Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

ΜΥΥ802: Μεταφραστές

Διδάσκων: Γ. Μανής

Ακαδημαϊκό έτος: 2018-2019

«Υλοποίηση μεταγλωττιστή της γλώσσας Starlet»

Καρέτση Φωτεινή Α.Μ.: 2990

Μπελλά Ελευθερία Α.Μ.: 3039

Πίνακας περιεχομένων

1. Λεκτικός αναλυτής

Ο λεκτικός αναλυτής της γλώσσας Starlet είναι μια συνάρτηση που διαβάζει γραμμα-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, το tokenID μπορεί να πάρει μία από τις ακόλουθες τιμές:

- Δεσμευμένες λέξεις, που 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 *(«+», «-», «*», «/»)*, τους τελεστές συσχέτισης *(«<», «>», «=», «<=», «>=», «<>»)*, το σύμβολο ανάθεσης *(«:=»)*, τους διαχωριστές *(«;», «,», «:»)* και 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, όπως 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 στις 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

Στις 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 (όπως αυτοί έχουν οριστεί στη λίστα whiteletters) 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 στις καταστάσεις 8,9 και 10 αντίστοιχα, ενώ αν συναντήσει κάποιον από τους 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. Επίσης, έχει χρησιμοποιηθεί ένας μετρητής που μετρά το πλήθος των χαρακτήρων του αλφαριθμητικού και επιστρέφει τους 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) ο 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 τους χαρακτήρες «/» ή «*»). Σε αυτές τις 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

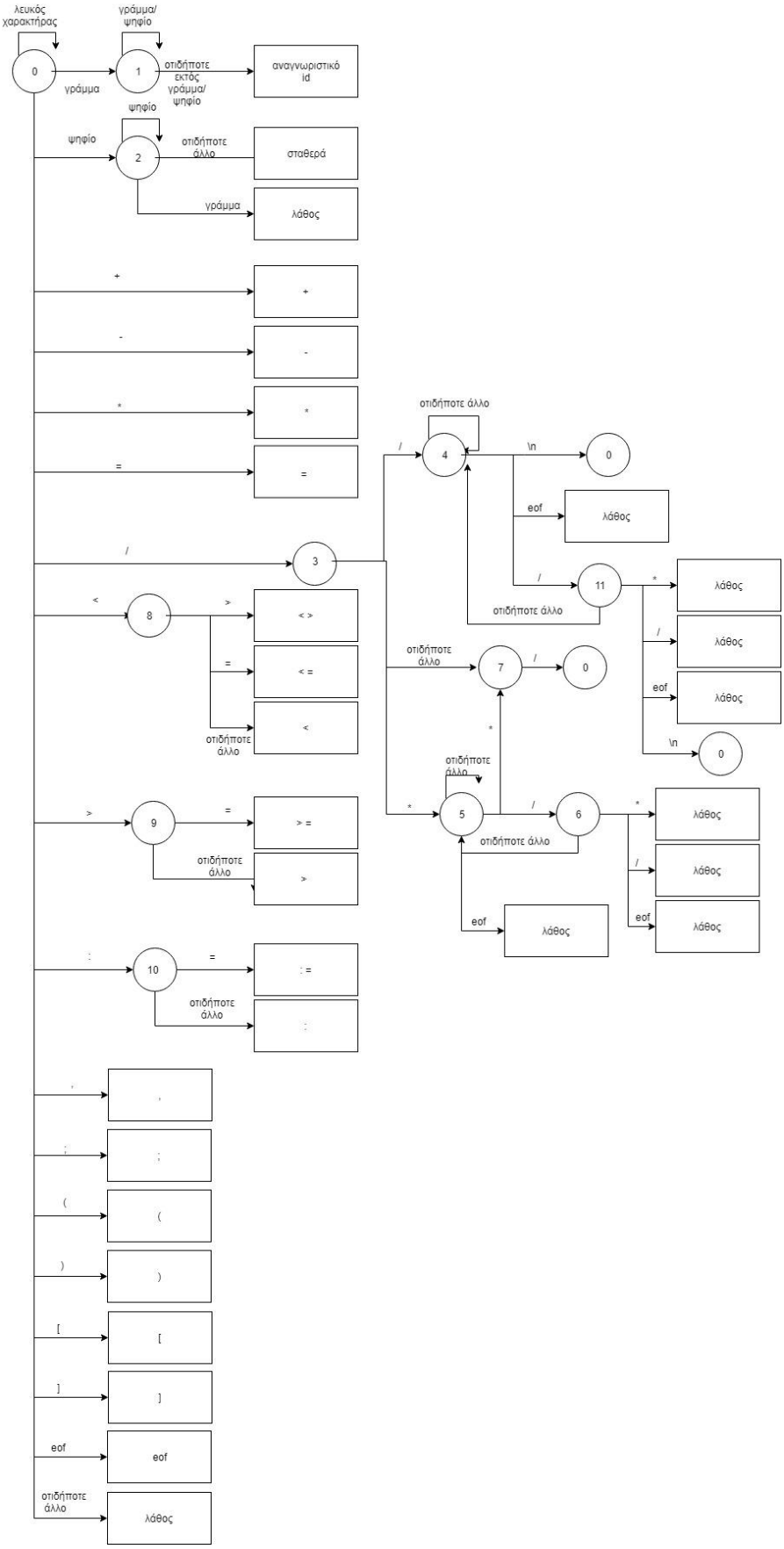
Από **την κατάσταση 9** θα αναγνωριστούν τα εξής σύμβολα:

- «>=», 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, στις καταστάσεις 1,2,3,8,9 και 10 αναιρούμε την ανάγνωση του τελευταίου χαρακτήρα που έχει διαβαστεί από τον λεκτικό αναλυτή, στις περιπτώσεις που αυτός δεν εμπίπτει σε κάποια από τις προκαθορισμένες κατηγορίες που έχουμε εξετάσει παραπάνω. Εάν δεν ληφθεί υπόψη η παραπάνω ιδιάζουσα 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.

2. 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 κατάλληλο μήνυμα λάθους. Κάθε λεκτική μονάδα που χρησιμοποιείται στο συντακτικό αναλυτή, προκύπτει από την κλήση του lex().

Η συντακτική ανάλυση ξεκινά με την κλήση της πρώτης συνάρτησης *program()* από τη main. Κατά κανόνα, η κλήση του λεκτικού αναλυτή γίνεται κάθε φορά που χρειάζεται να «καταναλωθεί» κάποιο token και να ανιχνευθεί το επόμενο. Ωστόσο, στην program() καλείται αρχικά ο lex() για να «γεμίσει» περιεχόμενο για πρώτη φορά το token και να συνεχιστεί ορθά η λειτουργία του μεταγλωττιστή.

