Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

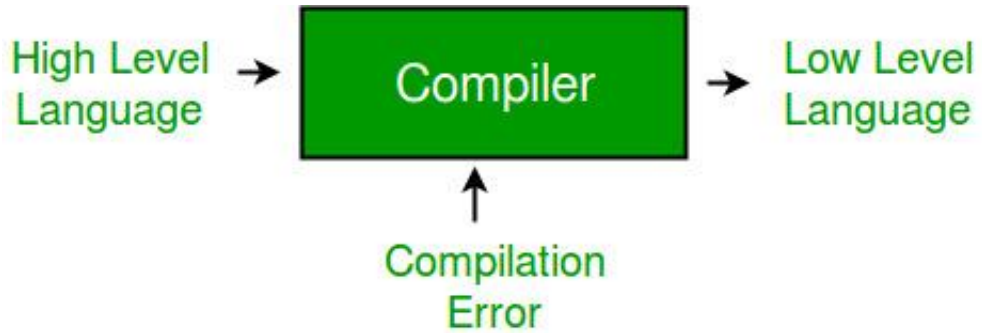
Προπτυχιακό Μάθημα: «Μεταφραστές»

Προγραμματιστική Εργασία Εξαμήνου

<u>Μέλη Ομάδας – Α.Μ.:</u>

Λάμπρος Βλαχόπουλος – 2948

Γεώργιος Κρομμύδας – 3260



ΙΩΑΝΝΙΝΑ,

2021

Πίνακας περιεχομένων

Μέρος – 1ο: Εισαγωγή:

Ο ***Μεταγλωττιστής*** ή ***μεταφραστής (compiler)*** είναι ένα πρόγραμμα του λειτουργικού συστήματος που διαβάζει κώδικα γραμμένο σε μια γλώσσα προγραμματισμού (την πηγαία γλώσσα) και τον μεταφράζει σε ισοδύναμο κώδικα σε μια άλλη γλώσσα προγραμματισμού (τη γλώσσα στόχο, συνήθως σε Assembly). Το κείμενο της εισόδου ονομάζεται *πηγαίος κώδικας* (source code), ενώ η έξοδος του προγράμματος, η οποία συχνά έχει δυαδική μορφή, *αντικειμενικός κώδικας* (object code).

Στην συγκεκριμένη εργασία είχαμε να υλοποιήσουμε τον μεταγλωττιστή της γλώσσας Cimple. Η ανάπτυξη του μεταγλωττιστή χωρίζεται σε έξι στάδια. Τα στάδια είναι τα εξής:

I. Λεκτικός Αναλυτής
II. Συντακτικός Αναλυτής
III. Παραγωγή Ενδιάμεσου κώδικα
IV. Σημασιολογική Ανάλυση
V. Πίνακας Συμβόλων
VI. Παραγωγή Τελικού Κώδικα

Αυτά τα στάδια μπορούν να χωριστούν σε δύο κατηγορίες, το **front-end** κομμάτι που αποτελεί τα στάδια από το I μέχρι το IV και το **back-end** που αποτελεί το στάδιο VI. Το V στάδιο αποτελεί την διασύνδεση μεταξύ των δύο κομματιών του μεταγλωττιστή. Παρακάτω θα αναλυθούν τα κομμάτια αυτά.

Η γλώσσα Cimple είναι μία μικρή γλώσσα προγραμματισμού, η οποία αντλεί τα χαρακτηριστικά της από την γλώσσα προγραμματισμού C. Στην ουσία αποτελεί μία πιο απλή εκδοχή της γλώσσας με συγκεκριμένες δομές. Οι μεταβλητές της γλώσσας είναι μόνο ακέραιοι αριθμοί. Το πέρασμα παραμέτρων γίνεται με τιμή και με αναφορά.

Ο σκοπός του συγκεκριμένου μεταγλωττιστή είναι να δέχεται ως είσοδο προγράμματα της παραπάνω γλώσσας (με κατάληξη .ci) και να εμφανίζει σε γλώσσα Assembly του MIPS τον τελικό παραγόμενο κώδικα.

Μέρος - 2ο: Υλοποίηση Μεταγλωττιστή:

Σε αυτό το κομμάτι έχουμε την ανάλυση των σταδίων του μεταγλωττιστή και την εκάστοτε λειτουργία τους που εκτελούν, έτσι ώστε να γίνει με ορθό τρόπο η παραγωγή του τελικού κώδικα.

