Εργασία μεταφραστών
Γλώσσα προγραμματισμού
Greek++

Ηλιάκης Ηλίας ΑΜ 5228
Χαλαϊδόπουλος Μιλτιάδης ΑΜ 5384

Εαρινό εξάμηνο 2024-2025

1.Εισαγωγή

Για τους σκοπούς της εργασίας καλούμαστε να φτιάξουμε έναν Μεταγλωττιστή μιας γλώσσας προγραμματισμού greek++ δηλαδή ένα πρόγραμμα το οποίο παίρνει σαν είσοδο ένα πρόγραμμα με μία δοθείσα γραμματική και παράγει κώδικα σε χαμηλότερο επίπεδο, σε assembly. Ταυτόχρονα εμφανίζει και μηνυματα λάθους αν κάτι πάει στραβά κατά την μεταγλώττιση.
Ο μεταφραστής χωρίζεται σε 5 διακριτά σημεία.

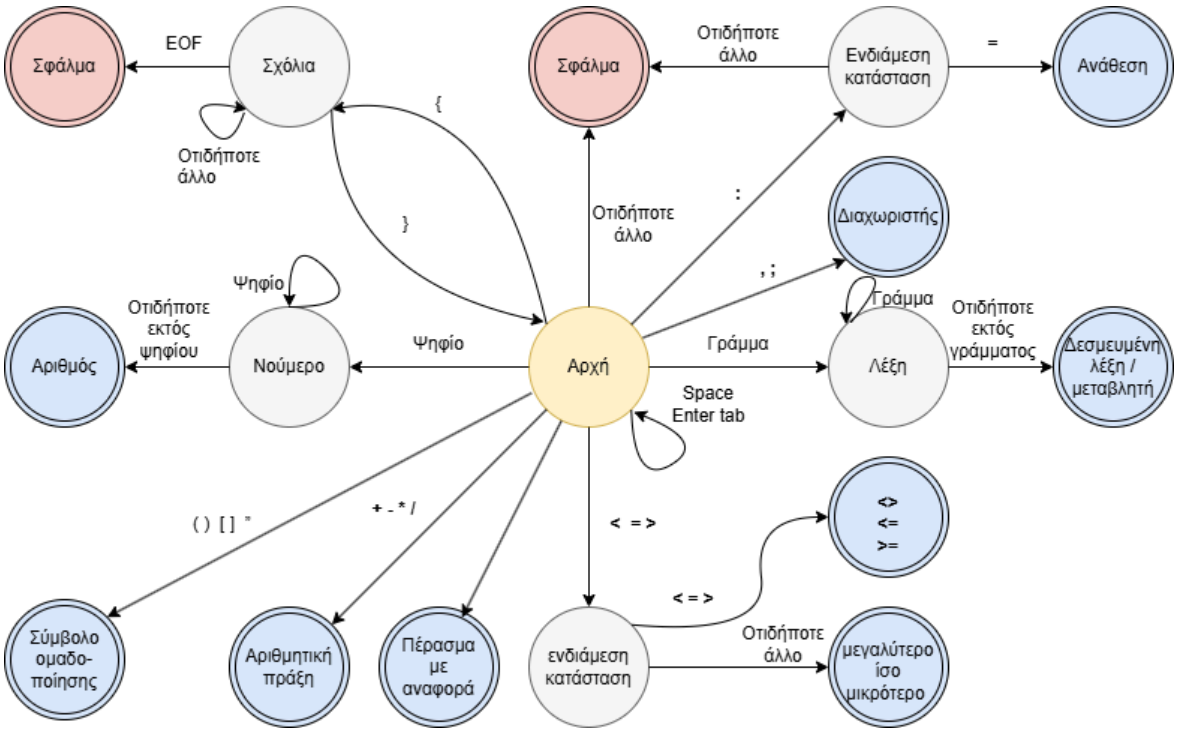
1. Λεκτικός αναλυτής : Ο διαχωρισμός του κώδικα σε στοιχειώδεις λεκτικές μονάδες
2. Συντακτικός αναλυτής : Δημιουργία του συντακτικού δέντρου. Καθώς και εύρεση συντακτικών λαθών
3. Ενδιάμεσος κώδικας : Παραγωγή τετράδων που συμβολίζουν πράξεις του προγράμματος
4. Πίνακας συμβόλων : Παρέχει πληροφορίες για τις συναρτήσεις, μεταβλητές και διαδικασίες του προγράμματος
5. Τελικός κώδικας : Μετατροπή του ενδιάμεσου κώδικα σε γλώσσα assembly

Στην συνέχεια αναφέρεται αναλυτικότερα η κάθε φάση του μεταφραστή

2.Λεκτικός αναλυτής

Πρώτο βήμα για την κατασκευή ενός μεταφραστή είναι η δημιουργία ενός λεκτικού αναλυτή. Ο Λεκτικός αναλυτής πρέπει από το αρχικό πρόγραμμα να μπορεί να αναγνωρίσει τις λεκτικές μονάδες βάση της γραμματικής της greek++ και να τις επιστρέφει. Επιπλέον θα είναι σε θέση να αναγνωρίσει και τα πρώτα πιθανά λάθη που δημιουργούνται όταν το αρχικό πρόγραμμα έχει λέξεις οι οποίες δεν μπορεί να ανήκουν στο λεξιλόγιο της greek++.

Για να αναγνωρίσουμε τις λέξεις πρέπει να κατασκευάσουμε ένα μη ντετερμινιστικό πεπερασμένο αυτόματο. Αυτό σχηματίζεται βάση της γραμματικής της γλώσσας.



Από το αυτόματο μπορούμε να καταλάβουμε όλες τις πιθανές λεκτικές μονάδες που μπορεί να αναγνωρίζει ο μεταφραστής καθώς και να κατανοήσουμε τα σφάλματα τα οποία πιθανόν να εμφανίζει. Ο τρόπος που έχουμε επιλέξει να διαχειριζόμαστε αυτή την φάση του μεταφραστή είναι μάλλον κάπως ανορθόδοξος καθώς σπάει το αρχικό πρόγραμμα σε στοιχειώδη κομμάτια και μετά τα συνενώνει, όμως δουλεύει και οι συναρτήσεις του εξηγούνται παρακάτω.

read_file(filename) : παίρνει σαν όρισμα το όνομα του αρχείου greek++ και επιστρέφει ένα string με το περιεχόμενο του αρχείου

tokenize(text) : παίρνει σαν όρισμα το περιεχόμενο ενός αρχείου greek++ και το σπάει σε λέξεις , λέξη θεωρείται μια σειρά από αριθμούς , γράμματα ή ένα σύμβολο. Τέλος επιστρέφει μία λίστα με τις λέξεις καθώς και σε ποια γραμμή βρίσκονται

merge_symbols(words_with_lines) : Παίρνει σαν όρισμα μία λίστα από λέξεις και συγχωνεύει τα σύμβολα τα οποία πρέπει να αποτελούν μία λεκτική μονάδα σύμφωνα με την γραμματική της greek++ (δηλαδή τα σύμβολα := , <= , >= , <>).

remove_comments(words_with_lines) : η συνάρτηση αυτή αφαιρεί τα σχόλια από μία λίστα από λέξεις.

check_illegal_chars(words_with_lines) : Ελέγχει αν μέσα στο πρόγραμμα υπάρχουν μη νόμιμοι χαρακτήρες. Αν βρει κάποιον πετάει ανάλογο exception.

lex(filename) : παίρνει σαν όρισμα το όνομα του αρχείου greek++ και καλεί τις προηγούμενες συναρτήσεις με την σειρά , στο τέλος επιστρέφει την παραγόμενη λίστα από λέξεις.

categorizelex(input_tokens) : Παίρνει σαν όρισμα την λίστα από λέξεις και τις κατηγοριοποιεί προσθέτοντας άλλο ένα γνώρισμα στις πλειάδες τους (πέρα από το περιεχόμενο των λέξεων και την γραμμή στην οποία βρίσκονται). Η κατηγοριοποίηση γίνεται ανάλογα με το αν ορίζουν αριθμό μεταβλητή , δεσμευμένη λέξη και τις υπόλοιπες τελικές καταστάσεις του πεπερασμένου αυτόματου.

giveNextToken() : Ίσως η σημαντικότερη όλων των συναρτήσεων του λεκτικού αναλυτή. Επιστρέφει κάθε φορά το επόμενο token από αυτό που επέστρεψε την προηγούμενη. Αυτό το κάνει χρησιμοποιώντας global πεδία που έχουν τον πίνακα με όλες τις λέξεις καθώς και την θέση στην οποία βρισκόμαστε.

