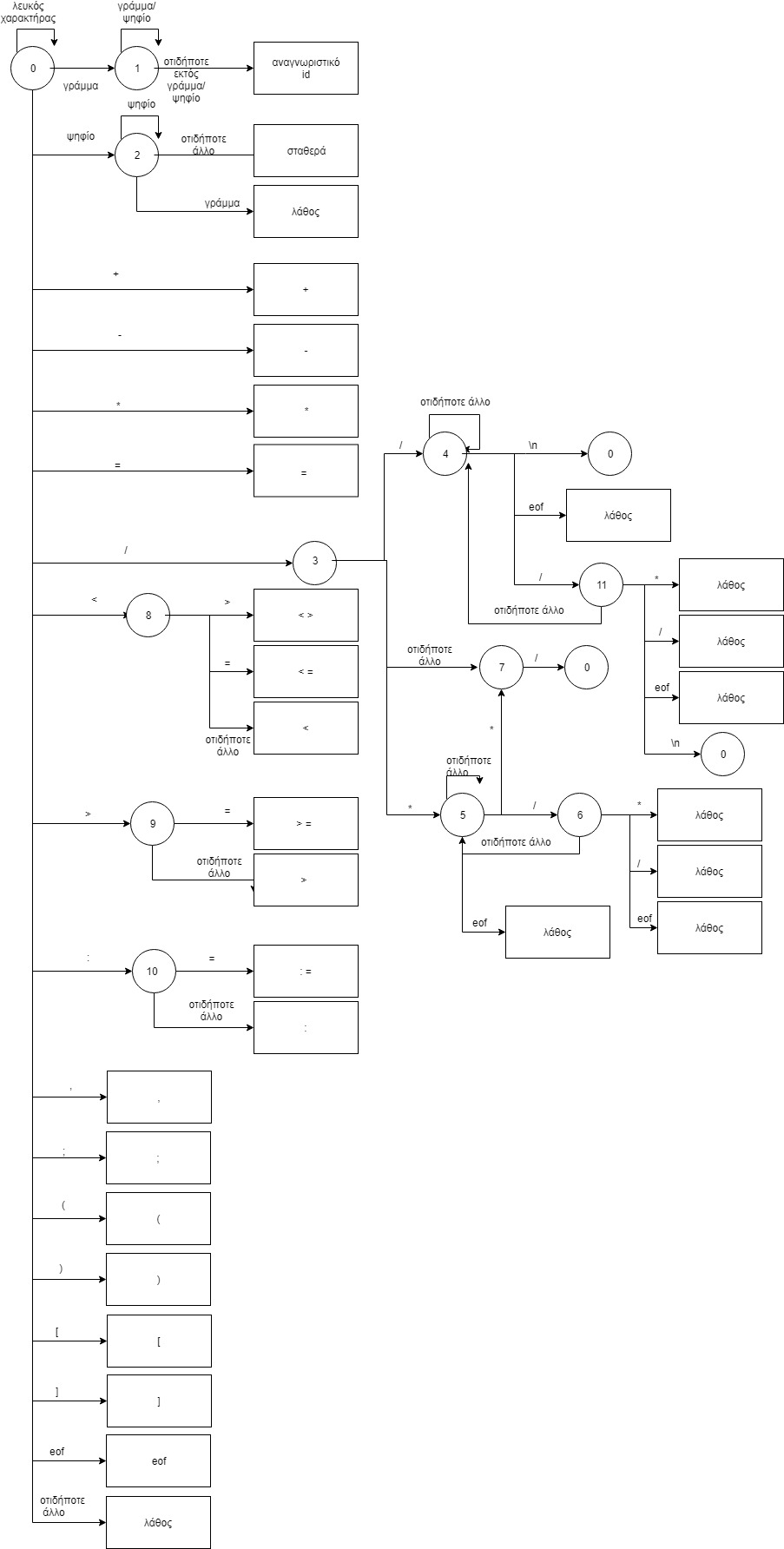
Από **την κατάσταση 9** θα αναγνωριστούν τα εξής σύμβολα:

* «>=», με τον χαρακτήρα «=»
* «>», ειδάλλως

Από **την κατάσταση 10** θα αναγνωριστούν τα εξής σύμβολα:

* «:=», με τον χαρακτήρα «=»
* «:», ειδάλλως

Για να γίνει σωστά η πρόβλεψη της λεκτικής μονάδας που θα επιστραφεί, της καταστάσεις 1,2,3,8,9 και 10 αναιρούμε την ανάγνωση του τελευταίου χαρακτήρα που έχει διαβαστεί από τον λεκτικό αναλυτή της περιπτώσεις που της δεν εμπίπτει σε κάποια από της προκαθορισμένες κατηγορίες που έχουμε εξετάσει παραπάνω. Εάν δεν ληφθεί υπόψη η παραπάνω περίπτωση, καταναλώνεται άσκοπα χαρακτήρας που μπορεί να είναι σημαντικός για τη μετέπειτα λειτουργία του lex. Παρακάτω παρατίθεται το αυτόματο καταστάσεων του λεκτικού αναλυτή σχηματικά.



Εικόνα 1: Σχηματική αναπαράσταση της λειτουργίας του λεκτικού αναλυτή.