2.1 Λεκτικός Αναλυτής

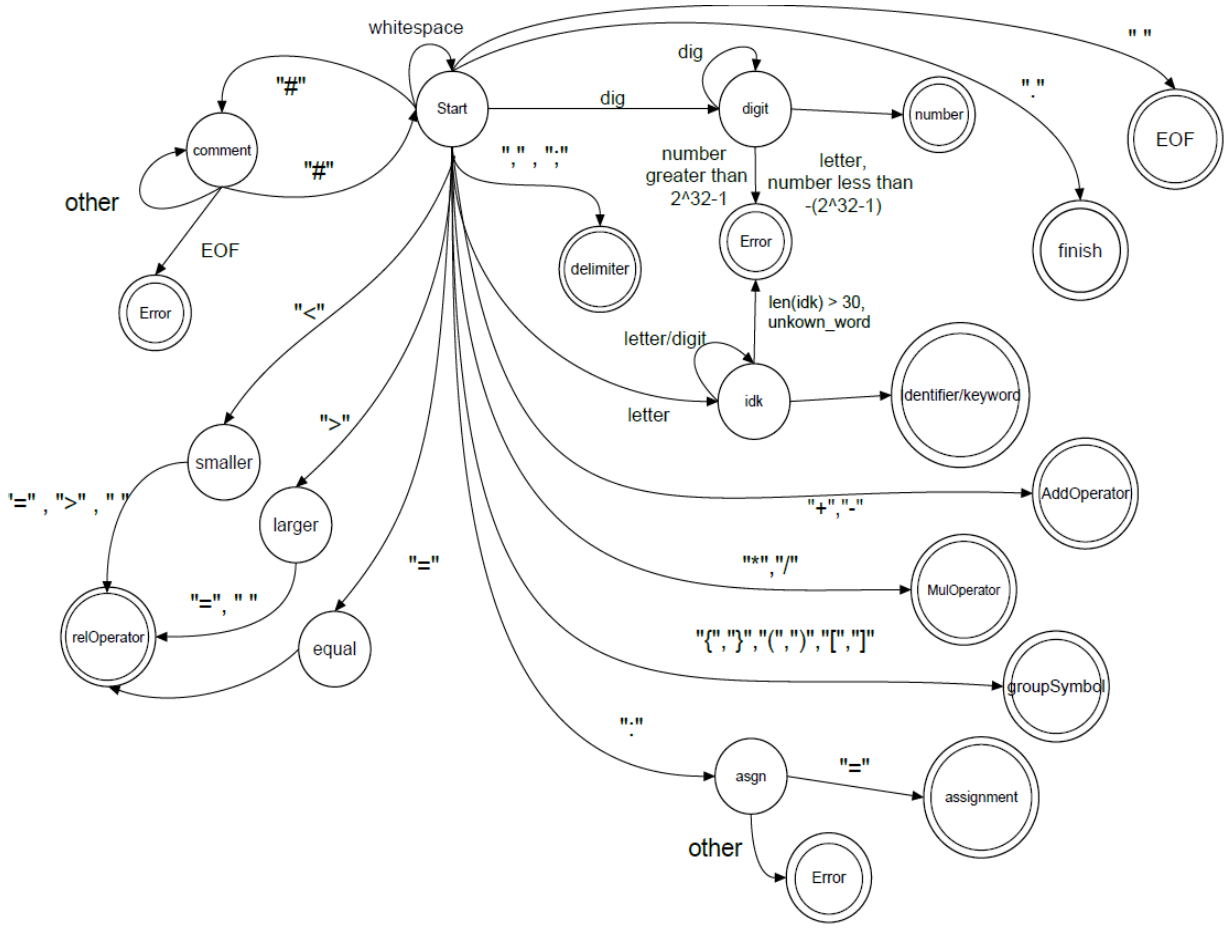
Το πρώτο μέρος ενός μεταγλωττιστή είναι ο Λεκτικός Αναλυτής. Η υλοποίησή του βασίζεται σε ένα μη ντετερμινιστικό αυτόματο πεπερασμένων καταστάσεων. Κάθε κατάσταση αποτελεί κάποιο χαρακτηριστικό της γλώσσας το οποίο ονομάζεται λεκτική μονάδα (**token**). Ένα πρόγραμμα αποτελείται από λεκτικές μονάδες, οι οποίες αναγνωρίζονται από το πεπερασμένο αυτόματο. Ο λεκτικός αναλυτής αποτελεί κομμάτι (συνάρτηση) του συντακτικού αναλυτή. Η λεκτική ανάλυση διαβάζει ένα ένα τα σύμβολα και δημιουργεί λεκτικές μονάδες. Κάθε σύμβολο της γλώσσας αποτελεί διαφορετική λεκτική μονάδα. Έτσι παράγεται μία λίστα η οποία περιέχει όλα τα tokens του προγράμματος. Μία λεκτική μονάδα ορίζεται ως ένα αντικείμενο τύπου Token που έχει ως πεδία τον τύπο του, το όνομα του και την γραμμή στην οποία διαβάστηκε. Οι λεκτικές μονάδες της γλώσσας είναι οι εξής:

- Μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (***A,…,Z*** και ***a,…,z*****)**
- Αριθμητικά ψηφία **(*0, …, 9*)**
- Σύμβολα αριθμητικών πράξεων **(+, -, *, /)**
- Τελεστές συσχέτισης**(<, >, =, <=, >=, <>)**
- Σύμβολο ανάθεσης **(:=)**
- Διαχωριστές **(“;” , “,”, “:”)**
- Σύμβολα ομαδοποίησης **([,], (,), {, })**
- Σύμβολο τερματισμού **(.)**
- Διαχωριστές σχολίων **(#)**

Επιπλέον, υπάρχουν και οι δεσμευμένες λέξεις, οι οποίες δεν μπορούν να δηλωθούν ως μεταβλητές. Οι λέξεις αυτές είναι ***[“program”, “declare”, “if”, “else”, “while”,***

"switchcase", "forcase", "incase", "case", "default", "not", "and", "or", "function", "procedure", "call", "return", "in", "inout", "input", "print"].

Ο λεκτικός αναλυτής βασίζεται στο πεπερασμένο αυτόματο που φαίνεται παρακάτω:



Εικόνα 1: Πεπερασμένο Αυτόματο Λεκτικού Αναλυτή

Αρχικά, ξεκινά από την κατάσταση **start** και διαβάζει την πρώτη λεκτική μονάδα. Ανάλογα με το περιεχόμενο είτε μεταβαίνει σε διαφορετική κατάσταση είτε παραμένει στην ίδια. Σε περίπτωση που μείνει στην ίδια, τότε διαβάζει τον επόμενο χαρακτήρα και αντίστοιχα μεταβαίνει σε διαφορετική κατάσταση. Όταν μεταβεί σε μία τελική κατάσταση, τότε θα επιστρέψει το αντίστοιχο αντικείμενο. Το αντικείμενο αυτό θα περιέχει τα τρία πεδία **(tokenType, tokenString, lineNo)**. Σε περίπτωση εσφαλμένης ανάγνωσης, τότε επιστρέφει ένα μήνυμα σφάλματος ο μεταγλωττιστής και τερματίζει την διαδικασία της μετάφρασης. Παρακάτω, φαίνονται οι βοηθητικές μεταβλητές του λεκτικού αναλυτή που χρησιμοποιήθηκαν για την δημιουργία του.