Παρατηρούμε ότι στην τελευταία συνάντηση έγινε λόγος για tokens και όχι για λίστες από πλειάδες. Κρίναμε σκόπιμο να δημιουργήσουμε μια κλάση αντικειμένων Token η οποία θα περιέχει πληροφορίες για κάθε λεκτική μονάδα που παράγεται. Πιο συγκεκριμένα αυτή η πληροφορία θα περιλαμβάνει :

1. *Word* το περιεχόμενο αυτής της λήξης
2. *Type* η κατηγορία λέξεων στην οποία συνεπάγεται
3. *lineNum* η γραμμή στην οποία βρισκόταν στο αρχικό πρόγραμμα

Έχοντας κάνει όλα τα παραπάνω μπορούμε να αναγνωρίζουμε κάποια στοιχειώδη λεκτικά σφάλματα του greek++ κώδικα μας. Επιπλέον έχουμε μία λειτουργική συνάρτηση *giveNextToken* η οποία μπορεί κάθε φορά να μας δίνει το επόμενο token καθώς και πληροφορίες για αυτό. Η συνάρτηση αυτή θα φανεί ιδιαίτερα χρήσιμη για να παράγουμε το συντακτικό δέντρο στην επόμενη φάση , τον συντακτικό αναλυτή.

3.Συντακτικός αναλυτής

Έχοντας διαιρέσει τον κώδικα μας σε λεκτικές μονάδες μπορούμε να περάσουμε στην επόμενη φάση της μεταγλώττισης του που είναι η συντακτική ανάλυση. Κατά την συντακτική ανάλυση παράγουμε ένα συντακτικό δέντρο του δοθέντος greek++ κώδικα ενω ταυτόχρονα ελέγχουμε για συντακτικά λάθη σε αυτόν. Η συντακτική ανάλυση για τον μεταφραστή θα γίνει με την χρήση αναδρομικής κατάβασης με την γραμματική LL(1). Η γραμματική LL(1) αναγνωρίζει από αριστερά προς τα δεξιά τις λεκτικές μονάδες και φτιάχνει την αριστερότερη δυνατή παραγωγή, όταν βρίσκεται σε δίλημμα ποιον κανόνα να ακολουθήσει της αρκεί να κοιτάξει το αμέσως επόμενο σύμβολο στην συμβολοσειρά εισόδου. Σε περίπτωση που λάβει μια λεκτική μονάδα που δεν περιμένει ή δεν μπορεί να εντάξει στο συντακτικό δέντρο εμφανίζει κάποιο ανάλογο μήνυμα λάθους.

Για τα μηνύματα λάθους συγκεκριμένα θα χρειαστεί να εμφανίζουμε την εσφαλμένη λεκτική μονάδα την οποία διαβάζει ο μεταφραστής την γραμμή στην οποία βρίσκεται (αυτός είναι και ο λόγος ο οποίος την αποθηκεύουμε σαν πεδίο στην κλάση token) καθώς και τον λόγο για τον οποίο εμφανίζεται το συγκεκριμένο σφάλμα ή ένα μήνυμα με την αναμενόμενη επόμενη λεκτική μονάδα.

Ξέροντας όλα αυτά χρειαζόμαστε και την γραμματική της γλώσσας greek++ την οποία θα χρησιμοποιήσουμε για να παράγουμε το συντακτικό δέντρο μέσω των συναρτήσεων του συντακτικού αναλυτή. Για κάθε κανόνα της γραμματικής θα κατασκευάσουμε μία συνάρτηση η οποία θα ελέγχει αν ο κανόνας της γραμματικής τηρείτε και στην συνέχεια θα καλεί μια επόμενη συνάντηση έως ότου φτάσουμε στο τέλος τους προγράμματος.

Ο συντακτικός αναλυτής ξεκινάει καλώντας την συνάρτηση *parser()* θέτει το token να είναι η πρώτη λέξη του προγράμματος και καλεί την συνάρτηση *program()* η οποία είναι και η πρώτη συνάρτηση που πρέπει να έχει κάθε πρόγραμμα σύμφωνα με τους κανόνες της greek++.

Παρακάτω βλέπουμε τους κανόνες της γραμματικής της greek++ όπως δόθηκαν στην εκφώνηση της άσκησης :

Πανεπιστήμιο Ιωαννίνων
Τμήμα Πληροφορικής
"Μεταφραστές"
Διδάσκων: Γ. Μανής
Φεβρουάριος 2025

Γραμματική της greek++

```
program             :   'πρόγραμμα' ID programblock
                    ;

programblock        :   declarations subprograms
                            'αρχή_προγράμματος' sequence 'τέλος_προγράμματος'
                    ;

declarations        :   ( 'δήλωση' varlist )*
                    |
                    ;

varlist             :   ID ( ',' ID )*
                    ;

subprograms         :   ( func | proc )*
                    ;

func                :   'συνάρτηση' ID '(' formalparlist  ')' funcblock
                    ;

proc                :   'διαδικασία' ID '(' formalparlist  ')' procblock
                    ;

formalparlist       :   varlist
                    |
                    ;

funcblock           :   'διαπροσωπεία' funcinput funcoutput declarations
                            'αρχή_συνάρτησης' sequence 'τέλος_συνάρτησης'
                    ;

procblock           :   'διαπροσωπεία' funcinput funcoutput declarations
                        'αρχή_διαδικασίας' sequence 'τέλος_διαδικασίας'
                    ;

funcinput           :   'είσοδος' varlist
                    |
                    ;

funcoutput          :   'έξοδος' varlist
                    |
                    ;

sequence            :   statement ( ';' statement )*
                    ;
```

```
statement           :   assignment_stat
                        |   if_stat
                        |   while_stat
                        |   do_stat
                        |   for_stat
                        |   input_stat
                        |   print_stat
                        |   call_stat
                        ;

assignment_stat     :   ID ':=' expression
                    ;

if_stat             :   'εάν' condition 'τότε' sequence elsepart 'εάν_τέλος'
                    ;

elsepart            :   'αλλιώς' sequence
                    |
                    ;

while_stat          :   'όσο' condition 'επανάλαβε' sequence 'όσο_τέλος'
                    ;

do_stat             :   'επανάλαβε' sequence 'μέχρι' condition
                    ;

for_stat            :   'για' ID ':=' expression 'έως' expression  step
                          'επανάλαβε' sequence 'για_τέλος'
                    ;

step                :   'με_βήμα' expression
                    |
                    ;

print_stat          :   'γράψε' expression
                    ;

input_stat          :   'διάβασε' ID
                    ;

call_stat           :   'εκτέλεσε' ID idtail
                    ;

idtail              :   actualpars
                    |
                    ;
```

```
actualpars          :   '(' actualparlist ')'
                    ;

actualparlist       :   actualparitem ( ',' actualparitem )*
                    |
                    ;

actualparitem       :   expression |  '%'ID
                    ;

condition           :   boolterm ( 'ή' boolterm )*
                    ;

boolterm            :   boolfactor ( 'και' boolfactor )*
                    ;

boolfactor          :   'όχι' '[' condition ']'
                    |   '[' condition ']'
                    |   expression relational_oper expression
                    ;

expression          :   optional_sign term ( add_oper term )*
                    ;

term                :   factor   ( mul_oper factor )*
                    ;

factor              :   INTEGER
                    |   '(' expression ')'
                    |   ID idtail
                    ;

relational_oper     :   '=' | '<=' | '>=' | '<>' | '<'  | '>'
                    ;

add_oper            :   '+' | '-'
                    ;

mul_oper            :   '*' | '/'
                    ;

optional_sign       :   add_oper
                    |
                    ;
```

Οι συναρτήσεις του συντακτικού αναλυτή είναι σχετικά απλές. Η κάθε συνάρτηση έχει όνομα όμοιο με τον κανόνα τον οποίο ελέγχει. Παρακάτω θα δούμε μερικές χαρακτηριστικές συναρτήσεις του συντακτικού αναλυτή.