1. **Συντακτικός αναλυτής**

Για κάθε κανόνα της γραμματικής έχει υλοποιηθεί η αντίστοιχη συνάρτηση. Σε κάθε συνάρτηση γίνεται έλεγχος για δεσμευμένες λέξεις / χαρακτήρες και σε περίπτωση μη τερματικών συμβόλων καλείται η αντίστοιχη συνάρτηση. Όταν συναντήσουμε τερματικό σύμβολο, τότε εάν ο λεκτικός αναλυτής επιστρέψει λεκτική μονάδα που αντιστοιχεί στο τερματικό σύμβολο το οποίο ελέγχουμε, συνεχίζει τη λειτουργία του. Στην αντίθετη περίπτωση, εμφανίζεται το κατάλληλο μήνυμα λάθους. Κάθε λεκτική μονάδα που εμφανίζεται στο συντακτικό αναλυτή, προκύπτει από την κλήση του lex().

Για κάθε κανόνα της γραμματικής ακολουθούνται οι εξής κανόνες :

* Το **αριστερό μέρος** κάθε κανόνα αποτελεί το όνομα του αντίστοιχου υποπρογράμματος.
* Γίνεται σύγκριση κάθε tokenID που επιστρέφει ο lex() με τη δεσμευμένη λέξη ή χαρακτήρα που αναμένει ο κανόνας.
* **Κλήση της κατάλληλης συνάρτησης** στις περιπτώσεις μη τερματικών συμβόλων.
* Στους κανόνες που εμφανίζεται *()\** είναι πιθανό το περιεχόμενο της παρένθεσης να καλείται από καμία έως πολλαπλές φορές. Αυτό υλοποιείται με τη χρήση της δομής while σε αυτές της συναρτήσεις.
* Στις παρακάτω συναρτήσεις γίνεται «παραβίαση» της LL1 γραμματικής που υλοποιούμε : *optional-sign, idtail, term, expression, statement, subprograms*. Με τον όρο παραβίαση εννοούμε τον έλεγχο της συγκεκριμένης λεκτικής μονάδας που αναμένει ο κανόνας. Σε περίπτωση αποτυχίας ταυτοποίησης εμφανίζεται μήνυμα λάθους.
* Στους κανόνες που επιτρέπεται η εμφάνιση του κενού χαρακτήρα *ε*, δεν εμφανίζεται κάποιο μήνυμα λάθους.

*Υποσημείωση*: Στην περίπτωση πρόβλεψης της επόμενης συνάρτησης που αναμένεται να εκτελεστεί έχουν παραλειφθεί πλεονάζοντες έλεγχοι ορισμένων λεκτικών μονάδων, π.χ. η συνάρτηση statement κάνει έλεγχο για την ύπαρξη της μονάδας if και καλεί την αντίστοιχη συνάρτηση if\_stat. Στην if\_stat δεν πραγματοποιείται έλεγχος για την εμφάνιση του if, ούτε ορίζεται αντίστοιχο λάθος καθώς, για να κληθεί η if\_stat, σημαίνει ότι η λέξη if θα υπάρχει πάντα.

1. **Ενδιάμεσος Κώδικας**

Για την παραγωγή του ενδιάμεσου κώδικα, χρησιμοποιήσαμε της βοηθητικές συναρτήσεις και μεταβλητές που ακολουθούν.

**3.1. Μεταβλητές**

* **quad\_program\_list :** Δομή λεξικού, εδώ διατηρούμε της παραγόμενες τετράδες του προγράμματος.
* **total\_quads** **:** Διατηρεί το συνολικό πλήθος των τετράδων , λειτουργεί σαν μετρητής.
* **temp\_value :** Βοηθητική μεταβλητή (λειτουργεί της σαν μετρητής), διατηρεί το πλήθος των προσωρινών μεταβλητών (*Τ\_1, Τ\_2, ...*).
* **programName :** Εδώ αποθηκεύεται το όνομα του προγράμματος.
* **findLoop :** Λίστα που υποδεικνύει την πρώτη τετράδα κάθε loop statement (ώστε να επιστρέφει το πρόγραμμα μετά από τον έλεγχο της κατάλληλης συνθήκης).
* **exitLoop() :** Λίστα που υποδεικνύει την ετικέτα κάθε exit statement. Κάθε φορά που ένα loop statement τερματίζει η τελευταία θέση της λίστας διαγράφεται και γίνεται backpatch με την επόμενη τετράδα.

**3.2. Συναρτήσεις**

* **next\_quad() :** Επιστρέφει τον αριθμό της ετικέτας της επομένης τετράδας που θα παραχθεί.
* **gen\_quad() :** Δημιουργεί την επόμενη τετράδα του προγράμματος.
* **newTemp() :** Δημιουργεί την προσωρινή μεταβλητή (Τ\_1, Τ\_2, ...).
* **emptylist() :** Επιστρέφει μία καινούρια κενή λίστα.
* **makelist() :** Δημιουργεί μία καινούρια λίστα που περιέχει μόνο μία ετικέτα, αυτή που δίνεται ως όρισμα στη συνάρτηση.
* **merge() :** Συνενώνει της δύο συναρτήσεις που δίνονται ως όρισμα στη συνάρτηση. Πιο συγκεκριμένα, προσθέτει στην πρώτη λίστα τη δεύτερη.
* **backpatch() :** Αυτή η συνάρτηση αναθέτει τιμές στα τελούμενα του προγράμματος που δεν έχουν ακόμα λάβει κάποια τιμή. Το πρώτο όρισμα συγκεκριμένα είναι η λίστα στην οποία θέλουμε να συμπληρώσουμε το τελευταίο τελούμενο, και το δεύτερο όρισμα είναι η τιμή του τελούμενου που θέλουμε να δώσουμε. Η λειτουργία είναι η εξής: ανατρέχοντας της ετικέτες του προγράμματος μία της μία, αν η τρέχουσα τιμή υπάρχει στη λίστα-όρισμα της συνάρτησης, τότε συμπληρώνουμε την ετικέτα αυτή με το δεύτερο όρισμα της συνάρτησης.
  1. **Λειτουργία ενδιάμεσου κώδικα**