Βοηθητικές Μεταβλητές	Περιγραφή
tokenString	Μία λίστα που περιέχει τις λεκτικές μονάδες που διαβάστηκαν από το πεπερασμένο αυτόματο. Μέσα στην λίστα γίνεται **append** το **char** που διαβάστηκε.
counter_for_letters	Είναι ένας μετρητής γραμμάτων των λέξεων που διαβάζονται. Ο τύπος των λέξεων είναι είτε **identifier** είτε **keywordString**.
char	Είναι οι χαρακτήρες όλων των λεκτικών μονάδων του προγράμματος. Ο κάθε χαρακτήρας διαβάζεται ένα ένα μέσω της **infile.read(1)**. Σε περίπτωση που διαβαστεί κάτι διαφορετικό, τότε γυρνάμε μία θέση πίσω τον μετρητή (**file pointer**) στην θέση που χρειαζόμαστε μέσω της **infile.seek(infile.tell()-1)**
counter_for_lines	Είναι ένας μετρητής γραμμών που επιστρέφει την γραμμή που βρέθηκε μία λεκτική μονάδα.

Πίνακας 1: Μεταβλητές Λεκτικού Αναλυτή

2.2 Συντακτικός Αναλυτής

Το επόμενο στάδιο της υλοποίησης είναι ο συντακτικός αναλυτής. Είναι υπεύθυνος για την δημιουργία του συντακτικού δένδρου ενός προγράμματος από τις λεκτικές μονάδες. Κάνει τον έλεγχο για την διαπίστωση του πηγαίου προγράμματος. Δηλαδή, ελέγχει εάν το πρόγραμμα ανήκει ή δεν ανήκει στην γλώσσα Cimple. Βασίζεται πάνω την γραμματική ανεξάρτητων συμφραζομένων **LL(1)**, η οποία αναγνωρίζει από τα αριστερά προς τα δεξιά τις λεκτικές μονάδες και παράγει την αριστερότερο δυνατό συντακτικό δένδρο. Σε περίπτωση που βρίσκεται σε δίλλημα ποιον κανόνα να ακολουθήσει, τότε κοιτάει το αμέσως επόμενο σύμβολο στην συμβολοσειρά εισόδου. Κάθε κανόνας της γραμματικής αποτελεί μία συνάρτηση του συντακτικού αναλυτή. Όταν συναντηθεί μη τερματικό σύμβολο, τότε καλείται η αντίστοιχη συνάρτηση. Σε

διαφορετική περίπτωση, εάν και ο λεκτικός αναλυτής επιστρέφει λεκτική μονάδα που αντιστοιχεί σε τερματικό σύμβολο, τότε αναγνωρίζεται η λεκτική μονάδα επιτυχώς. Στην περίπτωση που δεν επιστραφεί λεκτική μονάδα που περιμένει ο συντακτικός αναλυτής, τότε εμφανίζεται μήνυμα λάθους. Παρακάτω φαίνεται η γραμματική της γλώσσας:

```
# "program" is the starting symbol
program         :    program ID block .

# a block with declarations, subprogram and statements
block           :    declarations subprograms statements

# declaration of variables, zero or more "declare" allowed
declarations    :    ( declare varlist ; )*

# a list of variables following the declaration keyword
varlist         :    ID  ( , ID )*
                |    ε


# zero or more subprograms allowed
subprograms     :    ( subprogram )*

# a subprogram is a function or a procedure,
# followed by parameters and block
subprogram      :    function ID ( formalparlist ) block
                |    procedure ID ( formalparlist ) block

# list of formal parameters
formalparlist   :    formalparitem  ( , formalparitem )*
                |    ε

# a formal parameter ("in": by value, "inout" by reference)
formalparitem   :    in ID
                |    inout ID

# one or more statements
statements      :    statement ;
                |    { statement  ( ; statement )* }
```

```
# one statement
statement      :    assignStat
               |    ifStat
               |    whileStat
               |    switchcaseStat
               |    forcaseStat
               |    incaseStat
               |    callStat
               |    returnStat
               |    inputStat
               |    printStat
               |    ε

# assignment statement
assignStat     :    ID := expression

# if statement
ifStat         :    if ( condition ) statements elsepart

elsepart       :    else statements
               |    ε
```

```
# while statement
whileStat      :    while ( condition ) statements

# switch statement
switchcaseStat:     switchcase
                       ( case ( condition ) statements )*
                       default statements

# forcase statement
forcaseStat    :    forcase
                        ( case ( condition ) statements )*
                        default statements

# incase statement
incaseStat     :    incase
                         ( case ( condition ) statements )*

# return statement
returnStat     :    return( expression )

# call statement
callStat       :    call ID( actualparlist )

# print statement
printStat      :    print( expression )

# input statement
inputStat      :    input( ID )

# list of actual parameters
actualparlist :     actualparitem ( , actualparitem )*
               |    ε
```

```
# an actual parameter ("in": by value, "inout" by reference)
actualparitem :    in expression
              |    inout ID

# boolean expression
condition     :    boolterm ( or boolterm )*


 # term in boolean expression
 boolterm      :    boolfactor ( and boolfactor )*

 # factor in boolean expression
 boolfactor    :    not [ condition ]
               |    [ condition ]
               |    expression REL_OP expression

 # arithmetic expression
 expression    :    optionalSign term ( ADD_OP term )*

 # term in arithmetic expression
 term          :    factor  ( MUL_OP factor )*

 # factor in arithmetic expression
 factor        :    INTEGER
               |    ( expression )
               |    ID idtail

 # follows a function of procedure (parethnesis and parameters)
 idtail        :    ( actualparlist )
               |    ε

 # sumbols "+" and "-" (are optional)
 optionalSign  :    ADD_OP
               |    ε


# lexer rules: relational, arithentic operations, integers and ids
REL_OP        :    = | <= | >=  | > | < | <>
              ;
ADD_OP        :    + | -

MUL_OP        :    * | /

INTEGER       :    [0-9]+

ID            :    [a-zA-Z][a-zA-Z0-9]*
```