Για κάθε κανόνα της γραμματικής ακολουθούνται τα εξής επιμέρους βήματα:

- Το όνομα κάθε υποπρογράμματος προκύπτει από το **αριστερό μέρος** του αντίστοιχου κανόνα της γραμματικής.
- Για κάθε tokenID που επιστρέφει ο lex(), γίνεται σύγκριση με τη δεσμευμένη λέξη ή χαρακτήρα που αναμένει ο κανόνας. Σε περίπτωση σφάλματος, τυπώνεται το κατάλληλο μήνυμα και η λειτουργία του μεταγλωττιστή τερματίζεται.
- **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 τις 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 και σε περίπτωση αποτυχίας ταυτοποίησης, εμφανίζεται κατάλληλο μήνυμα λάθους. Για παράδειγμα, στη συνάρτηση statement ελέγχουμε την τιμή του tokenID με όλες τις επιτρεπτές τιμές που μπορούν να ακολουθούν.
- Στους κανόνες που επιτρέπεται η εμφάνιση του κενού χαρακτήρα *ε*, δεν εμφανίζεται κάποιο μήνυμα λάθους.

<u>*Υποσημείωση*</u>: 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.

3. Ενδιάμεσος Κώδικας

Για την παραγωγή του ενδιάμεσου κώδικα, έχουν χρησιμοποιηθεί οι βοηθητικές συναρτήσεις και μεταβλητές που ακολουθούν.

3.1. Μεταβλητές

- **quad_program_list:** Δομή λεξικού, εδώ διατηρούμε τις παραγόμενες τετράδες του προγράμματος.
- **total_quads :** Διατηρεί το συνολικό πλήθος των τετράδων, λειτουργεί σαν μετρητής.
- **temp_value:** Βοηθητική μεταβλητή (λειτουργεί επίσης σαν μετρητής), διατηρεί το πλήθος των προσωρινών μεταβλητών (*T_1, T_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 κρατά ένα δείκτη σε κάθε exit statement. Κάθε φορά που ένα loop statement τερματίζει, το τελευταίο στοιχείο της λίστας απομακρύνεται και η τετράδα αυτή συμπληρώνεται ώστε να γίνεται άλμα στην επόμενη τετράδα.
- **checkIfExit:** Λίστα που διατηρεί ένα πλήθος από προσωρινές μεταβλητές που προστίθενται σε κάθε εμφάνιση loop statement. Αυτές αρχικοποιούνται με την τιμή μηδέν. Εάν βρεθεί exit statement, η προσωρινή μεταβλητή που αντιστοιχεί στο πιο εμφωλευμένο loop statement τίθεται στη μονάδα και γίνεται έξοδος από το loop. Διαφορετικά, εάν η τιμή της δεν αλλάξει, εκτελείται εκ νέου το loop statement.

3.2. Συναρτήσεις

- **next_quad():** Επιστρέφει την ετικέτα της επομένης τετράδας που θα παραχθεί.
- **gen_quad():** Δημιουργεί την επόμενη τετράδα του προγράμματος.
- **newTemp():** Δημιουργεί μια προσωρινή μεταβλητή (της μορφής T_1, T_2, ...).
- **emptylist():** Δημιουργεί και επιστρέφει μία κενή λίστα.
- **makelist():** Δημιουργεί μία καινούρια λίστα που περιέχει μόνο μία ετικέτα, αυτή που δίνεται ως όρισμα στη συνάρτηση.
- **merge():** Συνενώνει τις δύο λίστες 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. Συγκεκριμένα, το πρώτο όρισμα 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 τις τετράδες του προγράμματος μία προς μία, αν η τρέχουσα ετικέτα τετράδας έχει σημειωθεί στη λίστα-όρισμα της συνάρτησης, τότε συμπληρώνουμε το κενό στοιχείο της τετράδας με την ετικέτα-όρισμα της συνάρτησης.

3.3. Λειτουργία ενδιάμεσου κώδικα

Για να εκτελεστεί ορθά ο μεταγλωττιστής, τροποποιήσαμε τις 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(), δημιουργούνται οι τετράδες begin_block και end_block, οριοθετώντας, έτσι, την αρχή και το τέλος του block αντίστοιχα. Επιπλέον, εάν 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 δημιουργείται η τετράδα που δείχνει την ανάθεση αυτή.
- **if_stat():** Η συνθήκη του statement επιστρέφει δύο λίστες που μπορούμε πλέον να συμπληρώσουμε μέσω της χρήσης της backpatch. Η condTrue θα οδηγήσει την εκτέλεση στην πρώτη τετράδα των if-statements. Μετά την εκτέλεσή τους πρέπει να βγούμε εκτός if statement και να μην εκτελεστούν οι εντολές του elsepart. Γι' αυτό το λόγο, προστίθεται μια τετράδα jump που θα συμπληρωθεί με την ετικέτα της πρώτης τετράδας εκτός if-endif μόλις αυτή θα είναι διαθέσιμη (δηλαδή αμέσως μετά την μετάφραση του elsepart). Στην περίπτωση που η συνθήκη δεν αληθεύει, η λίστα condFalse μέσω της backpatch οδηγεί το πρόγραμμα στην εκτέλεση του elsepart.
- **while_stat():** Η while_quad διατηρεί την ετικέτα της τετράδας στην οποία πραγματοποιείται ο έλεγχος της συνθήκης. Εάν η συνθήκη αληθεύει, η λίστα condTrue (μέσω backpatch) οδηγεί στην εκτέλεση της συνάρτησης statements και κατόπιν πρέπει να επανελεγχθεί η αρχική συνθήκη. Διαφορετικά ο έλεγχος μεταβαίνει εκτός while block (με κατάλληλο backpatch στην condFalse).
- **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 της συνθήκης, η backpatch οδηγεί την εκτέλεση του προγράμματος στην ετικέτα do_while_quad αν η συνθήκη είναι λανθασμένη ή συνεχίζει στην επόμενη τετράδα εκτός block αν αληθεύει.
- **loop_stat():** Η 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, γίνεται η ανακατεύθυνση της ροής εκτέλεσης. Αν έχει βρεθεί κάποιο exit statement, οδηγούμαστε εκτός loop-endloop block (συμπληρώνοντας κατάλληλα την τετράδα που έχει κρατηθεί από τη συνάρτηση exit_stat() στη λίστα exitloop), διαγράφοντας παράλληλα και τις αντίστοιχες καταχωρήσεις στις βοηθητικές λίστες. Ειδάλλως, επιστρέφουμε στην αρχική τετράδα του loop statement.
- **exit_stat():** Αν εντοπιστεί exit statement που περιέχεται σε κάποιο loop statement, ενημερώνεται με την τιμή «1» η προσωρινή βοηθητική μεταβλητή και δημιουργείται η τετράδα jump που οδηγεί σε έξοδο από το loop. Αυτή η τετράδα θα συμπληρωθεί με την ετικέτα της επόμενης εκτός loop-endloop, μόλις η συντακτική ανάλυση εντοπίσει το δεσμευμένο αναγνωριστικό "endloop".