Σημαντικό : Το συντακτικό δέντρο το οποίο παράγεται χρησιμοποιείται έπειτα και στην παραγωγή ενδιάμεσου κώδικα αλλά και στον πίνακα συμβόλων , για αυτό οι συναρτήσεις περιέχουν κομάτια και από αυτές τις φάσεις της εργασίας. Στο παρόν μέρος θα αγνοήσουμε όλα αυτά καθώς θα εξηγηθούν στην συνέχεια της αναφοράς και θα επικεντρωθούμε καθαρά στα κομμάτια του συντακτικού αναλυτή

Η πρώτη συνάρτηση η οποία καλείται από κάθε πρόγραμμα greek++ είναι η συνάρτηση *program()* . Η συνάρτηση μένει πιστή στον αντίστοιχο κανόνα και ελέγχει αν η πρώτη λέξη η οποία πήρε είναι η δεσμευμένη λέξη “πρόγραμμα” (αν δεν είναι πετάει συντακτικό σφάλμα με μία αντίστοιχη αιτιολόγηση) έπειτα παίρνει το επόμενο λεξικογραφικά token με την κλήση της *token = giveNextToken()* και ελέγχει αν αυτό είναι ένα έγκυρο όνομα ορίσματος συνάρτησης (ξανά αν δεν είναι εμφανίζει το ανάλογο σφάλμα) αν είναι παίρνει το επόμενο token και καλεί την επόμενη συνάρτηση σύμφωνα με τους κανόνες δηλαδή την *programblock* με όρισμα το όνομα του προγράμματος.

Στην συνέχεια μεταβαίνουμε στην *programblock*. Κοιτώντας τους κανόνες της γραμματικής διαπιστώνουμε ότι αυτή η συνάρτηση ξεκινάει με τα *declarations* και έπειτα τα *subprograms* αυτός είναι και ο λόγος που εμφανίζονται κλείσεις αυτών των συναρτήσεων στις 2 πρώτες γραμμές της *programblock*. Έπειτα από αυτά πρέπει να δούμε την δεσμευμένη λέξη “αρχή_προγράμματος” και μετά ένα *sequence()*. Τέλος όταν το sequence ολοκληρωθεί αναμένεται η δεσμευμένη λέξη “τέλος_προγράμματος” παρατηρούμε πάλι κάθε φορά όταν εμφανίζεται στην συνάρτηση μας κάποια μη αναμενόμενη τιμή εμφανίζεται κάποιο αντίστοιχο μήνυμα λάθους.

Πρώτη συνάρτηση που κλήθηκε από την *programblock* ήταν η *declarations()* οπότε θα κοιτάξουμε αυτή στην συνέχεια. Η συνάρτηση αυτή κοιτάει αν κάθε λέξη ξεκινάει με την δεσμευμένη λέξη “δήλωση” αν ναι αναμένει μια σειρά μεταβλητών χωρισμένη από κόμματα (δηλαδή ένα *varlist*) αν πάλι δεν βρει την λέξη “δήλωση” σημαίνει ότι τα *declarations* έχουν τελειώσει και η συντακτική ανάλυση μπορεί να συνεχιστεί κοιτώντας για τα υποπρογράμματα μέσω της *subprograms()*.

Η *subprograms()* τώρα με την σειρά της όταν κληθεί διαχωρίζει τις κλήσεις ανάμεσα σε διαδικασίες και συναρτήσεις ανάλογα με ποια δεσμευμένη λέξη βλέπει . Προφανώς αν δεν δει καμία από τις 2 δεσμευμένες λέξεις σημαίνει ότι η δήλωση υποπρογραμμάτων έχει ολοκληρωθεί.

Ας δούμε τώρα πως ορίζεται μια συνάρτηση στην greek++ (με παρόμοιο τρόπο ορίζουμε και ένα process). Αρχικά καλείτε από την *subprograms()* η συνάρτηση *func()*. Αυτή η συνάρτηση ελέγχει αν το όνομα που δόθηκε στην συνάρτηση ήταν έγκυρο και αν ναι το αποθηκεύει σε έναν global πίνακα. Στην συνέχεια κοιτάει αν το *formalparlist()* βρίσκεται ανάμεσα από παρενθέσεις διότι έτσι ορίζεται από τους κανόνες της γραμματικής και τέλος καλεί την *funcblock(fname)* η οποία θα είναι η συνάρτηση που θα ελέγξει αν το σώμα της συνάρτησης που ορίστηκε είναι σωστό.

Η *funcblock* τώρα με την σειρά της το πετυχαίνει αυτό κοιτώντας αν η πρώτη λέξη στο σώμα της συνάρτησης είναι η δεσμευμένη λέξη “διαπροσωπία”. Μετά από αυτό, γίνεται έλεγχος να δούμε αν η συνάρτηση παίρνει τις κατάλληλες εισόδους και εξόδους μέσω των *funcinput* και *funcoutput*. Στην συνέχεια μια συνάρτηση μπορεί να έχει τις δικές της δηλώσεις έτσι καλείτε η declarations που είδαμε νωρίτερα και τελειώνει με τον εκτελέσιμο κώδικα ο οποίος περικλείεται ανάμεσα από τις λέξεις "αρχή_συνάρτησης" και

την “τέλος_συνάρτησης”. Ο εκτελέσιμος κώδικας είναι ένα *sequence()* δηλαδή μια σειρά από statements χωρισμένα με ελληνικό ερωτηματικό.Συγκεκριμένα με τον τρόπο με τον οποίο έχουμε υλοποιήσει την συγκεκριμένη συνάρτηση ελέγχει αν έχει statements. Αν σε κάποια φάση αντί για ερωτηματικό η συνάρτηση συναντησει τις διάφορες λέξεις “τέλος” σημαίνει ότι το συγκεκριμένο sequence έχει τελειώσει.

Ας δούμε λίγο πιο αναλυτικά τώρα την συνάρτηση *statements()* η οποία περιέχει τις βασικές κλήσεις της γλώσσας greek++. Υπάρχουν διάφορα είδη statements τα οποία έχουν το καθένα δική του ξεχωριστή σύνταξη. Ο πιο εύκολος τρόπος να αναγνωρίσουμε ένα statement είναι κοιτώντας την πρώτη του λέξη. Εάν η πρώτη λέξη είναι

1. Όρισμα τότε κάνουμε ανάθεση
2. “εάν” τότε έχουμε να κάνουμε με την αν του ελέγχου ροής
3. “για” ξεκινάει επαναλιπτική ροή for loop
4. “όσο” ξεκινάει επαναληπτική ροή while loop
5. “επανέλαβε” ξεκινάει επαναληπτική ροή do
6. “διάβασε” λαμβάνει input από το πληκτρολόγιο
7. “γράψε” εξάγει output στην οθόνη
8. “εκτέλεσε” κάνει κλήση κάποιοι υποπρογράμματος

η συνάρτηση *statements()* τώρα βλέποντας την πρώτη λέξη μπορεί να κρίνει ποιο από αυτά τα statements είναι αυτό που έχει δηλωθεί στον κώδικα και να ανακατευθήνει την ροή του προγράμματος ανάλογα (προφανώς αν η είσοδος που πάρει δεν είναι τίποτα από τα παραπάνω πρέπει να πάρουμε ένα μήνυμα λάθους).