Εικόνα 2: Γραμματική Γλώσσας Cimple

Στην περίπτωση σφαλμάτων, τότε θα εμφανιστεί ένα μήνυμα λάθους, το οποίο αναφέρει αναγράφει τον τύπο της λεκτικής μονάδας που περίμενε. Επίσης, εμφανίζεται και στην γραμμή που εμφανίστηκε το λάθος και η εσφαλμένη λεκτική μονάδα που διαβάστηκε. Κατά το πέρας της εκτέλεσης του συντακτικού αναλυτή και όλων των κανόνων της γραμματικής, η τελευταία λεκτική μονάδα που θα εμφανιστεί θα πρέπει να είναι η **“.”**. Σε περίπτωση που η επόμενη λεκτική μονάδα δεν είναι **EOF**, τότε διαβάζει και τις επόμενες λεκτικές μονάδες και τις επιστρέφει με ένα μήνυμα WARNING, πως έχει βρεθεί κώδικας μετά το τερματικό σύνολο.

Τώρα ας δούμε μία μία τις συναρτήσεις του συντακτικού αναλυτή. Αρχικά έχουμε την συνάρτηση ***program()*** η οποία σηματοδοτεί την έναρξη της μεταγλώττισης ενός προγράμματος. Ένα πρόγραμμα σε ***Cimple*** είναι της

$$\boldsymbol{program}\ ID\ \boldsymbol{block}\ .$$

Εφόσον διαβαστούν όλα τα blocks του προγράμματος και βρεθεί η τελεία **“.”**, τότε τελειώνει η συντακτική ανάλυση. Η συνάρτηση ***block()*** περιέχει τις συναρτήσεις ***declarations, subprograms*** και ***statements***. οι οποίες αποτελούν την δομή ενός προγράμματος της γλώσσας.

Η συνάρτηση ***declerations*** διαβάζει όλες τις μεταβλητές που ορίστηκαν στην αρχή του προγράμματος και εμφανίζονται μετά την δεσμευμένη λέξη ***declare***. Όταν διαβαστεί αυτή, τότε καλείται η συνάρτηση ***varlist()*** και στην συνέχεια διαβάζεται ο χαρακτήρας “;”. Σε περίπτωση που δεν υπάρχει αρχικοποίηση κάποιας μεταβλητής, τότε η συνάρτηση δεν επιστρέφει κάτι και συνεχίζει συντακτική ανάλυση των μονάδων στις επόμενες συναρτήσεις. Η συνάρτηση ***varlist()*** θα πρέπει να επιστρέφει τουλάχιστον μία μεταβλητή, εφόσον καλέστηκε. Διαβάζει το όνομα της και στην συνέχεια προχωράει στην επόμενη λεκτική μονάδα. Στην περίπτωση που υπάρχουν παραπάνω αρχικοποιήσεις μεταβλητών, τότε πρέπει να διαβαστεί ανάμεσα τους ο χαρακτήρας **“,”** για να τελειώσει ο συντακτικός αναλυτής με τις αρχικοποιήσεις.

Το επόμενο βήμα είναι να γίνει έλεγχος υποπρογραμμάτων. Γενικά ένα πρόγραμμα δεν είναι απαραίτητο να έχει. Στην περίπτωση που δεν έχει, τότε ο συντακτικός αναλυτής προχωράει και ελέγχει τα ***statements()***. Σε διαφορετική περίπτωση καλείται η συνάρτηση ***subprogram()***. Η συνάρτηση αυτή θα διαβάσει αρχικά είτε μία συνάρτηση είτε μία διαδικασία. Μία συνάρτηση ή μια διαδικασία, εκτός από το όνομα

της, έχει και παραμέτρους που δηλώνονται από την συνάρτηση ***formalparlist()***. Στη συνέχεια καλείται η συνάρτηση ***block()*** για την δομή του εκάστοτε υποπρογράμματος που είναι για μεταγλώττιση. Αρχικά διαβάζεται η δεσμευμένη λέξη του υποπρογράμματος και στη συνέχεια το όνομα του. Ύστερα πρέπει να διαβαστεί ο χαρακτήρας **"("** και να καλεστεί η συνάρτηση για δήλωση των παραμέτρων.