- **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. Για να προκύψει αυτή η εκτέλεση, οι λίστες που προέρχονται από την condition() συμπληρώνονται με ανάλογο τρόπο με αυτόν της συνάρτησης if_stat() με μόνη διαφορά ότι η condFalse οδηγεί στην επόμενη διαθέσιμη συνθήκη ή στο default statement. Εάν εκτελεστεί το default statement, επιστρέφουμε στην τετράδα με ετικέτα forcase_quad και επανεξετάζουμε τις συνθήκες, αλλιώς συμπληρώνεται κατάλληλα η jump τετράδα που έχει προκύψει από προηγούμενο statement με την ετικέτα της επόμενης τετράδας εκτός block.
- **incase_stat():** Σε αυτή τη συνάρτηση δημιουργείται μία προσωρινή μεταβλητή t που αρχικοποιείται με την τιμή 0. Έπειτα, μετά από κάθε condition που αληθεύει αλλάζουμε την τιμή του t σε 1 και κάνουμε backpatch την cond_true ώστε να εκτελεστούν τα statements. Στην περίπτωση που το condition είναι ψευδές, κάνουμε backpatch την cond_false για να ελεγχθεί το επόμενο when. Στο τέλος, μετά τη δεσμευμένη λέξη endincase, γίνεται έλεγχος για την τιμή του t. Αν η τιμή είναι ίση με 1, σημαίνει ότι τουλάχιστον μία συνθήκη είναι αληθής και ο έλεγχος μεταβαίνει στην αρχή της incase, διαφορετικά γίνεται έξοδος από τη συνάρτηση.
- **return_stat():** Εδώ δημιουργείται τετράδα με την gen_quad(), που επιστρέφει την τιμή της έκφρασης που δίνεται από το expression (με τη δεσμευμένη λέξη “retv”).
- **print_stat():** Εδώ δημιουργείται τετράδα με την gen_quad(), που τυπώνει την τιμή της μεταβλητής έκφρασης που δίνεται(με τη δεσμευμένη λέξη “out”).
- **input_stat():** Εδώ δημιουργείται τετράδα με την gen_quad() με τη δεσμευμένη λέξη «inp» που ζητά από το χρήστη να δώσει τιμή στην επιθυμητή μεταβλητή.
- **actualparitem():** Δημιουργεί τετράδες που αφορούν τις παραμέτρους μιας συνάρτησης. Εάν δίνεται όνομα κάποιας μεταβλητής ως παράμετρος, δημιουργείται τετράδα που υποδεικνύει τον τύπο της παραμέτρου (“CV”, “REF” ή “CP” για in, inout και inandout αντίστοιχα) και το όνομα της μεταβλητής που προκύπτει από τον λεκτικό αναλυτή. Στην περίπτωση περάσματος με τιμή, αντί για το όνομα μπορεί να προστεθεί το αποτέλεσμα της έκφρασης αυτής ως παράμετρος.
- **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 (Btrue, Bfalse) οι οποίες περιέχουν τις τετράδες που είναι αληθείς και ψευδείς (η κάθε boolterm αντίστοιχα περιέχει δύο δικές της λίστες, τις 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 τις 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 τις δύο λανθασμένες λίστες και εξισώνουμε την τελική αληθή λίστα με αυτή του τελευταίου boolfactor.
- **boolfactor():** Εδώ και πάλι λαμβάνουμε τις λίστες που επιστρέφει η συνάρτηση condition. Έπειτα, η αληθής λίστα της condition εξισώνεται με την αντίστοιχη της boolfactor (το ανάλογο συμβαίνει με τις λανθασμένες λίστες). Στην περίπτωση που συναντήσουμε τη δεσμευμένη λέξη "not" συμβαίνει το αντίθετο (η αληθής λίστα της condition εξισώνεται με τη λανθασμένη της boolfactor κ.ο.κ.). Αν έχουμε αριθμητική παράσταση, λαμβάνουμε την τιμή των expressions σε δύο μεταβλητές (E1 και E2), συμπληρώνουμε την αληθή λίστα του boolfactor ώστε να δείχνει στην τετράδα που εκτελεί την πράξη ανάμεσα στις E1 και E2 και την ψευδή λίστα σε μια μη συμπληρωμένη τετράδα jump.
- **expression():** Εδώ δημιουργούνται οι τετράδες που αφορούν κάποια πράξη πρόσθεσης ή αφαίρεσης. Οι όροι για κάθε πράξη προέρχονται ως αποτέλεσμα από την κλήση της συνάρτησης term(). Ο αρχικός όρος προστίθεται με τον επόμενο και το αποτέλεσμα αποθηκεύεται σε μια προσωρινή μεταβλητή. Εάν υπάρχουν περισσότεροι από δύο όροι, πρέπει να ενημερωθεί κατάλληλα ο αρχικός όρος έτσι ώστε να δείχνει στο τρέχον άθροισμα κάθε φορά. Η μεταβλητή που φέρει το τελικό αποτέλεσμα επιστρέφεται σε κάθε κλήση της συνάρτησης expression().
- **term():** Με την ίδια λογική με τη συνάρτηση expression(), εδώ δημιουργούνται και επιστρέφονται οι παράγοντες που χρησιμοποιούνται για την αποτίμηση μιας έκφρασης.
- **factor():** Η μεταβλητή factor_result της συνάρτησης διατηρεί την τιμή της σταθεράς, της έκφρασης ή του id που αποτελεί όρο της συνάρτησης term και την επιστρέφει.

3.4. 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.

Επιπρόσθετα, 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.
- Σε περίπτωση κλήσης κάποιας άλλης συνάρτησης εκτός της main, το αρχείο C που δημιουργείται τυπώνει ένα κατάλληλο μήνυμα και τερματίζεται η διαδικασία καταγραφής του.

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.

4. 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): Αναζητά και επιστρέφει την οντότητα με όνομα name και το βάθος φωλιάσματός της, ξεκινώντας από το τρέχον βάθος φωλιάσματος. 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 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.

5. Σημασιολογική Ανάλυση

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

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

Συγκεκριμένα:

a) 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.
b) 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.
c) 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.
d) 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.

e) 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.

f) 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.

6. Τελικός Κώδικας

Για τον τελικό κώδικα έχουν προστεθεί οι μεταβλητές και συναρτήσεις που παρατίθενται παρακάτω.

6.1. Μεταβλητές

- **mainStartQuad:** Υποδεικνύει την αρχή του block της main (την πρώτη τετράδα της).
- **paramCounter:** Μετρητής που δείχνει τον αριθμό παραμέτρων που περνιούνται στις καλούμενες συναρτήσεις .
- **cpCallPos:** Λίστα στην οποία αποθηκεύεται η σειρά της inandout παραμέτρου της καλούμενης συνάρτησης (παράδειγμα: αν x(a,b,c,d) και d=inandout, τότε cpCallPos = [3]).
- **cpVarNames**: Λίστα στην οποία αποθηκεύεται το όνομα της αντίστοιχης inandout παραμέτρου (δηλαδή στο παραπάνω παράδειγμα θα αποθηκευόταν στη λίστα το όνομα της παραμέτρου d).