Για να εκτελεστεί ορθά ο μεταγλωττιστής της, τροποποιήσαμε της κατάλληλες συναρτήσεις της Starlet ώστε να μπορέσει να παραχθεί ο ενδιάμεσος κώδικας. Πιο συγκεκριμένα οι συναρτήσεις αυτές είναι οι ακόλουθες:

* **block() :** Με τη χρήση της gen\_quad(), δημιουργούμε της τετράδες begin\_block και end\_block, οριοθετώντας έτσι αντίστοιχα την αρχή και το τέλος του block. Επιπλέον, εφόσον το τρέχον block αντιστοιχεί σε αυτό του κύριου προγράμματος, δημιουργούμε και την τετράδα halt, που σηματοδοτεί το τέλος του προγράμματος.
* **assignment\_stat() :** Εδώ λαμβάνεται η τιμή της έκφρασης (που επιστρέφει η expression) και στη συνέχεια με την gen\_quad() δημιουργείται η τετράδα που δείχνει την ανάθεση αυτή.
* **if\_stat() :** Αφού λάβουμε της δύο λίστες που επιστρέφει η condition, της αναθέτουμε ως ορίσματα στην backpatch έτσι ώστε να οδηγήσουμε το πρόγραμμα στην κατάλληλη τετράδα ανάλογα με το αν αληθεύει ή της η συνθήκη ελέγχου.Στην περίπτωση που αληθεύει, το πρόγραμμα συνεχίζει στην επόμενη τετράδα, ενώ στην αντίθετη περίπτωση δημιουργεί την εντολή jump για έξοδο μετά την εκτέλεση και με την backpatch οδηγεί το πρόγραμμα στην ετικέτα του elsepart.
* **while\_stat() :** Η while\_quad διατηρεί την ετικέτα της τετράδας στην οποία πραγματοποιείται ο έλεγχος της συνθήκης. Εάν η συνθήκη αληθεύει, εκτελείται η συνάρτηση statements και ξαναελέγχεται η συνθήκη, διαφορετικά ο έλεγχος μεταβαίνει εκτός while block (με κατάλληλα backpatch).
* **do\_while\_stat() :** Η μεταβλητή do\_while\_quad έχει την ίδια λειτουργία με τη μεταβλητή while\_quad της while\_stat (δηλαδή οδηγεί στην αρχή του loop για να γίνει ο έλεγχος της συνθήκης) και μετά από τον έλεγχο η backpatch επιστρέφει το πρόγραμμα στην ετικέτα της do\_while\_quad αν η συνθήκη είναι λανθασμένη ή συνεχίζει στην επόμενη τετράδα αν αληθεύει.
* **loop\_stat() :** H προσωρινή μεταβλητή t, αρχικοποιείται στο 0 μέχρι να βρούμε το exit statement που αντιστοιχεί στο loop που βρισκόμαστε. Η πρώτη τετράδα του loop προστίθεται στη λίστα findLoop και μετά την εκτέλεση του endloop, βρεθεί το αντίστοιχο exit statement, επιστρέφουμε τον έλεγχο στην αρχή του τελευταίου loop, διαγράφωντας από της αντίστοιχες λίστες τα πιο πρόσφατα loop & exit statements.
* **exit\_stat():** Αν στην κλήση της συνάρτησης βρεθεί exit statementπου αντιστοιχεί στο καλόν loop statement, δημιουργείται η τετράδα jump που οδηγεί σε έξοδο από το loop.
* **forcase\_stat() :** Η μεταβλητή forcase\_quad δείχνει στην πρώτη τετράδα του forcase block και η λίστα exitList χρησιμοποιείται για να οδηγήσει τη ροή του προγράμματος εκτός block μετά την εκτέλεση των statements της πρώτης αληθούς συνθήκης. Για κάθε συνθήκη, αν αληθεύει εκτελούνται τα statements της αλλιώς ελέγχουμε την επόμενη διαθέσιμη συνθήκη. Με βάση τα παραπάνω, μέσω της backpatch, συμπληρώνουμε κατάλληλα τις λίστες που προέρχονται από την condition και εάν δεν αληθεύει καμία συνθήκη, εκτελείται το default statement. Σε αυτή την περίπτωση, επιστρέφουμε στην τετράδα με ετικέτα forcase\_quad και επανεξετάζουμε τις συνθήκες.
* **incase\_stat() :** Σε αυτή τη συνάρτηση δημιουργώ μία προσωρινή μεταβλητή t που αρχικοποιείται με την τιμή 0. Έπειτα, μετά από κάθε condition που αληθεύει αλλάζω την τιμή του t σε 1 και κάνω backpatch με την cond\_true ώστε να εκτελεστούν τα statements. Στην περίπτωση που το condition είναι ψευδές, κάνω backpatch με την cond\_false για να εκτελεστεί το επόμενο when. Στο τέλος, μετά τη δεσμευμένη λέξη endincase, γίνεται έλεγχος για την τιμή του t. Αν η τιμή είναι ίση με 1, σημαινει ότι τουλάχιστον μία when είναι αληθής και ο έλεχος μεταβαίνει στην αρχή της incase, διαφορετικά γίνεται έξοδος από τη συνάρτηση.