Ας δούμε τώρα μερικά από τα statements της γλώσσας ξεκινώντας με την ανάθεση *assignment_stat(variable_name)* η οποία παίρνει σαν όρισμα το όνομα της μεταβλητής η οποία ανατίθεται ελέγχει αν αμέσως μετά το όνομα έχουμε το σύμβολο “:=” και στην συνέχεια καλεί την *expression()* όπως γίνεται και στην γραμματική της greek++

Μία άλλη συνάρτηση είναι η *if_stat()* σε αυτή την συνάρτηση ελέγχουμε αν έχουμε κάποιο condition και στην συνέχεια την δεσμευμένη λέξη “τότε” ακολουθούμενη από κάποιο *sequence*. Τέλος υπάρχει περίπτωση η if να έχει και else κομάτι αυτό δηλώνεται με την δεσμευμένη λέξη “αλλιώς” ακολουθούμενη από το *elsepart()*.

Εν κατακλείδι μετά από τα statements υπάρχουν μερικοί ακόμα κανόνες οι οποίοι έχουν να κάνουν κυρίως με λογικές πράξεις της γραμματικής. Οι λογικές πράξεις χρησιμοποιούνται κυρίως σαν συνθήκες στον έλεγχο ροής που κάνουν οι δηλώσεις που είδαμε παραπάνω. Ξεκινόντας από την λογική πράξη “ή” συμβολίζεται με την ομώνυμη λέξη και με αυτή ασχολείται η συνάρτηση condition() του συντακτικού αναλυτή. Αυτή η συνάρτηση ξεκινάει με ένα boolterm (για το οποίο θα μιλήσουμε παρακάτω) και αποθηκεύει τα σύνολα των True/False εκχωρήσεων. Όσο υπάρχει ακόμα η δεσμευμένη λέξη “ή” είτε θα συνενώσει τα True σύνολα είτε αν βρει κάτι false θα ενημερώσει το νέο.

Στην συνέχεια κάνουμε την λογική πράξη “και” στην συνάρτηση *boolterm()* η συνάρτηση αυτή δουλεύει παρόμοια με την *condition()* μόνο που αυτή την φορά συνενωνουμε όλα τα False σύνολα και αν βρούμε κάτι True ενημερώνεται με το νέο.

Τέλος υπάρχει και η συνάρτηση *boolfactor()* αυτή έχει 3 διαφορετικούς τρόπους σύνταξης οπότε πρώτη δουλειά του συντακτικού αναλυτή είναι να δει σε ποια περίπτωση βρισκόμαστε. Έπειτα βρίσκουμε τις συνθήκες και στο τέλος επιστρέφουμε τα σύνολα από True και False που βρήκαμε.

Φυσικά υπάρχουν και άλλες συναρτήσεις για τις οποίες δεν έχουμε κάνει λόγο ωστώσο η λογικής τους είναι η ίδια.Βλέπουμε τους κανόνες της γραμματικής και από αυτούς φτιάχνουμε τις συναρτήσεις.

4.Ενδιάμεσος κώδικας

Με το πέρας του συντακτικού αναλυτή έχουμε δημιουργήσει πλέον το κατάλληλο περιβάλλον έτσι ώστε να καλέσουμε μερικές σημαντικές ρουτίνες της συνάρτησης. Αυτές του ενδιάμεσου κώδικα. Με τον όρο ενδιάμεσος κώδικας αναφερόμαστε σε ένα σύνολο από τετράδες οι οποίες παράγονται από το συντακτικό δέντρο της γραμματικής και αποτελούνται από έναν τελεστή και τρία τελούμενα στοιχεία. Οι τετράδες αυτές είναι αριθμημένες με αύξοντα σειρά καθώς προχωράει το πρόγραμμα. Για αυτές τις τετράδες δημιουργήσαμε μία νέα κλάση αντικειμένων τα *Quad* η κλάση αυτή έχει τα πεδία

1. *ID* : Ο αριθμός ο οποίος χαρακτηρίζει την συγκεκριμένη τετράδα
2. *op* : Ο τελεστής που καθορίζει την πράξη του Quad
3. *x,y,z* : Τα τρία τελούμενα του Quad

Πέραν τον πεδίων της κλάσης υπάρχει και η συνάρτηση *toString(self)* η οποία επιστρέφει την τετράδα ως string μαζί με το χαρακτηριστικό ID της.

Με την κλάση Quad πλέον έτοιμη μπορούμε να ορίσουμε μερικές βοηθητικές συναρτήσεις οι οποίες θα χρησιμοποιηθούν για την παραγωγή ενδιάμεσου κώδικα.

- *emptylist()* : επιστρέφει μία κενή λίστα ετικετών τετράδων
- *makelist(x)* : επιστρέφει μία λίστα ετικετών τετράδων που περιέχει μόνο το x
- *merge(list1, list2)* : επιστρέφει μία λίστα από τη συνένωση των λιστών list1 και list2
- *nextquad()* : επιστρέφει τον αριθμό της τελευταίας τετράδας που παρήχθη
- *genquad(op, x, y, z)* : δημιουργεί την επόμενη τετράδα (op, x, y, z)
- *newtmp()* : δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή
- *backpatch(list,z)* : επισκέπτεται μία μία τις τετράδες της list και τις συμπληρώνει με την ετικέτα z

Έχοντας αυτές τις βοηθητικές συναρτήσεις μπορούμε τώρα να πάμε στο συντακτικό δέντρο του μεταφραστή και να δημιουργήσουμε τα κατάλληλα quads έτσι ώστε να παράγεται ο ενδιάμεσος κώδικας.

Ξεκινάμε από μια σχετικά απλή συνάρτηση όσο πρόκειται για τον ενδιάμεσο κώδικα την *programblock(prname)* εδώ πέρα πρέπει να δημιουργήσουμε πρώτα από όλα ένα quad που έχει ως πράξη την begin_block και x το όνομα του υποπρογράμματος το οποίο δημιουργούμε (στην συγκεκριμένη περίπτωση επειδή η συνάρτηση μας είναι η main το υποπρόγραμμα θα πάρει το όνομα του προγράμματος όπως δηλώνεται στην πρώτη γραμμή του αυτό όμως είναι διαφορετικό για τις διαδικασίες και για τις συναρτήσεις) έπειτα αφότου δούμε την λέξη τέλος_προγράματος σημαίνει ότι το block μας έχει τελειώσει οπότε προσθέτουμε και ένα ακόμα *genquad("end_block", prname, "_", "_")*. Τέλος μπορούμε να διαπιστώσουμε από την γραμματική της γλώσσας ότι το τέλος του *programblock* σηματοδοτεί και το γενικότερο τέλος του προγράμματος για αυτό θα προσθέσουμε και το quad με την πράξη halt. Όμοια πράττουμε και για τις συναρτήσεις και τις διαδικασίες προσθέτοντας και εκεί τα κατάλληλα genquad.

Ένα πολύ σημαντικό κομμάτι στον ενδιάμεσο κώδικα είναι η δημιουργία τετράδων για τα *statements*. Διαχειριζόμαστε κάθε statement ξεχωριστά καθώς πρέπει να παράξει quads με διαφορετικό τρόπο από τα υπόλοιπα. Ξεκινώντας από το πιο απλό assignment_stat εδώ πέρα το quad το οποίο παράγουμε προκύπτει από την πράξη ":=" με τα x και και z να παίρνουν την έκφραση δεξιά από την ανάθεση και το όνομα της μεταβλητής αντοίστοιχα.Το y παραμένει κενό.