6.2. Συναρτήσεις

- **gnlvcode(v):** Αυτή η συνάρτηση αποθηκεύει τη διεύθυνσης μίας μη τοπικής μεταβλητής v στον καταχωρητή $t0. Αρχικά, καλώντας τη συνάρτηση **searchEntity(v)**, εντοπίζουμε το βάθος φωλιάσματος της μεταβλητής στον πίνακα συμβόλων. Μέσω του συνδέσμου προσπέλασης, φορτώνεται στον καταχωρητή $t0 η διεύθυνση της αρχής του εγγραφήματος δραστηριοποίησης του «πατέρα» (καλούσας συνάρτησης) και, επαναληπτικά, ανεβαίνοντας προς τα πάνω φορτώνεται στον $t0 η διεύθυνση του αντίστοιχου προγόνου μέχρι τον πρόγονο που περιέχει τη μεταβλητή στη στοίβα του. Τέλος, η διεύθυνση της μεταβλητής αποθηκεύεται στον καταχωρητή $t0 προσθέτοντάς του το offset της μεταβλητής(αρχικά δείχνει στην αρχή της τρέχουσας στοίβας).

- **loadvr(v, r):** Εδώ αποθηκεύεται η τιμή της μεταβλητής v στον προσωρινό καταχωρητή $tr. Διακρίνουμε περιπτώσεις ανάλογα με τον τύπο της v:
  1. Αν η v είναι σταθερά, γίνεται μία απλή φόρτωση της τιμής της στον καταχωρητή $tr με χρήση της εντολής *li*.
     (Σε αυτό το σημείο έχει γίνει ξεχωριστός έλεγχος στην περίπτωση που η v είναι αρνητικός αριθμός, ώστε να μην θεωρείται δηλωμένη οντότητα και να επηρεάζεται η ομαλή ροή του προγράμματος.)
  2. Αν η v είναι καθολική μεταβλητή, δηλαδή έχει βάθος φωλιάσματος 0 (ανήκει στο κύριο πρόγραμμα), τότε η τιμή που φορτώνεται στον καταχωρητή $tr διαβάζεται από την διεύθυνση που προκύπτει από τον καταχωρητή $s0, που δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της main, προσαυξημένο κατά το offset της μεταβλητής.
  3. Αν η v είναι τοπική μεταβλητή, τυπική παράμετρος που περνάει με τιμή ή με τιμή-αποτέλεσμα και βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή, εκτελείται παρόμοια εντολή με την προηγούμενη περίπτωση με τη

διαφορά ότι εδώ χρησιμοποιείται ο $sp αντί του $s0. Ο $sp δείχνει στην αρχή της στοίβας της τρέχουσας συνάρτησης.

4. Αν η v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, τότε πρώτα λαμβάνουμε τη διεύθυνση της μεταβλητής από τη στοίβα και μετά φορτώνουμε στον καταχωρητή $tr την τιμή που διαβάζουμε από αυτή.
5. Αν η v είναι τοπική μεταβλητή, τυπική παράμετρος που περνάει με τιμή ή με τιμή-αποτέλεσμα και έχει βάθος φωλιάσματος μικρότερο του τρέχοντος, πρώτα καλείται η **gnlvcode()** ώστε να φορτωθεί η διεύθυνση της μεταβλητής στον $t0 και στη συνέχεια διαβάζεται η τιμή της από αυτή και φορτώνεται στον $tr.
6. Αν η v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο του τρέχοντος, καλείται πρώτα η **gnlvcode(),** από τη διεύθυνση που περιέχει ο $t0 διαβάζουμε την πραγματική διεύθυνση της μεταβλητής και μετά φορτώνουμε την τιμή της στον $tr**.**

Σε διαφορετική περίπτωση τυπώνεται μήνυμα λάθους.

- **storerv(r, v):** Εδώ πραγματοποιείται μεταφορά δεδομένων από τον καταχωρητή $tr στη μεταβλητή v. Διακρίνουμε ανάλογες περιπτώσεις όπως και στη **loadvr()**:
    1. Αν η v είναι καθολική μεταβλητή, δηλαδή έχει βάθος φωλιάσματος 0 (ανήκει στο κύριο πρόγραμμα) τότε αποθηκεύεται η τιμή του καταχωρητή $tr στη θέση μνήμης που δείχνει η διεύθυνση που προκύπτει από τον καταχωρητή $s0 προσαυξημένο κατά το offset της μεταβλητής.
    2. Αν η v είναι τοπική μεταβλητή, τυπική παράμετρος που περνάει με τιμή ή με τιμή-αποτέλεσμα και έχει βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή, εκτελείται παρόμοια εντολή με την προηγούμενη περίπτωση με τη διαφορά ότι εδώ χρησιμοποιείται ο $sp αντί του $s0.
    3. Αν η v είναι τυπική παράμετρος που περνάει με αναφορά και έχει βάθος φωλιάσματος ίσο με το τρέχον, τότε πρώτα λαμβάνουμε την πραγματική διεύθυνση της μεταβλητής (που είναι αποθηκευμένη στη θέση $sp + offset στη στοίβα) και μετά αποθηκεύουμε την τιμή του καταχωρητή $tr σε αυτή.
    4. Αν η v είναι τοπική ή προσωρινή μεταβλητή ή τυπική παράμετρος που περνάει με τιμή ή με τιμή-αποτέλεσμα και βάθος φωλιάσματος μικρότερο του τρέχοντος, τότε πρώτα καλείται η **gnlvcode()** και μετά αποθηκεύεται το περιεχόμενο του $tr στη θέση μνήμης που δείχνει η διεύθυνση της v.
    5. Αν η v είναι τυπική παράμετρος που περνάει με αναφορά και έχει βάθος φωλιάσματος μικρότερο από το τρέχον, καλείται η **gnlvcode()**, στη συνέχεια φορτώνεται στον $t0 η πραγματική διεύθυνση της μεταβλητής και τελικά αποθηκεύεται το περιεχόμενο του $tr στη θέση αυτή.

    Σε διαφορετική περίπτωση τυπώνεται μήνυμα λάθους.

- **producemipsfile(quadlist, sq, funcname):** Η συγκεκριμένη συνάρτηση παράγει το .asm αρχείο με τον τελικό κώδικα. Παίρνει ως παραμέτρους τις:
    1. **quadlist**: Τετράδα του προγράμματος που ελέγχεται.

2. **sq**: Η ετικέτα της τετράδας.
3. **funcname:** Το όνομα της συνάρτησης που εκτελείται αυτή τη στιγμή.