Η συνάρτηση ***formalparlist()*** μπορεί να επιστρέψει είτε καμία παράμετρο, είτε μία είτε πολλές. Εάν υπάρχουν πολλές, τότε καλείται η συνάρτηση ***formalparitem()*** και παράλληλα θα πρέπει να υπάρχει ο χαρακτήρας ***","***. Η συνάρτηση ***formalparitem()***, αρχικά διαβάζει τον τύπο της παραμέτρου και μετά το όνομά του. Ο τύπος μια τυπικής παραμέτρου μπορεί να είναι είτε ***"in"*** που σημαίνει πως η παράμετρο περνάει με τιμή, είτε ***"inout"*** που σημαίνει πως η παράμετρος περνά με αναφορά. Σε περίπτωση ανυπαρξίας παραμέτρων, τότε υπάρχει σφάλμα από τον συντακτικό αναλυτή. Αφού διαβαστούν και οι παράμεροι, τότε η επόμενη λεκτική μονάδα πρέπει να είναι ο χαρακτήρας ***")"***, και στην συνέχεια έπεται η κλήση της ***blok()***.

Το επόμενο και τελικό βήμα είναι η συνάρτηση ***statements()***, η οποία περίεχει όλες τις δομές ενός προγράμματος. Υπάρχει γενικά τουλάχιστον μία δομή σε ένα πρόγραμμα. Η δομή αυτή καλείται από τη συνάρτηση ***statement()***. Εν συνεχεία, θα πρέπει να διαβαστεί και ο χαρακτήρας ***";"***. Στην περίπτωση που έχουμε πολλαπλά statements, τότε θα πρέπει αρχικά να διαβαστεί ο χαρακτήρας ***"{"***. Ύστερα, καλείται η συνάρτηση ***statement()*** και η επόμενη λεκτική μονάδα που περιμένουμε είναι είτε ο χαρακτήρας ***";"*** και στη συνέχεια καλείται ξανά η παραπάνω συνάρτηση, είτε πρέπει να διαβαστεί ο χαρακτήρας ***"}"***.

Κάθε φορά που καλείται ένα statement, αυτό μπορεί να είναι ένα από τις εξής συναρτήσεις:

- ***assignStat()***: Καλείται σε περίπτωση εκχώρησης κάποιας τιμής ή έκφρασης και όταν αναγνωριστεί ο χαρακτήρας ***":="***. Αφού διαβαστεί, τότε καλείται η συνάρτηση ***expression()*** η οποία αποτιμά την έκφραση.
- ***ifStat()***: Καλείται όταν ο λεκτικός αναλυτής αναγνωρίσει την δεσμευμένη λέξη "***if***'. Στη συνέχεια, ελέγχεται η ύπαρξη της αριστερής παρένθεσης. Αν υπάρχει, τότε καλείται η συνάρτηση ***condition()***. Έπειτα ελέγχει εάν υπάρχει και η δεξιά παρένθεση. Στην περίπτωση ύπαρξης καλείται η ξανά

η συνάρτηση ***statements()*** για αναγνώριση περαιτέρω δομών. Τέλος, καλεί την συνάρτηση ***elsepart()***. Η επόμενη λεκτική μονάδα που θα διαβάζει μπορεί να είναι είτε το κενό, είτε η δεσμευμένη λέξη ***"else"***. Στην περίπτωση του κενού χαρακτήρα, τερματίζει ο συντακτικός έλεγχος της συγκεκριμένης δομής και συνεχίζεται η μεταγλώττιση. Διαφορετικά, διαγνώσθηκε η λέξη ***"else"*** και εν συνεχεία καλείται η συνάρτηση ***statements()*** για τον έλεγχο ύπαρξης παραπάνω δομών.

- ***whileStat()***: Καλείται όταν ο λεκτικός αναλυτής αναγνωρίσει την δεσμευμένη λέξη "***while***". Στη συνέχεια, ελέγχεται η ύπαρξη της αριστερής παρένθεσης. Αν υπάρχει, τότε καλείται η συνάρτηση ***condition()***. Έπειτα ελέγχει εάν υπάρχει και η δεξιά παρένθεση. Στην περίπτωση ύπαρξης καλείται η ξανά η συνάρτηση ***statements()*** για αναγνώριση περαιτέρω δομών μέχρι να αποτιμηθούν όλες.
- ***switchcaseStat()***: Καλείται όταν ο λεκτικός αναλυτής αναγνωρίσει την δεσμευμένη λέξη "***switchcase***". Στη συνέχεια, η επόμενη λέξη που διαγνώσκει είναι η δεσμευμένη λέξη ***"case"***. Στη συνέχεια θα πρέπει να διαβαστεί η αριστερή παρένθεση. Σε αυτή την περίπτωση θα καλεστεί η συνάρτηση ***condition()*** που θα αποτιμήσει τις συνθήκες και εν συνεχεία πρέπει να διαβαστεί η δεξιά παρένθεση. Αν ο έλεγχος είναι σωστός μέχρι εκείνο το σημείο, τότε θα κληθεί η συνάρτηση ***statements()***. Αυτός ο έλεγχος επαναλαμβάνεται όσο εμφανίζεται η δεσμευμένη λέξη "***case"***. Το επόμενο βήμα είναι να διαβαστεί η δεσμευμένη λέξη ***"default"***. Όταν διαβαστεί, τότε καλείται εκ νέου η συνάρτηση ***statements()*** για να αποτιμηθούν οι υπόλοιπες δομές.
- ***forcaseStat()***: Καλείται όταν ο λεκτικός αναλυτής αναγνωρίσει την δεσμευμένη λέξη "***forcase***". Στη συνέχεια, η επόμενη λέξη που διαγνώσκει είναι η δεσμευμένη λέξη ***"case"***. Στη συνέχεια θα πρέπει να διαβαστεί η αριστερή παρένθεση. Σε αυτή την περίπτωση θα καλεστεί η συνάρτηση ***condition()*** που θα αποτιμήσει τις συνθήκες και εν συνεχεία πρέπει να διαβαστεί η δεξιά παρένθεση. Αν ο έλεγχος είναι σωστός μέχρι εκείνο το σημείο, τότε θα κληθεί η συνάρτηση ***statements()***. Αυτός ο έλεγχος επαναλαμβάνεται όσο εμφανίζεται η δεσμευμένη λέξη "***case"***. Το επόμενο βήμα είναι να διαβαστεί η δεσμευμένη λέξη ***"default"***. Όταν διαβαστεί, τότε

καλείται εκ νέου η συνάρτηση ***statements()*** για να αποτιμηθούν οι υπόλοιπες δομές.