Ένα άλλο statement είναι το if εδώ πέρα ξεκινάμε κάνοντας backpatch την λίστα των αληθών τιμών της συνθήκης της if και τους δίνουμε την ετικέτα την θέση στην οποία

βρισκόμαστε. Στην συνέχεια δημιουργούμε μια λίστα με την ετικέτα στην οποία βρισκόμαστε και φτιάχνουμε ένα jump quad . Ο λόγος που δεν δίνουμε ακόμα τιμή στο z του jump είναι διότι δεν γνωρίζουμε ακόμα σε ποια ετικέτα θα πρέπει να μεταφερθούμε μετά το άλμα. Έπειτα από αυτό μπορούμε να κάνουμε backpatch την λίστα των ψευδών από την αρχική μας συνθήκη στην νέα ετικέτα στην οποία βρισκόμαστε μετά το jump. Κάτω κάτω στην συνάρτηση κάνουμε μια τελευταία *backpatch(if_list, nextquad())* ώστε να διαπιστώσουμε που θα κάνουμε άλμα , αξίζει να αναφερθεί ότι αν η συνθήκη μας έχει else condition προσθέτουμε μια ακόμα λειτουργία. Όταν γίνεται ο έλεγχος για να αντιληφθούμε αν η συνθήκη condition αληθεύει, αν βγει ψευδείς αντί να προχωρήσουμε στο επόμενο quad όπως θα κάναμε κανονικά πρέπει να κάνουμε jump εκεί πέρα που βρίσκεται ο κώδικας για την else.

Άλλα δύο statements με τα οποία δεν έχουμε ασχοληθεί ακόμα στην αναφορά είναι τα *print_stat()* και *input_stat()*. Ξεκινώντας από το πρώτο θα παράξει ένα quad με πράξη το “out” και x το expression το οποίο θέλουμε να τυπωθεί. Από την άλλη το *input_stat()* παράγει και αυτό με την σειρά του ένα quad με πράξη αυτή την φορά το “inp” και ως x έχει την λέξη μετά από την δεσμευμένη λέξη “διάβασε” δηλαδή την μεταβλητή στην οποία θα γραφτεί πάνω η είσοδος.

Συνεχίζουμε με την συνάρτηση *factor()* η οποία σύμφωνα με την γραμματική περιέχει αριθμούς και ορισμένες εκφάσεις. Για να παράξουμε τα quads αυτής της συνάρτησης πηγαίνουμε στην περίπτωση όπου το token είναι τύπου definition (δηλαδή μεταβλητή ή συνάρτηση), τότε η συνάρτηση καλεί την *idtail()* για να διαπιστώσει αν είναι μεταβλητή ή κλήση αν είναι κλήση τώρα παράγει τις τετράδες με πράξη par για κάθε παράμετρο, επιπλέον δημιουργεί ret_temp προσωρινή για την επιστρεφόμενη τιμή. Τέλος παράγει call quad και επιστρέφει το ret_temp.

Συμπληρώνοντας όλα τα quads και δημιουργώντας όλες τις προσωρινές μεταβλητές μπορούμε πλέον να παράγουμε τον ενδιάμεσο κώδικα του προγράμματος που δίνεται στον μεταφραστή. Ο ενδιάμεσος κώδικας θα φανεί ιδιαίτερα χρήσιμος καθώς χρησιμοποιώντας αυτόν θα καταφέρουμε στο τέλος να δημιουργήσουμε τον τελικό κώδικα σε assembly. Δεν μπορούμε όμως ακόμα να φτάσουμε εκεί πέρα καθώς υπάρχει ανάμεσα ένα ακόμα βήμα, πρέπει να κρατήσουμε στοιχεία σχετικά με τα αναγνωριστικά (δηλαδή τις μεταβλητές , τις διαδικασίες και τις συναρτήσεις) του προγράμματος. Πιο συγκεκριμένα χρειάζονται πράγματα όπως η εμβέλεια των μεταβλητών , το εύρος τους καθώς και άλλα στοιχεία σαν το offset. Με αυτές τις λειτουργίες θα ασχοληθούμε στο επόμενο κομμάτι της αναφοράς καθώς στην συνεχεια θα κάνουμε λόγο για τον πίνακα συμβόλων του μεταφραστή.

5.Πίνακας συμβόλων

Ο πίνακας συμβόλων είναι απαραίτητο κομμάτι για κάθε μεταφραστή και διαχειρίζεται οτιδήποτε έχει να κάνει με τις μεταβλητές , τα πεδία , τις συναρτήσεις και τις διαδικασίες καθώς κρατάει σημαντικές πληροφορίες για αυτά που θα χρησιμοποιηθούν για να διαπιστωθούν σφάλματα αλλά και για να παραχθεί ο τελικός κώδικας του προγράμματος.

Για την υλοποίηση του πίνακα συμβόλων χρησιμοποιήσαμε μερικές νέες κλάσεις αντικειμένων. Τις *Scope , Entity και Argument* τα πεδία τους φαίνονται παρακάτω:

Entity : Μια μεταβλητή μια παράμετρος ένα πεδίο μία συνάρτηση ή μία διαδικασία

- *name* : Το όνομα της οντότητας
- *kind* : Τύπος οντότητας (για παράδειγμα συνάρτηση ή μεταβλητή)
- *offset* : Θέση της μεταβλητής στο Scope
- *parMode* : Τρόπος που περνιέται η παράμετρος (αν είναι με αναφορά ή με τιμή)

Scope :

- *nestinglevel* : Ο αριθμός εμφώλευσης 0 είναι για global. 1 για την 1 συνάρτηση 2 για την επόμενη κτλ
- *entities* : Λίστα με όλα τα σύμβολα του scope
- *offset_counter* : το offset κάθε μεταβλητής

επιπλέον η κλάση έχει και τις εξής συναρτήσεις

- *add_entity(self, entity)*: προσθέτει μια οντότητα στο scope αν είναι μεταβλητή τότε της δίνει και ανάλογο offset
- *get_entities()* : επιστρέφει την λίστα οντωτήτων
- *get_next_offset()* : επιστρέφει το επόμενο offset

Argument : Μια παράμετρος μίας συνάρτησης ή μιας διαδικασίας

- *parMode* : Τρόπος που περνιέται η παράμετρος (αν είναι με αναφορά ή με τιμή)
- *type* : τύπος της παραμέτρου
- *numofArgs* : αναφορά στην ομώνυμη global μεταβλητή αναφέρει τον αριθμό παραμέτρων που έχουν δηλωθεί
- *nextArg* : επόμενο argument

Επιπλέον έχουμε και μερικές βοηθητικές συναρτήσεις για την παραγωγή του πίνακα συμβόλων. Πιο συγκεκριμένα έχουμε τις :

- *lookupEntity(name)*: ψάχνει όλα τα scopes για το name και επιστρέφει την οντότητα στην οποία το βρήκε καθώς και το επίπεδο , αν δεν βρεθεί επιστρέφει None
- *currentLevel()* : επιστρέφει το επίπεδο φωλιάσματος του τρέχον scope
- *find_entity_and_scope(x, scopes)* : ψάχνει το x μέσα στα scopes

Τέλος θα χρειαστούμε και μερικές global μεταβλητές για τον πίνακα συμβόλων.

- *allScopes*: Λίστα με όλα τα scopes που έχουν δημιουργηθεί
- *scopeStack*: stack από Scopes
- *currentScope*: ορίζεται κάτω κάτω στον πίνακα συμβόλων, δηλώνει το τρέχον scope. Στην αρχή ξεκινάει με 0

Έχοντας όλα αυτά μπορούμε πλέον να επιστρέψουμε στο συντακτικό δέντρο και να κάνουμε τα απαραίτητα βήματα έτσι ώστε να παράγουμε τον πίνακα συμβόλων. Για να γίνει αυτό θα χρειαστεί να ασχοληθούμε μόνο με συναρτήσεις του συντακτικού αναλυτή οι οποίες ασχολούνται με μεταβλητές ή με συναρτήσεις και διαδικασίες, μερικές από αυτές είναι οι f*unc(),proc()* και *actualparitem()*.