Η συγκεκριμένη συνάρτηση καλείται αμέσως μετά την ολοκλήρωση της μετάφρασης ενός block κώδικα και πριν τη διαγραφή του από τον πίνακα συμβόλων στη συνάρτηση *block*() του συντακτικού αναλυτή.

Ανάλογα με το είδος της τετράδας παράγεται ο κατάλληλος κώδικας:

1. **begin_block διαφορετικό του κύριου προγράμματος**: Αποθηκεύεται ο $ra στην πρώτη θέση του εγγραφήματος δραστηριοποίησης της συνάρτησης, ώστε να επιστρέψει στη συνέχεια στο σωστό σημείο του κώδικα.
2. **begin_block για το κύριο πρόγραμμα**: παράγεται η ετικέτα Lmain , μετακινείται ο $sp ανάλογα με το framelength της main και καταχωρείται η τιμή του στον $s0 (για την ευκολότερη προσπέλαση των global μεταβλητών στην υπόλοιπη εκτέλεση).
3. **Ανάθεση**: Καλείται η **loadvr** (ώστε να αποθηκευτεί στον $t1 η τιμή της μεταβλητής που ανατίθεται) και η **storerv** (ώστε να αποθηκευτεί η τιμή του $t1 στη μεταβλητή-στόχο, δηλαδή το τέταρτο στοιχείο της τετράδας).
4. **Αριθμητικές πράξεις (+, -, *, /):** Καλείται δύο φορές η loadvr (για τη φόρτωση των τιμών των τελεστέων, μία για κάθε τελεστέο της πράξης), στη συνέχεια παράγεται η εντολή για την αντίστοιχη πράξη (add, sub, mul, div) και το αποτέλεσμά της αποθηκεύεται στην επιθυμητή μεταβλητή.
5. **Συγκρίσεις (<=, >=, =, <, >, <>)**: Ακολουθείται παρόμοια διαδικασία με την περίπτωση 5 με τη διαφορά, ωστόσο, ότι η πράξη που εκτελείται είναι σχεσιακή (μία εκ των ble, bge, beq, blt, bgt, bne) και δεν αποθηκεύεται κάπου το αποτέλεσμα αλλά γίνεται άλμα σε κάποια κατάλληλη ετικέτα.
6. **Άλμα**: Παράγεται η εντολή που υποδεικνύει το άλμα στην κατάλληλη ετικέτα του προγράμματος.
7. **Επιστροφή τιμής**: Καλείται η **loadvr** (ώστε να φορτωθεί στον $t1 η τιμή επιστροφής), αποθηκεύεται στον $t0 η διεύθυνση της τιμής επιστροφής (η $3^{η}$ θέση του εγγραφήματος δραστηριοποίησης της συνάρτησης) και το περιεχόμενο του $t1 καταγράφεται στη διεύθυνση που δείχνει ο $t0. Φορτώνεται η διεύθυνση επιστροφής συνάρτησης και γίνεται άλμα σε αυτή.
8. **Είσοδος τιμής (input)**: Γίνεται φόρτωση της καθορισμένης σταθεράς 5 στον $v0, καλείται η ρουτίνα **syscall** και μεταφέρεται η τιμή που έδωσε ο χρήστης από τον καταχωρητή $v0 στον $t0.
9. **Εκτύπωση τιμής**: Γίνεται φόρτωση της καθορισμένης σταθεράς 1 στον $v0, και στη συνέχεια αν θέλουμε να εκτυπώσουμε μία σταθερά/αριθμό φορτώνεται η τιμή στον καταχωρητή ορισμάτων $a0 (γίνεται ξανά ο έλεγχος για τους αρνητικούς αριθμούς). Διαφορετικά, αν εκτυπώνουμε κάποια μεταβλητή, φορτώνουμε την τιμή της στον καταχωρητή $t0 με κλήση της **loadvr()** και μεταφέρουμε την τιμή του καταχωρητή αυτού στον καταχωρητή ορισμάτων $a0.
10. **Παράμετροι**: Στην περίπτωση της πρώτης παραμέτρου, γίνεται αναζήτηση της οντότητας της συνάρτησης που θα εκτελεστεί για να υπολογιστεί το

σωστό framelength, που απαιτείται για την μετακίνηση του καταχωρητή $fp και κατόπιν του $sp. Πριν πραγματοποιηθεί ο έλεγχος για το είδος της παραμέτρου μετακινούμε τον καταχωρητή $fp ώστε να δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της επιθυμητής συνάρτησης.

- **Αν η παράμετρος περνάει με τιμή** φορτώνονται στον $t0 τα περιεχόμενα της μεταβλητής και αυτά αποθηκεύονται στην κατάλληλη θέση του εγγραφήματος δραστηριοποίησης (η πρώτη παράμετρος τοποθετείται στη θέση 12 της στοίβας και ύστερα όλες οι υπόλοιπες σε θέσεις που είναι πολλαπλάσια του 4-ανάλογα με το πλήθος των παραμέτρων που έχουν ήδη περαστεί).
- **Αν η παράμετρος περνάει με αναφορά** καλείται η συνάρτηση **refParameterActions** (αναλύεται αμέσως μετά).
- **Αν η παράμετρος περνάει με τιμή-αποτέλεσμα**, αποθηκεύεται στην **cpCallPos** η θέση της παραμέτρου αυτής αλλά και το όνομα αυτής στην **cpVarNames**. Αμέσως μετά, φορτώνονται στον $t0 τα περιεχόμενα της μεταβλητής με κλήση της loadvr() και αυτά αποθηκεύονται στην κατάλληλη θέση του εγγραφήματος δραστηριοποίησης.
- Τέλος, **αν πρόκειται για παράμετρο στην οποία αποθηκεύεται η τιμή επιστροφής της συνάρτησης** (πρόκειται για κάποια προσωρινή μεταβλητή), αναζητείται η οντότητα αυτή και αποθηκεύεται η διεύθυνσή της στην $3^{η}$ θέση του εγγραφήματος δραστηριοποίησης της καλούμενης συνάρτησης. Εκεί θα αποθηκευτεί η τιμή επιστροφής της συνάρτησης που θα κληθεί.

(Μετά από κάθε παράμετρο αυξάνεται ο μετρητής paramCounter, που προσδιορίζει τον αριθμό ορισμάτων της κάθε συνάρτησης.)