* **return\_stat() :** Επιστρέφουμε την τιμή της expression σε μία μεταβλητή Ε και στη συνέχεια με την gen\_quad δημιουργούμε την τετράδα που επιστρέφει την τιμή της μεταβλητής Ε (αντιστοιχεί στη δεσμευμένη λέξη “retv”).
* **print\_stat() :** Επιστρέφουμε την τιμή της expression σε μία μεταβλητή Ε και στη συνέχεια με την gen\_quad δημιουργούμε την τετράδα που τυπώνει την τιμή της μεταβλητής Ε (αντιστοιχεί στη δεσμευμένη λέξη “out”).
* **input\_stat() :** Εδώ με την gen\_quad δημιουργούμε την τετράδα με τη δεσμευμένη λέξη “inp” και την τιμή που δώθηκε από το χρήστη στην επιθυμητή μεταβλητή.
* **actualparitem() :** Επιστρέφουμε την τιμή της expression στη μεταβλητή actualparitem\_result και δημιουργούμε την επόμενη τετράδα που υποδεικνύει τη δημιουργία παραμέτρου με την τιμή του actualparitem\_result και τον τύπο της παραμέτρου (“CV”, “REF” ή “CP” για in, inout και inandout αντίστοιχα).
* **condition() :** Η συνάρτηση αυτή επιστρέφει το αποτέλεσμα της λογικής παράστασης or (true, false) που δίνεται μέσω της αποτίμησης των συναρτήσεων boolterm που περιέχονται σε αυτή. Το αποτέλεσμα αποθηκεύεται σε δύο λίστες (Βtrue, Βfalse) οι οποίες περιέχουν της τετράδες που είναι αληθείς και ψευδείς (η κάθε boolterm αντίστοιχα περιέχει δύο δικές της λίστες, της Q1true, Q1false). Η λογική είναι η εξής: όταν αποτιμηθεί η πρώτη boolterm, η αληθής λίστα της εξισώνεται με την αληθή λίστα της condition και αντίστοιχα η λανθασμένη λίστα της με τη λανθασμένη της condition. Στη συνέχεια μετά από κάθε or και πριν από κάθε επόμενο boolterm , γίνεται συμπλήρωση όσων τετράδων μπορούν να συμπληρωθούν μέσα στη συνάρτηση (δηλαδή οι λανθασμένες τετράδες) και μετά την εκτέλεση του επόμενου boolterm, συγχωνεύουμε της δύο αληθείς λίστες και εξισώνουμε την τελική λανθασμένη λίστα με αυτή του τελευταίου boolterm.
* **boolterm() :** Η συνάρτηση αυτή επιστρέφει το αποτέλεσμα της λογικής παράστασης and και ακολουθεί την ίδια εκτέλεση με την condition, με τη διαφορά ότι εδώ οι τετράδες που μπορούν να συμπληρωθούν είναι οι αληθείς, και μετά την εκτέλεση του επόμενου boolfactor, συγχωνεύουμε της δύο λανθασμένες λίστες και εξισώνουμε την τελική αληθή λίστα με αυτή του τελευταίου boolfactor.
* **boolfactor() :** Εδώ καλείται αναδρομικά η condition.Έπειτα, η αληθής λίστα της condition εξισώνεται με την αντίστοιχη της boolfactor (το ανάλογο συμβαίνει με της λανθασμένες λίστες). Στην περίπτωση που συναντήσουμε τη δεσμευμένη λέξη “not” συμβαίνει το ανάποδο (η αληθής λίστα της condition εξισώνεται με τη λανθασμένη της boolfactor κ.ο.κ.). Αν έχουμε αριθμητική παράσταση επιστρέφουμε την τιμή των expression σε δύο μεταβλητές (Ε1 και Ε2 ), θέτουμε την αληθή λίστα της συνάρτησης ίση με την τετράδα που εκτελεί την πράξη ανάμεσα της Ε1 και Ε2 και την ψευδή λίστα ίση με την παραγωγή της τετράδας jump.
* **expression() :** Αυτή η συνάρτηση δημιουργεί και επιστρέφει την προσωρινή μεταβλητή στην οποία αποθηκεύεται το αποτέλεσμα της αριθμητικής πράξης (με κλήση της newTemp) και η παράγει την τετράδα που της δείχνει την πράξη αυτή.
* **term() :** Με την ίδια λογική εδώ δημιουργούνται και επιστρέφονται αναδρομικά οι πράγοντες που αποτιμούνται στη συνάρτηση expression.
* **Factor() :** Στη μεταβλητή factor\_result της συνάρτησης επιστρέφεται η τιμή της σταθεράς, της έκφρασης ή του id που αποτελεί τους όρους της συνάρτησης term.
  1. **Παραγωγή ισοδύναμου κώδικα σε C**

Για την παραγωγή του ισοδύναμου αρχείου σε C, ουσιαστικά χρησιμοποιήθηκαν οι τετράδες που παρήχθησαν από τον ενδιάμεσο κώδικα και έχουν αποθηκευτεί στο λεξικό *quad\_program\_list*. Για την ευκολότερη αναγνώριση των ονομάτων των μεταβλητών που πρέπει να δηλωθούν στο C αρχείο, έχουν οριστεί οι global λίστες *allFunctions* και *bindedCharacters*.