Ας δούμε τώρα πιο αναλυτικά τι προσθέτουμε για τις συναρτήσεις (για τα processes είναι παρόμοια) Πρώτα από όταν καλείτε η *func()* πρέπει να δημιουργείται ένα καινούργιο Entity το οποίο θα είναι τύπου “func” έπειτα δημιουργούμε και ένα νέο Scope το οποίο θα έχει επίπεδο φωλιάσματος το τωρινό + 1 ας πούμε αν είμαστε ακόμα στην αρχή και έχουμε επίπεδο φωλιάσματος 0 το νέο θα είναι 1. Μετά από αυτό προσθέτουμε το νέο scope στο stack με τα scopes και θέτουμε την global μεταβλητή *currentScope* ίση με το scope το οποίο δημιουργήσαμε. Φυσικά όταν τελειώσει η συνάρτηση δεν ξεχνάμε ότι πρέπει να αφαιρέσουμε το scope από το scope stack κάνοντας pop.

Μεταβαίνουμε τώρα στην συνάρτηση *actualparitem()* όπου και θα δημιουργήσουμε νέα Entities αυτή την φορά όμως για μεταβλητές για αυτό και θα πάρουν τον τύπο “var” τα Entities αυτά θα προστίθενται στο scope στο οποίο βρισκόμαστε την δεδομένη στιγμή. Από το συντακτικό δέντρο γνωρίζουμε ότι η *actualparitem()* έχει δύο διαφορετικούς τρόπους σύνταξης ανάλογα με το αν το πέρασμα μιας μεταβλητής γίνεται με αναφορά ή αν γίνεται με τιμή. Αυτό βοηθάει στην παραγωγή πίνακα συμβόλων καθώς μπορούμε να φτιάξουμε διαφορετικό *Entity* όταν γίνεται πέρασμα με αναφορά και όταν γίνεται με τιμή. Έτσι στο πεδίο *parMode* βάζουμε “in” ή “inout” ανάλογα με το τι θέλουμε.

Με αυτά έχουμε ολοκληρώσει και τον πίνακα συμβόλων του μεταφραστή. Το μόνο που μένει να κάνουμε για να ολοκληρωθεί είναι να πάρουμε αυτόν και τον ενδιάμεσο κώδικα και να παράξουμε τον τελικό κώδικα ο οποίος θα είναι σε Risk-5 Assembly.

6.Τελικός κώδικας

Το τελευταίο κομμάτι κάθε μεταφραστή είναι η παραγωγή τελικού κώδικα , δηλαδή η ικανότητα του να πάρει τον πίνακα τετράδων ο οποίος δημιουργήθηκε από τον ενδιάμεσο κώδικα και να παράγει κώδικα σε μία γλώσσα χαμηλού επιπέδου όπως είναι η Assembly.

Όπως και σε κάθε άλλη φάση στην αρχή παραθέτουμε τις βοηθητικές συναρτήσεις οι οποίες χρησιμοποιήθηκαν για την παραγωγή του τελικού κώδικα :

gnlvcode(v) : η συνάρτηση αυτή βρίσκει σε ποιο scope δηλώθηκε η v και από τον πίνακα συμβόλων υπολογίζει πόσα επίπεδα πάνω βρίσκεται και φτιάχνει κατάλληλες εντολές ώστε να μπορέσει να την εντοπίσει.

loadvr(v,r) : η συνάρτηση αυτή φορτώνει την τιμή της μεταβλητής v στον καταχωρητή t{r} .Ο τρόπος με τον οποίο το κάνει εξαρτάται από τον τύπο της v . Αν είναι σταθερά φορτώνεται απευθείας με την ψευδοεντολή li . Απο την άλλη αν είναι τοπική φορτώνεται από τον s0 ή sp ανάλογα με το αν είναι τοπική στο level 0 ή στο τρέχον επίπεδο. Φυσικά αν δεν είναι τοπική θα πρέπει να τρέξουμε πρώτα την *gnlvcode(v)*

storevr(v,r) : Παρόμοια με την load μόνο που τώρα αποθηκεύει την τιμή της μεταβλητής v στον καταχωρητή t{r} η διαφορά εδώ είναι ότι κοιτάμε και αν η παράμετρος είναι περασμένη με τιμή ή με αναφορά . Προφανώς αν είναι με αναφορά δηλαδη το parMode είναι “inout” πρέπει να περάσουμε την διεύθυνση της όχι απλά την τιμή της.

Θα χρειαστούμε μια τελευταία κλάση για την παραγωγή τελικού κώδικα την *FinalCode_Transformer* η κλάση αυτή έχει τα εξής πεδία :

- *quads* : Λίστα από quads τα οποία θα μετατραπούν σε τελικός κώδικας
- *code* : Λίστα με ολόκληρο τον τελικό κώδικα
- *all_scopes* : Όλα τα scope του προγράμματος
- *main_program_name* : το όνομα του προγράμματος το οποίο μεταφράζεται
- *frame_lengths* : δομή δεδομένων η οποία αποθηκεύει το μήκως κάθε frame
- *current_nesting* : Το nesting level στο οποίο βρισκόμαστε. Στην αρχή 0

Η κλάση αυτή έχει δημιουργεί τον τελικό κώδικα από τα quads καλώντας την συνάρτηση *transform(self)* η οποία για κάθε quad καλεί την *transform_quad(self,quad)* . Η *transform_quad(self,quad)* ανάλογα με την πράξη την οποία κάνει το κάθε quad καλεί και μία διαφορετική συνάρτηση handle για παράδειγμα αν το quad το οποίο καλείται να διαχειριστεί έχει ως πράξη του την := καλεί την *handle_assign()* .

Πρώτα από όλα ας δούμε πως δουλεύουν 3 απλές hande συναρτήσεις :

- *handle_arithmetic(self, op, x, y, z)* : Η συνάρτηση αυτή παράγει τελικό κώδικα για αριθμητικές πράξεις. Πρώτα από όλα βρίσκει ποια πράξη κάνουμε και την αντιστοιχεί με την σωστή πράξη σε assembly, έπειτα φορτώνει τα x και y στους καταχωρητές t1 και t2 με την χρήση της *loadvr* και ζητάει από το αποτέλεσμα να αποθηκευτεί στην t3. Τέλος καλεί την *storevr* για να γράψει το αποτέλεσμα πίσω στην z.
- *handle_assign(self,x,z)* : Η συνάρτηση αυτή παράγει τελικό κώδικα για την πράξη ανάθεσης. Πρώτα από όλα φορτώνει το x στον t1 με την συνάρτηση *loadvr* και έπειτα με την κλήση της *storevr* αποθηκεύει το περιεχόμενο του t1 στο z.
- *handle_conditional_jump(self, op, x, y, z)* : Η συνάρτηση αυτή παράγει τελικό κώδικα για τις λογικές πράξεις. Για άλλη μια φορά φορτώνει τα x,y στους t1,t2 και όπως και η συνάρτηση για τις αριθμητικές πράξεις αντιστοιχεί την λογική πράξη του quad με την αντίστοιχη εντολή της assembly.

Υπάρχουν όμως και άλλες συναρτήσεις πιο περίπλοκες. Μία από αυτές είναι η *handle_begin_block(self, block_name)* της οποίας η λειτουργία αλλάζει ανάλογα με το αν βρισκόμαστε στο κύριο πρόγραμμα ή σε ένα υποπρόγραμμα.. Αν είμαστε στο κύριο πρόγραμμα το nesting level θα είναι 0. Η συνάρτηση θα υπολογίσει το *frame_length* από το *offset_counter* και τέλος θα αρχικοποιηθεί ο καταχωρητής s0 και το stack pointer. Αν από την άλλη βρισκόμαστε σε κάποιο υποπρόγραμμα η συνάρτηση εντοπίζει το scope στο οποίο βρίσκεται το block_name βρίσκει πάλι το *frame_length* και ενημερώνει το *current_nesting* .