11. **Κλήση συνάρτησης**: Σε πρώτο στάδιο εντοπίζουμε την οντότητα και το βάθος φωλιάσματος της συνάρτησης που θα κληθεί με τη **searchEntity**, ώστε να καθοριστεί κατόπιν ο σύνδεσμος προσπέλασης της καλούμενης συνάρτησης. Επιπλέον, εντοπίζουμε και το βάθος φωλιάσματος της καλούσας συνάρτησης, καθώς και θέτουμε τον $fp να δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της καλούμενης συνάρτησης σε περίπτωση που δεν υπάρχουν ορίσματα στη συνάρτηση. Όσον αφορά το **σύνδεσμο προσπέλασης**, αν η καλούσα και η καλούμενη συνάρτηση έχουν το ίδιο βάθος φωλιάσματος, έχουν δηλωθεί στο ίδιο scope, άρα μοιράζονται κοινό πατέρα. Σε διαφορετική περίπτωση, αν το βάθος φωλιάσματος της καλούσας είναι μικρότερο, η καλούμενη συνάρτηση είναι «παιδί» της καλούσας. Στην πρώτη περίπτωση, ως σύνδεσμος προσπέλασης της καλούμενης συνάρτησης θα εκχωρηθεί ο σύνδεσμος προσπέλασης της καλούσας, ειδάλλως, αποθηκεύεται ο καταχωρητής $sp, που δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της καλούσας συνάρτησης. Μετά την εκτέλεση της εντολής jal (έχει προηγηθεί κατάλληλη μεταφορά του καταχωρητή $sp), πραγματοποιείται η επιστροφή τιμών για τις

inandout παραμέτρους. Συγκεκριμένα, για κάθε οντότητα που έχει κρατηθεί στην *cpVarNames*, διαβάζεται η τιμή της από τη στοίβα της καλούμενης συνάρτησης, μεταφέρεται ο $sp στην αρχή της στοίβας της καλούσας συνάρτησης και γίνεται αποθήκευση της νέας τιμής στην αρχική μεταβλητή που είχε περαστεί ως παράμετρος (με χρήση της συνάρτησης **storerv**). Ο $sp αλλάζει εκ νέου τιμή και επιστρέφει στην προηγούμενη θέση του, για κάθε παράμετρο που έχει κρατηθεί στις βοηθητικές λίστες, μέχρι την οριστική επαναφορά του στην αρχή της στοίβας της καλούσας συνάρτησης όταν ολοκληρωθεί η επιστροφή των τιμών των inandout μεταβλητών. Επιπλέον, η μεταβλητή **paramCounter** μηδενίζεται, ώστε να είναι έτοιμη για την επόμενη κλήση συνάρτησης.

Για τις τετράδες που συμβολίζουν το τέλος των συναρτήσεων και του κυρίως προγράμματος καθώς και την **έξοδο (halt)** από το τελευταίο, δεν πραγματοποιείται κάποια λειτουργία στον τελικό κώδικα assembly.

- **refParameterActions(varname)**: Σε αυτή τη συνάρτηση γίνεται έλεγχος για την οντότητα var με όνομα varname, προκειμένου να αποθηκευτεί το κατάλληλο περιεχόμενο στην σωστή θέση του εγγραφήματος δραστηριοποίησης της καλούμενης συνάρτησης. Οι περιπτώσεις που ελέγχονται είναι οι εξής:
    1. *Αν η var είναι τοπική μεταβλητή, τυπική παράμετρος που περνάει με τιμή ή με τιμή-αποτέλεσμα και έχει βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή*, φορτώνεται η διεύθυνση της μεταβλητής από τη στοίβα της καλούσας συνάρτησης και αυτή αποθηκεύεται στην κατάλληλη θέση μνήμης στη στοίβα της καλούμενης (το offset υπολογίζεται όμοια με την περίπτωση περάσματος με τιμή).
    2. *Αν η var είναι τυπική παράμετρος που περνάει με αναφορά και έχει βάθος φωλιάσματος ίσο με το τρέχον*, τότε πρώτα λαμβάνουμε την πραγματική διεύθυνση της μεταβλητής (που είναι αποθηκευμένη στη θέση $sp + offset στη στοίβα) και μετά αποθηκεύεται στην κατάλληλη θέση μνήμης στη στοίβα της καλούμενης.
    3. *Αν η var είναι τοπική ή προσωρινή μεταβλητή ή τυπική παράμετρος που περνάει με τιμή ή με τιμή-αποτέλεσμα και βάθος φωλιάσματος μικρότερο του τρέχοντος*, τότε πρώτα καλείται η **gnlvcode()** και, μετά, η διεύθυνση που επιστρέφει η *gnlvcode()* αποθηκεύεται στην κατάλληλη θέση μνήμης στη στοίβα της καλούμενης.
    4. *Αν η var είναι τυπική παράμετρος που περνάει με αναφορά και έχει βάθος φωλιάσματος μικρότερο από το τρέχον*, καλείται η **gnlvcode()**, λαμβάνεται η πραγματική διεύθυνση της μεταβλητής και τελικά αποθηκεύεται στην κατάλληλη θέση μνήμης στη στοίβα της καλούμενης.

Επιπλέον, τροποποίηση έχει πραγματοποιηθεί και στη συνάρτηση **block()**, στην οποία μηδενίζεται σε κάθε νέα συνάρτηση ο αριθμός παραμέτρων (μεταβλητή paramCounter), εντοπίζεται η πρώτη τετράδα του κυρίως προγράμματος και καλείται η **producemipsfile** για παραγωγή του τελικού κώδικα για κάθε τετράδα που ανήκει στο block της εκάστοτε συνάρτησης.

7. Παράδειγμα λειτουργίας

Παρακάτω παρατίθεται ένα παράδειγμα λειτουργίας του μεταγλωττιστή που υλοποιήθηκε. Συγκεκριμένα, παρατίθεται ο αρχικός κώδικας σε γλώσσα Starlet, ο ενδιάμεσος κώδικας που προκύπτει, ο αντίστοιχος πίνακας συμβόλων και το τελικό πρόγραμμα σε assembly. Το βοηθητικό αρχείο σε C επίσης δημιουργείται, αλλά δεν είναι χρήσιμο πλέον, οπότε παραλείπεται για λόγους απλότητας. Περισσότερα παραδείγματα έχουν παραδοθεί μαζί με την υλοποίηση του μεταγλωττιστή.

Κώδικας Starlet:

```
program test1
        declare a,b,c,d,e;
        declare x,y,z,w;

        function noarguments()
                declare arg1, arg2;
                arg1 := x;
                arg2 := arg1 + y;
                x := arg1 - (arg2/2);
                print x;
                print arg1;
                print arg2;
                return 0
        endfunction

        function hasnested(in theta, inout r,
inandout p)

                function alpha(in par1)
                        if(par1>2) then
                                print par1*2;
                                return par1;
                                par1 := par1 - 1
                        else
                                par1 := par1 + 1;
                                print par1
                        endif;
                        return par1
                endfunction

                a := 1;
                while (a<5)
                        a := a + 1;
                        b := alpha(in a);
                        print b
                endwhile;
                p := theta*r + p;
                print a;
                return p
        endfunction
```