Η πρώτη διατηρεί τα ονόματα όλων των συναρτήσεων που μπορεί να περιέχει το πρόγραμμα που μεταφράζεται. Τα ονόματα αυτά προστίθενται στη λίστα *allFunctions* κατά τη διάρκεια της κλήσης της συνάρτησης *subprogram,* όπου γίνεται η μετάφραση της κάθε συνάρτησης.

Η λίστα *bindedCharacters* διατηρεί δεσμευμένους χαρακτήρες και εκφράσεις που εμφανίζονται στις τετράδες και αφορούν το πέρασμα παραμέτρων στις συναρτήσεις. Πρόκειται για το χαρακτήρα *‘\_’,* που συμβολίζει το κενό στοιχείο σε μια τετράδα, καθώς και τις εκφράσεις ‘*CV’* (πέρασμα παραμέτρου με τιμή), *‘CP’* (πέρασμα παραμέτρου με αντιγραφή τιμής), *‘REF’* (πέρασμα παραμέτρου με αναφορά) και *‘RET’* (πέρασμα παραμέτρου επιστροφής της συνάρτησης).

Η παραγωγή του ισοδύναμου κώδικα σε C πραγματοποιείται μέσω της συνάρτησης ***produceCFile***, η οποία δέχεται ως όρισμα το domain name του αρχείου που θα δημιουργηθεί (το ίδιο με το όνομα του αρχικού προγράμματος καθώς και του ισοδύναμου ενδιάμεσου κώδικα).

Οι μεταβλητές που θα δηλωθούν στο C αρχείο διατηρούνται στη λίστα *variables*. Σε αυτή τη λίστα προστίθενται όλα τα μοναδικά ονόματα μεταβλητών που έχουν χρησιμοποιηθεί στις τετράδες. Εάν υπάρχουν δηλωμένες μεταβλητές στο αρχικό πρόγραμμα που δεν έχουν χρησιμοποιηθεί για την παραγωγή κάποιας τετράδας, παραβλέπεται η δήλωσή τους στο αρχείο C. Η συνάρτηση *checkIfNegative* χρησιμοποιείται για την αναγνώριση των προσημασμένων μεταβλητών και την προσθήκη των κατάλληλων μεταβλητών στη λίστα μεταβλητών. Πριν πραγματοποιηθεί ο έλεγχος για την προσθήκη της εκάστοτε μεταβλητής στη λίστα *variables* αφαιρείται το πρόσημο (εάν υπάρχει), χωρίς, ωστόσο, να επηρεάζεται η ορθή λειτουργία του προγράμματος. Επιπλέον, ελέγχεται εάν η μεταβλητή που δόθηκε είναι μεταβλητή ή αριθμητική σταθερά. Στη δεύτερη περίπτωση, δεν προστίθενται στη λίστα μεταβλητών.

Πριν την καταγραφή των μεταβλητών στο αρχείο C, γίνεται έλεγχος για την ύπαρξη χαρακτήρων από τις λίστες *bindedCharacters* και *allFunctions* στη λίστα *variables* και απομακρύνονται κατάλληλα.

Πριν την καταγραφή των εκτελέσιμων εντολών, προστίθενται οι κατάλληλες κεφαλίδες στο αρχείο C («includes»). Στη συνέχεια, για κάθε τετράδα στο λεξικό *quad\_program\_list* γίνονται οι παρακάτω έλεγχοι:

* Εάν πρόκειται για τετράδα *begin\_block* με το όνομα του προγράμματος, γίνεται έναρξη της συνάρτησης int main() και δήλωση των μεταβλητών της λίστας *variables*.
* Εάν πρόκειται για τετράδα ανάθεσης, δημιουργείται εκτελέσιμη εντολή της μορφής a = b;, όπου a η μεταβλητή στην οποία πραγματοποιείται η ανάθεση και b η ανατιθέμενη τιμή.
* Εάν πρόκειται για τετράδα με αριθμητική πράξη, δημιουργείται εκτελέσιμη εντολή της μορφής a = b op c;, όπου a η μεταβλητή στην οποία πραγματοποιείται η ανάθεση και b ο πρώτος τελεστέος, op ο αριθμητικός τελεστής (ένας εκ των +,-,\*,/) και c ο δεύτερος τελεστέος.
* Εάν πρόκειται για τετράδα με συγκριτικό τελεστή, δημιουργείται εκτελέσιμη εντολή της μορφής if (x op y) goto L\_z;, όπου x, ο πρώτος τελεστέος, y ο δεύτερος τελεστέος, op ο συγκριτικός τελεστής (ένας εκ των <,>,=,<>,<=,>=) στην κατάλληλη αποδεκτή μορφή για το αρχείο C (π.χ. ο τελεστής <> μετατρέπεται σε !=) και z η ετικέτα κάποιας τετράδας.
* Εάν πρόκειται για τετράδα jump σε κάποια ετικέτα, δημιουργείται εκτελέσιμη εντολή της μορφής goto L\_z;, όπου z η ετικέτα της τετράδας.
* Εάν πρόκειται για τετράδα retv και όνομα μεταβλητής/σταθεράς, δημιουργείται εκτελέσιμη εντολή της μορφής return x;, όπου x το όνομα της μεταβλητής/σταθεράς.
* Εάν πρόκειται για τετράδα inp και όνομα μεταβλητής, δημιουργείται εκτελέσιμη εντολή της μορφής scanf(“%d”, x);, όπου x το όνομα της μεταβλητής που ζητάμε την τιμή της ως είσοδο στο πρόγραμμα.
* Εάν πρόκειται για τετράδα out και όνομα μεταβλητής/σταθεράς, δημιουργείται εκτελέσιμη εντολή της μορφής printf("%d, x\\n");, όπου x το όνομα της μεταβλητής/σταθεράς που θα εμφανιστεί στην οθόνη.
* Εάν πρόκειται για τετράδαhalt προστίθεται η εντολή return 0; και το πρόγραμμα ολοκληρώνεται.