- ***incaseStat()***: Καλείται όταν ο λεκτικός αναλυτής αναγνωρίσει την δεσμευμένη λέξη "***incase***". Στη συνέχεια, η επόμενη λέξη που διαγνώσκει είναι η δεσμευμένη λέξη ***"case"***. Στη συνέχεια θα πρέπει να διαβαστεί η αριστερή παρένθεση. Σε αυτή την περίπτωση θα καλεστεί η συνάρτηση ***condition()*** που θα αποτιμήσει τις συνθήκες και εν συνεχεία πρέπει να διαβαστεί η δεξιά παρένθεση. Αν ο έλεγχος είναι σωστός μέχρι εκείνο το σημείο, τότε θα κληθεί η συνάρτηση ***statements()***. Αυτός ο έλεγχος επαναλαμβάνεται όσο εμφανίζεται η δεσμευμένη λέξη "***case"***.
- ***callStat()***: Καλείται όταν ο λεκτικός αναλυτής αναγνωρίσει την δεσμευμένη λέξη "***call***". Χρησιμοποιείται μόνο σε περίπτωση κλήσης μιας διαδικασίας. Το επόμενο βήμα είναι να αναγνωρίσει το όνομα της συγκεκριμένης διαδικασίας. Αν το όνομα της ID γίνεται δεκτό από τον συντακτικό αναλυτή, τότε μεταβαίνει ο έλεγχος στην επόμενη λεκτική μονάδα η οποία θα πρέπει να είναι η αριστερή παρένθεση. Αν εμφανιστεί, τότε καλείται η συνάρτηση ***actualparlist()***, η οποία θα αποτιμήσει τις παραμέτρους της διαδικασίας. Έπειτα πρέπει να ελεγχθεί η δεξιά παρένθεση από τον συντακτικό αναλυτή. Στην περίπτωση που υπάρχει τότε τελειώνει ο έλεγχος της συγκεκριμένης εντολής.
- ***returnStat()***: Καλείται όταν ο λεκτικός αναλυτής διαβάσει την δεσμευμένη λέξη ***"return"***. Αυτή η εντολή χρησιμοποιείται στην περίπτωση που έχουμε ως υποπρόγραμμα μία συνάρτηση και θέλουμε να επιστρέψουμε το αποτέλεσμα της. Το επόμενο που πρέπει να ελεγχθεί είναι η ύπαρξη της αριστερής παρένθεσης. Σε περίπτωση που υπάρχει καλούμε την συνάρτηση ***expression()***, όπου θα αποτιμηθεί η τιμή επιστροφής. Τέλος, θα πρέπει να ελεγχθεί και η ύπαρξη δεξιάς παρένθεσης. Αν υπάρχει τότε τελειώνει ο έλεγχος της εντολής και συνεχίζει ο συντακτικός αναλυτής στον έλεγχο της επόμενης λεκτικής μονάδας.
- ***inputStat()***: Καλείται όταν ο λεκτικός αναλυτής διαβάσει την δεσμευμένη λέξη ***"input"***. Εν συνεχεία, ελέγχει την ύπαρξη της αριστερής παρένθεσης. Αν έχει διαβαστεί, τότε το επόμενο βήμα είναι να διαβάσει το όνομα της μεταβλητής που καλείται να εισαχθεί από το πληκτρολόγιο. Το όνομα της

ελέγχεται εάν υπάρχει και εάν ανήκει στην γλώσσα. Σε περίπτωση που ανήκει, τότε διαβάζει και ελέγχει την επόμενη λεκτική μονάδα. Αν εμφανιστεί η δεξιά παρένθεση, τότε ο έλεγχος της εντολής είναι επιτυχής και μεταβαίνει στην επόμενη λεκτική μονάδα.

- ***printStat()***: Καλείται όταν ο λεκτικός αναλυτής διαβάσει την δεσμευμένη λέξη ***"print"***. Εν συνεχεία, ελέγχει την ύπαρξη της αριστερής παρένθεσης. Αν έχει διαβαστεί, τότε το επόμενο βήμα είναι να καλέσει την συνάρτηση ***expression()*** για να αποτιμήσει την έκφραση που επρόκειτο να τυπωθεί στην οθόνη. Μετά διαβάζει την επόμενη λεκτική μονάδα. Αν εμφανιστεί η δεξιά παρένθεση, τότε ο έλεγχος της εντολής είναι επιτυχής και μεταβαίνει στην επόμενη λεκτική μονάδα.