Συνεχίζουμε με την συνάρτηση *handle_call(self, func_name)*. Η συνάρτηση στην 1η της γραμμή βρίσκει το scope όπου υπάρχει entity με όνομα *func_name* και τύπο func ή proc. Στην συνέχεια βρίσκει Nesting και Frame Length και έπειτα αν η συνάρτηση έχει ίδιο επίπεδο εμφώλευσης με το αντικείμενο που κάλεσε την handle φορτώνει και αποθηκεύει στον t0 κατάλληλα , αλλιώς αποθηκεύει στον stack pointer.

Η τελευταία συνάρτηση η οποία θα δούμε είναι η *handle_par(self, x, mode)* . Η συνάρτηση αυτή είναι υπεύθυνη για τη δημιουργία του κώδικα για την προώθηση παραμέτρων κατά την κλήση υποπρογράμματος. Παίρνει σαν όρισμα το όνομα της παραμέτρου x και αν προωθείτε με αναφορά ή με τιμή (mode). Το par_counter το οποίο ορίζουμε μετράει πόσοι παράμετροι έχουν ήδη προωθηθεί. Αν δεν έχει προωθηθεί καμία και αυτή είναι η πρώτη τότε η συνάρτηση εντοπίζει το επόμενο call quad, και βάσει αυτού προσδιορίζει το scope του καλούμενου υποπρογράμματος (next_call_scope) και το μήκος του frame του (frame_length). Στην συνέχεια αν η μεταβλητή έχει περάσει με τιμή την φορτώνει και την αποθηκεύει στο σωστό offset του νέου frame. Αν έχει περάσει με αναφορά από την άλλη αποθηκεύει την διεύθυνση αυτή την φορά της τιμής. Τέλος αν έχουμε επιστροφή τιμής, την επιστρέφουμε μέσω του pointer fp.

Με όλα αυτά τελειωμένα πλέον όταν καλούμε την transform από έναν FinalCode_Transformer μπορούμε από τον ενδιάμεσο κώδικα να παράξουμε τελικό. Στο επόμενο κεφάλαιο της αναφοράς κάνουμε μια σύντομη περιγραφή πως τρέχει το πρόγραμμα του μεταφραστή της greek++.

7.Ροή προγράμματος

Ο μεταφραστής παίρνει το όνομα του προγράμματος το οποίο θέλει να τρέξει σαν όρισμα. Το πρόγραμμα αυτό πρέπει να έχει την κατάληξη .gr για παράδειγμα για να τρέξουμε το example.gr θα γράψουμε στην κονσόλα *python3 greek_5228_5384.py example.gr* . Το πρόγραμμα τότε θα πάρει αυτό το όρισμα και θα ελέγξει αν έχει κατάληξη .gr και αν ναι θα καλέσει τον lexer έπειτα θα καλέσει τον parser και στην συνέχεια θα εμφανίσει στην οθόνη τον πίνακα συμβόλων , τον αφήσαμε στην οθόνη καθώς ίσως φανεί χρήσιμος στον έλεγχο του κώδικα. Επειτα δημιουργεί τον ενδιάμεσο κώδικα και τον αποθηκεύει σε ένα ομόνημο αρχείο αλλά με .int κατάληξη. Τέλος δημιουργούμε τον τελικό κώδικα ο οποίος αποθηκεύεται σε ένα αρχείο με .asm κατάληξη. Αν υπάρξει οποιοδήποτε σφάλμα νωρίτερα κατά την διάρκεια της μεταγλώττισης θα εμφανιστεί ένα ανάλογο μήνυμα λάθους στην κονσόλα πετώντας ένα exception. Στο τελευταίο κεφάλαιο θα δούμε ένα μικρό παράδειγμα της μετατροπής από το αρχικό πρόγραμμα σε τελικό κώδικα.

8. Έλεγχος κώδικα

Ας δούμε τώρα ένα απλό παράδειγμα το οποίο δείχνει πως γίνεται η μετάφραση της greek++ το πρόγραμμα που επιλέξαμε είναι αυτό που βρίσκεται το example.gr και έχει σύνταξη

```
πρόγραμμα Μπάμπης
δήλωση α,β,γ,δ

συνάρτηση μέσος_όρος(α,β,γ)
        διαπροσωπεία
        είσοδος α,β
        έξοδος γ
αρχή_συνάρτησης
        γ:=(α+β)/2
τέλος_συνάρτησης

αρχή_προγράμματος
        α:=1;
        β:=3;
        δ:=μέσος_όρος(α,β)
τέλος_προγράμματος
```

Ας δούμε τώρα πως τρέχει ξεκινόντας με τον λεκτικό αναλυτή προσθέτοντας μία *print(lexer_List)* Λίγο πριν αρχίσει η συντακτική ανάλυση στο τέλος του προγράματος μπορούμε να δούμε τις λεκτικές μονάδες οι οποίες παράγονται. Η λίστα των λεκτικών μονάδων είναι η :

[('πρόγραμμα', 1, 'keyword'), ('Μπάμπης', 1, 'definition'), ('δήλωση', 2, 'keyword'), ('α', 2, 'definition'), (',', 2, 'delimiter'), ('β', 2, 'definition'), (',', 2, 'delimiter'), ('γ', 2, 'definition'), (',', 2, 'delimiter'), ('δ', 2, 'definition'), ('συνάρτηση', 4, 'keyword'), ('μέσος_όρος', 4, 'definition'), ('(', 4, 'grouping sumbols'), ('α', 4, 'definition'), (',', 4, 'delimiter'), ('β', 4, 'definition'), (',', 4, 'delimiter'), ('γ', 4, 'definition'), (')', 4, 'grouping sumbols'), ('διαπροσωπεία', 5, 'keyword'), ('είσοδος', 6, 'keyword'), ('α', 6, 'definition'), (',', 6, 'delimiter'), ('β', 6, 'definition'), ('έξοδος', 7, 'keyword'), ('γ', 7, 'definition'), ('αρχή_συνάρτησης', 8, 'keyword'), ('γ', 9, 'definition'), (':=', 9, 'assignment'), ('(', 9, 'grouping sumbols'), ('α', 9, 'definition'), ('+', 9, 'add operation'), ('β', 9, 'definition'), (')', 9, 'grouping sumbols'), ('/', 9, 'multyply operation'), ('2', 9, 'number'), ('τέλος_συνάρτησης', 10, 'keyword'), ('αρχή_προγράμματος', 12, 'keyword'), ('α', 13, 'definition'), (':=', 13, 'assignment'), ('1', 13, 'number'), (';', 13, 'delimiter'), ('β', 14, 'definition'), (':=', 14, 'assignment'), ('3', 14, 'number'), (';', 14, 'delimiter'), ('δ', 15, 'definition'), (':=', 15, 'assignment'), ('μέσος_όρος', 15, 'definition'), ('(', 15, 'grouping sumbols'), ('α', 15, 'definition'), (',', 15, 'delimiter'), ('β', 15, 'definition'), (')', 15, 'grouping sumbols'), ('τέλος_προγράμματος', 16, 'keyword')]

Όπου σε κάθε πλειάδα το πρώτο όρισμα είναι η λέξη το 2ο η γραμμή στην οποία βρίσκεται και 3η η γενική κατηγορία λέξεων στην οποία βρίσκεται. Όπως μπορούμε να παρατηρήσουμε από την γραμματική της γλώσσας όλες οι λεκτικές μονάδες είναι σωστές.