Ενδιάμεσος κώδικας:

```
0 ['begin_block', 'noarguments', '_',
'_']
1 [':=', 'x', '_', 'arg1']
2 ['+', 'arg1', 'y', 'T_0']
3 [':=', 'T_0', '_', 'arg2']
4 ['/', 'arg2', '2', 'T_1']
5 ['-', 'arg1', 'T_1', 'T_2']
6 [':=', 'T_2', '_', 'x']
7 ['out', 'x', '_', '_']
8 ['out', 'arg1', '_', '_']
9 ['out', 'arg2', '_', '_']
10 ['retv', '0', '_', '_']
11 ['end_block', 'noarguments', '_',
'_']
12 ['begin_block', 'alpha', '_', '_']
13 ['>', 'par1', '2', '15']
14 ['jump', '_', '_', '21']
15 ['*', 'par1', '2', 'T_3']
16 ['out', 'T_3', '_', '_']
17 ['retv', 'par1', '_', '_']
18 ['-', 'par1', '1', 'T_4']
19 [':=', 'T_4', '_', 'par1']
20 ['jump', '_', '_', '24']
21 ['+', 'par1', '1', 'T_5']
22 [':=', 'T_5', '_', 'par1']
23 ['out', 'par1', '_', '_']
24 ['retv', 'par1', '_', '_']
25 ['end_block', 'alpha', '_', '_']
26 ['begin_block', 'hasnested', '_',
'_']
27 [':=', '1', '_', 'a']
28 ['<', 'a', '5', '30']
29 ['jump', '_', '_', '38']
30 ['+', 'a', '1', 'T_6']
31 [':=', 'T_6', '_', 'a']
32 ['par', 'a', 'CV', '_']
33 ['par', 'T_7', 'RET', '_']
34 ['call', 'alpha', '_', '_']
35 [':=', 'T_7', '_', 'b']
36 ['out', 'b', '_', '_']
37 ['jump', '_', '_', '28']
```

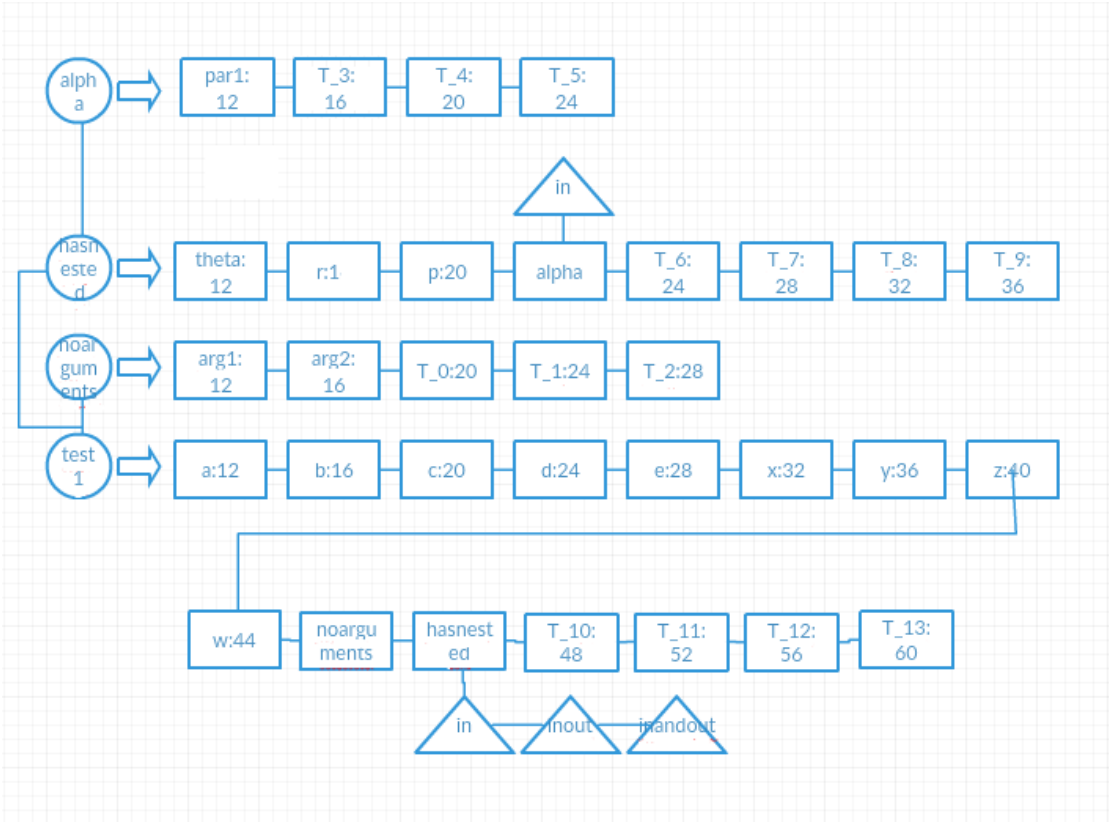
```
        b := 3;
        w := 2;
        while(b < 5)
                print b;
                a := noarguments();
                b := hasnested(in w, inout b,
inandout x);
                print b;
                print a
        endwhile;

        if(b < a+2) then
                print b+a
        else
                print a
        endif;

        print x
endprogram
```

```
38 ['*', 'theta', 'r', 'T_8']
39 ['+', 'T_8', 'p', 'T_9']
40 [':=', 'T_9', '_', 'p']
41 ['out', 'a', '_', '_']
42 ['retv', 'p', '_', '_']
43 ['end_block', 'hasnested', '_', '_']
44 ['begin_block', 'test1', '_', '_']
45 [':=', '3', '_', 'b']
46 [':=', '2', '_', 'w']
47 ['<', 'b', '5', '49']
48 ['jump', '_', '_', '62']
49 ['out', 'b', '_', '_']
50 ['par', 'T_10', 'RET', '_']
51 ['call', 'noarguments', '_', '_']
52 [':=', 'T_10', '_', 'a']
53 ['par', 'w', 'CV', '_']
54 ['par', 'b', 'REF', '_']
55 ['par', 'x', 'CP', '_']
56 ['par', 'T_11', 'RET', '_']
57 ['call', 'hasnested', '_', '_']
58 [':=', 'T_11', '_', 'b']
59 ['out', 'b', '_', '_']
60 ['out', 'a', '_', '_']
61 ['jump', '_', '_', '47']
62 ['+', 'a', '2', 'T_12']
63 ['<', 'b', 'T_12', '65']
64 ['jump', '_', '_', '68']
65 ['+', 'b', 'a', 'T_13']
66 ['out', 'T_13', '_', '_']
67 ['jump', '_', '_', '69']
68 ['out', 'a', '_', '_']
69 ['out', 'x', '_', '_']
70 ['halt', '_', '_', '_']
71 ['end_block', 'test1', '_', '_']
```

Πίνακας Συμβόλων:



Τελικός κώδικας:

```
   j  Lmain
L0:
   sw  $ra, ($sp)
L1:
   lw  $t1, -32($s0)
   sw  $t1, -12($sp)
L2:
   lw  $t1, -12($sp)
   lw  $t2, -36($s0)
   add  $t1, $t1, $t2
   sw  $t1, -20($sp)
L3:
   lw  $t1, -20($sp)
   sw  $t1, -16($sp)
L4:
   lw  $t1, -16($sp)
   li  $t2, 2
   div  $t1, $t1, $t2
   sw  $t1, -24($sp)
L5:
   lw  $t1, -12($sp)
   lw  $t2, -24($sp)
   sub  $t1, $t1, $t2
   sw  $t1, -28($sp)
L6:
   lw  $t1, -28($sp)
   sw  $t1, -32($s0)
L7:
   li  $v0, 1
   lw  $t0, -32($s0)
   move  $a0, $t0
   syscall
L8:
   li  $v0, 1
   lw  $t0, -12($sp)
   move  $a0, $t0
   syscall
L9:
   li  $v0, 1
   lw  $t0, -16($sp)
   move  $a0, $t0
   syscall
L10:
   li  $t1, 0
   lw  $t0, -8($sp)
   sw  $t1, ($t0)
   lw  $ra, ($sp)
   jr  $ra
L12:
   sw  $ra, ($sp)
L13:
   lw  $t1, -12($sp)
   li  $t2, 2
   bgt  $t1, $t2, L15
L14:
   j  L21
L15:
   lw  $t1, -12($sp)
   li  $t2, 2
   mul  $t1, $t1, $t2
   sw  $t1, -16($sp)
L16:
   li  $v0, 1
   lw  $t0, -16($sp)
   move  $a0, $t0
   syscall
L17:
   lw  $t1, -12($sp)
   lw  $t0, -8($sp)
   sw  $t1, ($t0)
   lw  $ra, ($sp)
   jr  $ra
L18:
   lw  $t1, -12($sp)
   li  $t2, 1
   sub  $t1, $t1, $t2
   sw  $t1, -20($sp)
L19:
   lw  $t1, -20($sp)
   sw  $t1, -12($sp)
L20:
   j  L24
L21:
   lw  $t1, -12($sp)
   li  $t2, 1
   add  $t1, $t1, $t2
   sw  $t1, -24($sp)
L22:
   lw  $t1, -24($sp)
   sw  $t1, -12($sp)
L23:
   li  $v0, 1
   lw  $t0, -12($sp)
   move  $a0, $t0
   syscall
L24:
   lw  $t1, -12($sp)
   lw  $t0, -8($sp)
   sw  $t1, ($t0)
   lw  $ra, ($sp)
   jr  $ra
```

```
L26:
  sw  $ra, ($sp)
L27:
  li  $t1, 1
  sw  $t1, -12($s0)
L28:
  lw  $t1, -12($s0)
  li  $t2, 5
  blt  $t1, $t2, L30
L29:
  j  L38
L30:
  lw  $t1, -12($s0)
  li  $t2, 1
  add  $t1, $t1, $t2
  sw  $t1, -24($sp)
L31:
  lw  $t1, -24($sp)
  sw  $t1, -12($s0)
L32:
  add  $fp, $sp, 28
  lw  $t0, -12($s0)
  sw  $t0, -12($fp)
L33:
  add  $t0, $sp, -28
  sw  $t0, -8($fp)
L34:
  sw  $sp, -4($fp)
  add  $sp, $sp, 28
  jal L12
  add  $sp, $sp, -28
L35:
  lw  $t1, -28($sp)
  sw  $t1, -16($s0)
L36:
  li  $v0, 1
  lw  $t0, -16($s0)
  move  $a0, $t0
  syscall
L37:
  j  L28
L38:
  lw  $t1, -12($sp)
  lw  $t0, -16($sp)
  lw  $t2, ($t0)
  mul  $t1, $t1, $t2
  sw  $t1, -32($sp)
L39:
  lw  $t1, -32($sp)
  lw  $t2, -20($sp)
  add  $t1, $t1, $t2
  sw  $t1, -36($sp)
L40:
  lw  $t1, -36($sp)
  sw  $t1, -20($sp)
L41:
  li  $v0, 1
  lw  $t0, -12($s0)
  move  $a0, $t0
  syscall
L42:
  lw  $t1, -20($sp)
  lw  $t0, -8($sp)
  sw  $t1, ($t0)
  lw  $ra, ($sp)
  jr  $ra
Lmain:

  add  $sp, $sp, 64
  move  $s0, $sp
L45:
  li  $t1, 3
  sw  $t1, -16($s0)
L46:
  li  $t1, 2
  sw  $t1, -44($s0)
L47:
  lw  $t1, -16($s0)
  li  $t2, 5
  blt  $t1, $t2, L49
L48:
  j  L62
L49:
  li  $v0, 1
  lw  $t0, -16($s0)
  move  $a0, $t0
  syscall
L50:
  add  $fp, $sp, 32
  add  $t0, $sp, -48
  sw  $t0, -8($fp)
L51:
  sw  $sp, -4($fp)
  add  $sp, $sp, 32
  jal L0
  add  $sp, $sp, -32
L52:
  lw  $t1, -48($s0)
  sw  $t1, -12($s0)
L53:
  add  $fp, $sp, 40
  lw  $t0, -44($s0)
```

```
  sw  $t0, -12($fp)
L54:
  add  $t0, $sp, -16
  sw  $t0, -16($fp)
L55:
  lw  $t0, -32($s0)
  sw  $t0, -20($fp)
L56:
  add  $t0, $sp, -52
  sw  $t0, -8($fp)
L57:
  sw  $sp, -4($fp)
  add  $sp, $sp, 40
  jal L26
  lw  $t0, -20($sp)
  add  $sp, $sp, -40
  sw  $t0, -32($s0)
  add  $sp, $sp, 40
  add  $sp, $sp, -40
L58:
  lw  $t1, -52($s0)
  sw  $t1, -16($s0)
L59:
  li  $v0, 1
  lw  $t0, -16($s0)
  move  $a0, $t0
  syscall
L60:
  li  $v0, 1
  lw  $t0, -12($s0)
  move  $a0, $t0
  syscall
L61:
  j  L47
L62:
  lw  $t1, -12($s0)
  li  $t2, 2
  add  $t1, $t1, $t2
  sw  $t1, -56($s0)
L63:
  lw  $t1, -16($s0)
  lw  $t2, -56($s0)
  blt  $t1, $t2, L65
L64:
  j  L68
L65:
  lw  $t1, -16($s0)
  lw  $t2, -12($s0)
  add  $t1, $t1, $t2
  sw  $t1, -60($s0)
L66:
  li  $v0, 1
  lw  $t0, -60($s0)
  move  $a0, $t0
  syscall
L67:
  j  L69
L68:
  li  $v0, 1
  lw  $t0, -12($s0)
  move  $a0, $t0
  syscall
L69:
  li  $v0, 1
  lw  $t0, -32($s0)
  move  $a0, $t0
  syscall
L70:
```