Όταν πρόκειται να καλεστεί μία διαδικασία ή μία συνάρτηση από το κυρίως πρόγραμμα, τότε χρειαζόμαστε και την αποτίμηση των παραμέτρων της. Η αποτίμηση γίνεται μέσω της συνάρτησης ***actualparlist()***. Αυτή με την σειρά της καλεί την συνάρτηση ***actualparitem()*** η οποία ελέγχει τις παραμέτρους μία μία. Πρέπει να υπάρχει τουλάχιστον μία παράμετρος μέσα σε μία διαδικασία ή σε μία συνάρτηση αντίστοιχα. Στην περίπτωση που έχουμε από δύο και πάνω, τότε πρέπει αυτές να διαχωρίζονται μεταξύ τους με τον χαρακτήρα ***","***. Επιπλέον, στην περίπτωση που έχουμε τουλάχιστον μία παράμετρο, τότε μεταβαίνουμε στην συνάρτηση ***actualparitem()***. Σε περίπτωση που η επόμενη λεκτική μονάδα είναι το ***"in"***, τότε καλείται η συνάρτηση ***expression()*** για να αποτιμηθεί η έκφραση. Στην περίπτωση που διαβαστεί η λέξη ***"inout"*** τότε η επόμενη λεκτική μονάδα που πρέπει να διαβαστεί είναι το όνομα μιας μεταβλητής ID.

Το τελικό βήμα είναι να αποτιμήσουμε τις λογικές και αριθμητικές εκφράσεις. Ας ξεκινήσουμε πρώτα με τις λογικές. Η συνάρτηση ***condition()*** αποτιμά τις λογικές συνθήκες με τους τελεστές ***["or", "and", "not"]***. Αρχικά καλούμε την συνάρτηση ***boolterm()***, η οποία θα μας εξασφαλίσει την ύπαρξη τουλάχιστον μίας λογικής έκφρασης. Στην περίπτωση που έχουμε παραπάνω λογικές εκφράσεις, τότε θα πρέπει να ελέγξουμε την ύπαρξη του τελεστή ***"or"***. Εάν υπάρχει, τότε μπορούμε να προχωρήσουμε στις επόμενες εκφράσεις καλώντας την συνάρτηση ***boolterm()***.

Η συνάρτηση ***boolterm()*** αποτελεί το επόμενο κομμάτι μιας συνθήκης. Αποτιμά τους όρους που υπάρχουν στην συνθήκη. Πρώτα από όλα, καλεί την συνάρτηση

boolfactor(), η οποία θα αποτιμήσει τους παράγοντες που υπάρχουν στις συνθήκες. Σε περίπτωση που έχουμε παραπάνω όρους, τότε θα πρέπει η επόμενη λεκτική μονάδα που θα διαβαστεί να είναι η δεσμευμένη λέξη ***"and"*** και να καλείται η συνάρτηση ***boolfactor()***.

Η συνάρτηση ***boolfactor()*** αποτιμά τις ίδιες τις εκφράσεις που εμφανίζονται σε μία συνθήκη. Αρχικά ελέγχει εάν εμφανίζεται η δεσμευμένη λέξη ***"not"*** ή όχι. Σε περίπτωση που εμφανίζεται, τότε η επόμενη λεκτική μονάδα που πρέπει να εμφανιστεί είναι η αριστερή αγκύλη. Αν εμφανιστεί, τότε καλείται η συνάρτηση ***condition()*** και αποτιμάται η αντίστοιχη συνθήκη. Σε περίπτωση που δεν εμφανίζεται η παραπάνω δεσμευμένη λέξη, τότε στην θέση της θα πρέπει να διαβαστεί η αριστερή αγκύλη. Αν διαβαστεί, τότε καλείται επίσης η συνάρτηση ***condition()***. Και στις δύο περιπτώσεις αφού καλεστεί η συνάρτηση, τότε η επόμενη λεκτική μονάδα θα πρέπει να είναι η δεξιά αγκύλη και να σταματάει ο έλεγχος των συνθηκών. Στην περίπτωση που δεν διαβαστεί τίποτα από τα παραπάνω, θα πρέπει να καλεστεί η συνάρτηση ***expression***, η οποία θα αποτιμήσει την αριθμητική έκφραση. Επιπλέον, η επόμενη λεκτική μονάδα που θα διαβαστεί θα πρέπει να είναι ένας τελεστής συσχέτισης. Αυτός, θα ελεγχθεί από την συνάρτηση ***REL_OP()***. Τέλος, καλείται ξανά η συνάρτηση ***expression()*** για να αποτιμηθεί η αριθμητική έκφραση δεξιά του τελεστή συσχέτισης.

Η συνάρτηση ***expression()*** αποτιμά τις αριθμητικές εκφράσεις που υπάρχουν μέσε σε ένα πρόγραμμα. Το πρώτο πράγμα που γίνεται σε μία έκφραση είναι να ελεγχθεί το πρόσημο μιας έκφρασης. Αυτό γίνεται με την συνάρτηση ***optionalSign()***. Στη συνέχεια καλείται η συνάρτηση ***term()***, η οποία θα επιστρέψει τους όρους τις έκφρασης. Επιπλέον, σε περίπτωση που η επόμενη λεκτική μονάδα είναι κάποιος προσθετικός αριθμητικός τελεστής, πρώτα ελέγχουμε με την συνάρτηση ***ADD_OP()*** εάν είναι προσθετική. Όσο έχουμε ότι ισχύει, τότε θα καλούμε την ***term()*** για να αποτιμήσουμε και τους όρους δεξιά του τελεστή. Σε άλλη περίπτωση προχωράμε στην επόμενη λεκτική μονάδα.