Στην συνέχεια έχουμε τον συντακτικό αναλυτή , στην κονσόλα εμφανίζεται το μήνυμα “Η συντακτική ανάλυση ολοκληρώθηκε” που σημαίνει ότι δεν βρέθηκαν

συντακτικά λάθη, για άλλη μια φορά κοιτώντας το συντακτικό της greek++ διαπιστώνουμε ότι δεν υπάρχουν συντακτικά λάθη στο πρόγραμμα το οποίο δώσαμε οπότε ορθά δεν εμφανίζεται κάποιο μήνυμα λάθους (αν αλλάξουμε κάτι ώστε να είναι συντακτικά λάθος το example τότε αυτό αλλάζει)

Στην συνέχεια έχουμε την παραγωγή ενδιάμεσου κώδικα , σε αυτή την περίπτωση το πρόγραμμα παρήγαγε.

1: begin_block , μέσος_όρος , _ , _
2: + , α , β , T_1
3: / , T_1 , 2 , T_2
4: := , T_2 , _ , γ
5: end_block , μέσος_όρος , _ , _
6: begin_block , Μπάμπης , _ , _
7: := , 1 , _ , α
8: := , 3 , _ , β
9: par , α , cv , _
10: par , β , cv , _
11: par , T_3 , ret , _
12: call , μέσος_όρος , _ , _
13: := , T_3 , _ , δ
14: halt , _ , _ , _
15: end_block , Μπάμπης , _ , _

Ας δούμε λίγο τα quads που παρήχθησαν ξεκινόντας από αυτά που βρίσκονται μέσα στην συνάρτηση μέσος_όρος όπου θέλουμε να κάνουμε την πράξη γ:=(α+β)/2. Αφότου ανοίξουμε το block παίρνουμε T_1 = α + β έπειτα παίρνουμε T_2 = T_1 / 2 και γ = T_2 αν κάνουμε αντικατάσταση του T_1 της 1ης σχέσης και του T_2 της 3ης σχέσης στην δεύτερη σχέση παίρνουμε γ = (α+β)/2 δηλαδή αυτό που θέλουμε. Έπειτα από αυτά κλείνει το block της συνάρτησης. Στο block του κύριου προγράμματος τώρα ξεκινάμε με 2 σωστές αναθέσεις τιμών στα α,β και στην συνέχεια 3 περάσματα παραμέτρων τα α και β είναι είσοδοι της συνάρτησης περασμένοι με αναφορά (το οποίο είναι σωστό καθώς δεν έχουν %) ενώ το T_3 είναι η μια προσωρινή μεταβλητή που αποθηκεύει την επιστρεφόμενη τιμή. Στη συνέχεια καλούμε την συνάρτηση μέσος_όρος και θέτουμε την T3=δ δηλαδή επιστρέφουμε εκεί το αποτέλεσμα. Τέλος κλείνουμε το block και κάνουμε halt. Από όλα αυτά συμπεραίνουμε ότι και ο ενδιάμεσος κώδικας του προγράμματος παρήχθησε σωστά.

Τέλος φτάνουμε στον τελικό κώδικα του προγράμματος ο οποίος είναι

```
# Quad 1: 1: begin_block , μέσος_όρος , _ , _
μέσος_όρος:
        sw $ra, 0($sp)
        addi $sp, $sp, -28
        sw $fp, 4($sp)
        move $fp, $sp
# Quad 2: 2: + , α , β , T_1
lw t0, -4(sp)
        addi t0, t0, --4
        lw $t1, ($t0)
lw t0, -4(sp)
        addi t0, t0, --8
        lw $t2, ($t0)
add $t3, $t1, $t2
```

```
lw t0, -4(sp)
	addi t0, t0, --28
	sw $t3, ($t0)
# Quad 3: 3: / , T_1 , 2 , T_2
lw t0, -4(sp)
	addi t0, t0, --28
	lw $t1, ($t0)
li $t2, 2
div $t3, $t1, $t2
lw t0, -4(sp)
	addi t0, t0, --32
	sw $t3, ($t0)
# Quad 4: 4: := , T_2 , _ , γ
lw t0, -4(sp)
	addi t0, t0, --32
	lw $t1, ($t0)
lw t0, -4(sp)
	addi t0, t0, --12
	sw $t1, ($t0)
# Quad 5: 5: end_block , μέσος_όρος , _ , _
	move $sp, $fp
	lw $fp, 4($sp)
	addi $sp, $sp, 28
	lw $ra, 0($sp)
	jr $ra
# Quad 6: 6: begin_block , Μπάμπης , _ , _
Lmain:
	addi $sp, $sp, -28
	move $s0, $sp
# Quad 7: 7: := , 1 , _ , α
li $t1, 1
lw t0, -4(sp)
	addi t0, t0, --4
	sw $t1, ($t0)
# Quad 8: 8: := , 3 , _ , β
li $t1, 3
lw t0, -4(sp)
	addi t0, t0, --8
	sw $t1, ($t0)
# Quad 9: 9: par , α , cv , _
	addi $fp, $sp, 28
lw t0, -4(sp)
	addi t0, t0, --4
	lw $t1, ($t0)
sw $t1, -12($fp)
# Quad 10: 10: par , β , cv , _
lw t0, -4(sp)
	addi t0, t0, --8
```

```
        lw $t1, ($t0)
sw $t1, -16($fp)
# Quad 11: 11: par , T_3 , ret , _
addi $t0, $sp, --28
sw $t0, -8($fp)
# Quad 12: 12: call , μέσος_όρος , _ , _
                lw $t0,-4($sp)
                sw $t0,-4($fp)
                addi $sp,$sp,28
                jal μέσος_όρος
                addi $sp,$sp,-28
# Quad 13: 13: := , T_3 , _ , δ
lw $t1, --28($s0)
sw $t1, --16($s0)
# Quad 14: 14: halt , _ , _ , _
                li $v0, 10
                syscall
# Quad 15: 15: end_block , Μπάμπης , _ , _
        move $sp, $fp
        lw $fp, 4($sp)
        addi $sp, $sp, 28
        lw $ra, 0($sp)
        jr $ra
```

Παίρνοντας έναν έναν τον κώδικα που δημιουργήσαμε για κάθε quad έχουμε
Quad1 : Σώζεται ο ra (επιστροφή), αποθηκεύεται το fp
Quad2 :Βρίσκει τα α,β τα φορτώνει και μετά τα αποθηκεύει σε προσωρινούς καταχωρητές.
Quad3 : Παρόμοιο με το 2ο Quad μόνο που τώρα εκτελείτε πράξη διαίρεσης.
Quad4 : Αντιγράφεται η τιμή του T_2 στο offset του γ (δηλαδή στο -12)
Quad5 : Επαναφορά για το τέλος της συνάρτησης και επιστροφή.
Quad6 : δημιουργία νέου Label για το πρόγραμμα και activation record
Quad7 : Φωρτωση του α και ανάθεση τιμής li $t1, 1
Quad8 : Όμοια με το 7 αλλά για άλλη μεταβλητή και άλλη σταθερά
Quad9 : Περνάμε το α τιμή (δηλαδη αποθηκεύουμε την τιμή του και όχι την διεύθυνσή του)
Quad10 : Όμοια με το 9
Quad11 : Επιστροφή τιμής στο fp
Quad12: Εκτέλεση jal, με αποθήκευση δυναμικής βάσης έπειτα γίνεται αύξηση της στοίβας και αποκατάσταση στο τέλος.
Quad13 : Μεταφορά του αποτελέσματος στην θέση της δ (όπως είδαμε και παραπάνω)
Quad14 : Τερματισμός προγράμματος
Quad15 : Έξοδος από το πρόγραμμα

Όπως μπορούμε να δούμε όλα δουλεύουν σωστά και ο κώδικας μας ξεκινόντας από ένα αρχικό πρόγραμμα greek++ μετατρέπεται σε τελικό κώδικα σε assembly.