**Σημείωση:** Όλες οι εκτελέσιμες εντολές διαθέτουν μια ετικέτα L\_z, όπου z η τιμή της ετικέτας. Επιπλέον, στο τέλος κάθε εντολής έχει προστεθεί με σχόλια η εκάστοτε ισοδύναμη εντολή σε ενδιάμεσο κώδικα.

1. **Πίνακας Συμβόλων**

Για την υλοποίηση των οντοτήτων που περιέχει ο πίνακας συμβόλων δημιουργήθηκαν οι παρακάτω κλάσεις:

* **Entity**: Μητρική κλάση που περιέχει τα πεδία ***Name*** και ***Entity\_type*.** Το πεδίο Entity\_type μπορεί να πάρει μία από τις παρακάτω τιμές: *Variable* (αφορά μεταβλητές και προσωρινές μεταβλητές), *Function* (για συναρτήσεις) και *Parameter* (για παραμέτρους συναρτήσεων). Η γλώσσα Starlet δεν χρησιμοποιεί σταθερές. Περιλαμβάνει τις μεθόδους *\_\_init\_\_* (constructor) και *printEntity* που επιστρέφει ένα string με τις τιμές των πεδίων του αντικειμένου.
* **Variable:** «Παιδί» της κλάσης *Entity.*Ορίζει ένα επιπλέον πεδίο ***Offset,*** που υποδηλώνει την απόσταση στη μνήμη από την αρχή του block που ανήκει. Περιλαμβάνει τις ίδιες μεθόδους με την γονική κλάση Entity αλλά overridden, καλύπτοντας και τις ανάγκες του επιπλέον πεδίου.
* **Function:** «Παιδί» της κλάσης *Entity.*Ορίζει ένα επιπλέον πεδίο ***Framelength,*** που υποδηλώνει το μέγεθος του block της συνάρτησης στη μνήμη. Επιπλέον, περιλαμβάνει το πεδίο ***returnType***, που δείχνει τον τύπο της τιμής επιστροφής (integer), το πεδίο ***startQuad***, που είναι ένας δείκτης στην πρώτη τετράδα που θα εκτελέσει η συνάρτηση και το πεδίο ***argList***, που είναι μια λίστα η οποία διατηρεί τον τύπο των ορισμάτων της συνάρτησης. Περιλαμβάνει τις ίδιες μεθόδους με την γονική κλάση Entity αλλά overridden, καλύπτοντας και τις ανάγκες των επιπλέον πεδίων.
* **Parameter:** «Παιδί» της κλάσης *Entity.*Περιλαμβάνει ένα επιπλέον πεδίο ***Offset,*** που υποδηλώνει την απόσταση στη μνήμη από την αρχή του block που ανήκει, καθώς και το πεδίο ***parMode***, που υποδεικνύει τον τύπο περάσματος του ορίσματος. Το πεδίο parMode μπορεί να πάρει μία εκ των τιμών «*in*», «*inout*», «*inandout*». Περιλαμβάνει τις ίδιες μεθόδους με την γονική κλάση Entity αλλά overridden, καλύπτοντας και τις ανάγκες του επιπλέον πεδίου.
* **Scope:** Κλάση που υποδηλώνει το βάθος φωλιάσματος. Περιλαμβάνει τα πεδία ***nestingLevel,*** που δείχνει το βάθος φωλιάσματος και παίρνει τιμές >=0 καθώς και το πεδίο ***framelength,*** το οποίο ενημερώνεται κάθε φορά που προστίθεται στο scope νέα μεταβλητή ή παράμετρος. Περιλαμβάνει τις μεθόδους *\_\_init\_\_* (constructor) και *printScope*, που επιστρέφει ένα string με τις τιμές των πεδίων του αντικειμένου.
* **Argument:** Κλάση που αφορά τα ορίσματα που δέχεται μια συνάρτηση. Περιλαμβάνει ένα πεδίο ***argMode***, που συμβολίζει τον τύπο περάσματος του ορίσματος, καθώς και τις μεθόδους *\_\_init\_\_* (constructor) και *printArgument*, που επιστρέφει ένα string με τις τιμές των πεδίων του αντικειμένου.

Εκτός από τις παραπάνω κλάσεις, διατηρείται η global λίστα **symbolList,** που περιέχει όλα τα ενεργά scopes του προγράμματος. Η συνάρτηση **printSymbolList**() τυπώνει τα περιεχόμενα της λίστας *symbolList* και καλείται πριν τη διαγραφή ενός scope και αμέσως μετά από αυτή.

Επιπρόσθετα, προστέθηκαν οι παρακάτω συναρτήσεις που αποσκοπούν στη δημιουργία αντικειμένων τύπου Variable, Function, Parameter, Scope και Argument, στη διαγραφή ενός Scope καθώς και στην αναζήτηση Entities, μεταβλητών ή παραμέτρων. Συγκεκριμένα:

* **createArgument** (mode, arglist): Δημιουργεί ένα αντικείμενο τύπου Argument με argMode = mode και το προσθέτει στη λίστα arglist (λίστα ορισμάτων κάποιας συνάρτησης που δίνεται ως όρισμα).
* **createVariable**(name, entity\_type, offset, scopelist): Δημιουργεί ένα αντικείμενο τύπου Variable και το προσθέτει στη λίστα scopelist (λίστα οντοτήτων κάποιου scope που δίνεται ως όρισμα).
* **createFunction**(name, entity\_type, scopelist): Δημιουργεί ένα αντικείμενο τύπου Function και το προσθέτει στη λίστα scopelist (λίστα οντοτήτων κάποιου scope που δίνεται ως όρισμα).
* **createParameter**(name, entity\_type, offset, parMode, scopelist): Δημιουργεί ένα αντικείμενο τύπου Parameter και το προσθέτει στη λίστα scopelist (λίστα οντοτήτων κάποιου scope που δίνεται ως όρισμα).
* **createScope**(nestingLevel): ): Δημιουργεί ένα αντικείμενο τύπου Scope και το προσθέτει στη λίστα symbolList. Καθορίζει το βάθος φωλιάσματος του αντικειμένου με βάση το πλήθος των ενεργών scopes στη λίστα symbolList.
* **deleteScope**(): Διαγράφει το τελευταίο scope της λίστας symbolList.
* **searchEntity**(name, nl): Αναζητά και επιστρέφει την οντότητα με όνομα name, ξεκινώντας από βάθος φωλιάσματος nl. Εάν δεν υπάρχει τέτοιο αντικείμενο, τυπώνεται κατάλληλο μήνυμα λάθους και το πρόγραμμα τερματίζει.
* **searchVariableOrParameter**(sc): Αναζητά και επιστρέφει την τελευταία μεταβλητή ή παράμετρο που έχει προστεθεί στη λίστα οντοτήτων του scope sc. Εάν δεν υπάρχει τέτοιο αντικείμενο, επιστρέφει None.

Για τη συμπλήρωση του πίνακα συμβόλων έχουν τροποποιηθεί τα εξής σημεία στον κώδικα του μεταγλωττιστή:

1. Στη συνάρτηση **newTemp**() δημιουργείται μεταβλητή με το όνομα της προσωρινής μεταβλητής που επιστρέφεται και ορίζεται κατάλληλα το framelength του scope που τροποποιήθηκε.
2. Στη συνάρτηση **program**() δημιουργείται το scope του main προγράμματος.
3. Στη συνάρτηση **block**(name, returnList = []), εάν πρόκειται για block συνάρτησης και όχι του βασικού προγράμματος, ορίζεται το πεδίο startQuad να δείχνει στην εντολή begin\_block της συνάρτησης. Κατόπιν, πριν το τέλος του block, ορίζεται η τιμή του framelength της κάθε συνάρτησης και στη συνέχεια διαγράφεται το τελευταίο scope που προστέθηκε. Εάν πρόκειται για το block βασικού προγράμματος, το framelength αποθηκεύεται σε μία global μεταβλητή με το όνομα **mainFramelength**.
4. Στη συνάρτηση **varlist**() για κάθε μεταβλητή που δηλώνεται, δημιουργείται ένα αντικείμενο Variable και ενημερώνεται κατάλληλα το framelength του scope που τροποποιήθηκε.
5. Στη συνάρτηση **subprogram** () για κάθε συνάρτηση που ορίζεται, δημιουργείται ένα αντικείμενο Function με το όνομα της συνάρτησης στο scope που ανήκει και στη συνέχεια δημιουργείται ένα καινούριο αντικείμενο Scope με μεγαλύτερο βάθος φωλιάσματος που αντιστοιχίζεται στη συνάρτηση αυτή.
6. Στη συνάρτηση **formalparitem**() για κάθε παράμετρο που διαβάζεται, δημιουργείται δημιουργείται ένα αντικείμενο Argument στη λίστα ορισμάτων της συνάρτησης που μεταφράζεται μέσω της συνάρτησης createArgument και στο τελευταίο scope προστίθεται ένα νέο αντικείμενο Parameter με τον κατάλληλο τύπο περάσματος και ενημερώνοντας κατάλληλα το framelength του scope που τροποποιήθηκε.
7. **Σημασιολογική Ανάλυση**

Για τη σημασιολογική ανάλυση έχει προστεθεί κώδικας που ικανοποιεί τους παρακάτω περιορισμούς:

1. *Κάθε συνάρτηση έχει μέσα της τουλάχιστον ένα return.*
2. *Δεν υπάρχει return έξω από συνάρτηση.*
3. *Υπάρχει exit μόνο μέσα σε βρόχους loop-endloop.*
4. *Κάθε μεταβλητή ή συνάρτηση που έχει δηλωθεί να μην έχει δηλωθεί πάνω από μία φορά στο βάθος φωλιάσματος στο οποίο βρίσκεται.*
5. *Κάθε μεταβλητή ή συνάρτηση που χρησιμοποιείται έχει δηλωθεί και μάλιστα με τον τρόπο που χρησιμοποιείται (σαν μεταβλητή ή σαν συνάρτηση).*
6. *Οι παράμετροι με τις οποίες καλούνται οι συναρτήσεις είναι ακριβώς αυτές με τις οποίες έχουν δηλωθεί και με τη σωστή σειρά.*

Συγκεκριμένα:

1. Στη συνάρτηση **block**(name, returnList = []) προστέθηκε ως επιπλέον –προαιρετικό- όρισμα μια λίστα *returnList*, στην οποία αποθηκεύονται όλες οι εμφανίσεις των return statements μέσα σε μια συνάρτηση. Αυτή η λίστα αρχικοποιείται κάθε φορά που εκτελείται η συνάρτηση *subprogram()* και μεταβάλλεται με την κλήση της συνάρτησης return\_stat. Στη συνάρτηση **subprogram()** γίνεται έλεγχος για το μέγεθος της λίστας returnList και εάν δεν έχει βρεθεί κανένα return statement τυπώνεται κατάλληλο μήνυμα λάθους και η μετάφραση του προγράμματος διακόπτεται.
2. Η global boolean μεταβλητή ***enableReturnSearch*** χρησιμοποιείται για την ανίχνευση return statements εκτός συναρτήσεων. Στη συνάρτηση **block**(name, returnList = []) γίνεται έλεγχος για το εάν πρόκειται για κλήση της block μέσα στο βασικό πρόγραμμα και αλλάζει η τιμή της μεταβλητής. Στην πρώτη εμφάνιση return statement (κλήση της *return\_stat*) μέσα στο βασικό πρόγραμμα τυπώνεται κατάλληλο μήνυμα λάθους και ο compiler τερματίζει.
3. Η global λίστα **loopCounterList** διατηρεί το πλήθος των εμφανίσεων των loop statements. Κάθε φορά που ξεκινά ένα καινούριο loop statement προστίθεται ένα νέο στοιχείο στο τέλος της λίστας (κατά σύμβαση η πρώτη τετράδα του statement) και μόλις αυτό ολοκληρωθεί (μετά το endloop) αφαιρείται από τη λίστα το τελευταίο στοιχείο της. Όσο η λίστα αυτή δεν είναι κενή, υπάρχουν ενεργά loop statements, επομένως είναι επιτρεπτή η εμφάνιση exit statements. Σε κάθε κλήση της συνάρτησης *exit\_stat(),* εάν η λίστα είναι κενή, τότε το exit statement έχει τοποθετηθεί εκτός loop-endloop, επομένως τυπώνει κατάλληλο μήνυμα λάθους και τερματίζει.
4. Στη συνάρτηση **declarations**() γίνεται έλεγχος για τη δήλωση περισσότερων από μία φορές της ίδιας μεταβλητής μέσα στο πιο μεγάλο βάθος φωλιάσματος. Εάν υπάρχουν τουλάχιστον δύο μεταβλητές με το ίδιο όνομα, τυπώνεται κατάλληλο μήνυμα λάθους και ο compiler τερματίζει. Στη συνάρτηση **subprogram(),** κάθε φορά που προστίθεται ένας νέος ορισμός συνάρτησης γίνεται έλεγχος για την ύπαρξη οντότητας (μεταβλητής, συνάρτησης ή παραμέτρου) με το ίδιο όνομα και εάν βρεθεί τέτοια οντότητα, τυπώνεται κατάλληλο μήνυμα λάθους και η μετάφραση διακόπτεται.
5. Στη συνάρτηση **factor**() και στην περίπτωση του κανόνα id <idtail>, γίνεται έλεγχος για τον τύπο της οντότητας που έχει δοθεί. Εάν το id αντιστοιχεί σε όνομα συνάρτησης αλλά δεν δίνεται λίστα παραμέτρων ή αντίστροφα, εάν το id αντιστοιχεί σε όνομα μεταβλητής/παραμέτρου και δίνεται και λίστα παραμέτρων, τυπώνεται κατάλληλο μήνυμα λάθους και το πρόγραμμα τερματίζει.
6. Σε κάθε κλήση συνάρτησης, η συνάρτηση **factor**() βρίσκει τον ορισμό της συνάρτησης αυτής στο scope που έχει δηλωθεί και καλεί την συνάρτηση *idtail* περνώντας ως όρισμα τη λίστα των ορισμάτων που δέχεται αυτή. Η συνάρτηση **actualparlist**() επιστρέφει μια λίστα που περιέχει τους τύπους των παραμέτρων που δίνονται ως όρισμα σε κάποια συνάρτηση κατά την κλήση της. Η σύγκριση αυτών των δύο λιστών γίνεται στη συνάρτηση *actualpars*, όπου γίνεται σύγκριση των μεγεθών των δύο λιστών καθώς και της διάταξης των στοιχείων τους. Εάν το πλήθος διαφέρει ή έχουν δοθεί παράμετροι με διαφορετική σειρά τότε τυπώνεται κατάλληλο μήνυμα λάθους και το πρόγραμμα τερματίζει.
7. **Τελικός Κώδικας**