Η συνάρτηση ***term()*** αποτιμά τους όρους που βρίσκονται αριστερά και δεξιά ενός αριθμητικού τελεστή. Αρχικά, καλεί την συνάρτηση ***factor()*** η οποία θα ελέγξει τους όρους που εμφανίζονται σε μία αριθμητική έκφραση. Σε περίπτωση που η επόμενη λεκτική μονάδα είναι ένας πολλαπλασιαστικός αριθμητικός τελεστής, τότε καλούμε την συνάρτηση ***MUL_OP()***, η οποία θα ελέγξει τη συγκεκριμένη λεκτική μονάδα. Στη

συνέχεια καλείται η συνάρτηση ***factor()*** για να αποτιμηθούν και οι όροι δεξιά των τελεστών.

Η συνάρτηση ***factor()*** ελέγχει τους παράγοντες που εμφανίζονται στους όρους μιας αριθμητικής έκφρασης. Εάν έχουμε πως ο παράγοντας είναι ένας αριθμός, τότε καλείται η συνάρτηση ***INTEGER()*** και ελέγχεται εάν ο αριθμός είναι εντός των ορίων. Στην περίπτωση που η επόμενη λεκτική μονάδα είναι αριστερή παρένθεση, τότε στο επόμενο βήμα καλεί την συνάρτηση ***expression()*** για την αποτίμηση της αριθμητικής έκφρασης. Η επόμενη λεκτική μονάδα που πρέπει να διαβαστεί είναι η δεξιά παρένθεση για να τελειώσει ο έλεγχος αυτής της έκφρασης. Στην περίπτωση που η επόμενη λεκτική μονάδα που θα διαβαστεί είναι όνομα συνάρτησης ή μιας μεταβλητής, τότε το επόμενο βήμα είναι να καλέσουμε την συνάρτηση ***idtail()*** και στη συνέχεια να την επιστρέψουμε.

Στην περίπτωση μιας συνάρτησης, θα πρέπει να ελεγχθούν και οι παράμετροι της. Αυτή την δουλειά την κάνει η ***idtail()***. Αρχικά, η πρώτη λεκτική μονάδα που αναγνωρίζεται είναι η αριστερή παρένθεση. Στη συνέχεια, καλείται η συνάρτηση ***actualparlist()*** η οποία θα αποτιμήσει τις παραμέτρους της συνάρτησης. Τέλος, η επόμενη λεκτική μονάδα που ελέγχεται είναι δεξιά παρένθεση. Αν υπάρχει, τότε συνεχίζει στην επόμενη λεκτική μονάδα. Στην περίπτωση που δεν υπάρχει συνάρτηση, τότε η συνάρτηση ***idtail()*** επιστρέφει κενό για να μπορέσει να ανγωριστεί μία μεταβλητή.

Η συνάρτηση ***optionalSign()*** ελέγχει την ύπαρξη προσήμου μπροστά από μία έκφραση. Αν υπάρχει κάποιο, τότε καλεί την συνάρτηση ***ADD_OP()*** για να το ελέγξει. Σε διαφορετική περίπτωση, το πρόσημο είναι προαιρετικό και δεν υπάρχει ανάγκη ύπαρξης.

Οι επόμενες συναρτήσεις αποτελούν τους τελεστές των αριθμητικών, λογικών και των σχεσιακών. Η συνάρτηση ***REL_OP()*** ελέγχει τον σχεσιακό τελεστή ο οποίος μπορεί να είναι ένας από τους **[“=”, “<=”, “>=”, “<”, “>” “<>”]**. Η συνάρτηση ***ADD_OP()*** ελέγχει τον προσθετικό αριθμητικό τελεστή ο οποίος μπορεί να είναι είτε **“+”** είτε **“-”**. Η συνάρτηση ***MUL_OP()*** ελέγχει τον πολλαπλασιαστικό αριθμητικό τελεστή ο οποίος μπορεί να είναι είτε **“*”** είτε **“/”**. Η συνάρτηση ***INTEGER()*** επιστρέφει έναν ακέραιο δεκαδικό αριθμό ο οποίος ελέγχεται από τον λεκτικό αναλυτή,

ένα βρίσκεται στα όρια των ακεραιών που έχει η συγκεκριμένη γλώσσα. Τέλος, ένα όνομα ID θα πρέπει να έχει είτε μόνο μικρά και μεγάλα γράμματα, είτε να έχει επιπλέον και αριθμούς του δεκαδικού συστήματος. Με την μόνη διαφορά, πως οι αριθμοί αυτοί πρέπει να εμφανίζονται υποχρεωτικά μετά από τα γράμματα.

2.3 Παραγωγή Ενδιάμεσου Κώδικα

Το επόμενο στάδιο μεταγλώττισης είναι η παραγωγή του ενδιάμεσου κώδικα. Καθώς έχουν παραχθεί οι λεκτικές μονάδες από τον λεκτικό αναλυτή και παράλληλα ελέγχονται από τον συντακτικό αναλυτή αν είναι έγκυρες, τότε ξεκινάει η μετάφραση σε μία ενδιάμεση γλώσσα προγραμματισμού. Αυτή η ενδιάμεση γλώσσα αποτελείται από έναν χαρακτηριστικό αριθμό γραμμής (**ID**) και από μία τετράδα εντολών. Κάθε τετράδα αποτελεί μία αντίστοιχη έκφραση του πηγαίου κώδικα σε ενδιάμεσο. Έτσι, ξεκινάει σιγά σιγά η μεταγλώττιση σε ενδιάμεσο στάδιο και παράγεται μία πιο απλή μορφή του κώδικα. Η λειτουργία αυτή πραγματοποιείται βάσει κάποιων βοηθητικών υπορουτίνων. Είναι συναρτήσεις οι οποίες ξεκινούν να μεταφράζουν τον πηγαίο κώδικα σε ενδιάμεσο γραμμή γραμμή. Αυτές οι συναρτήσεις είναι οι εξής:

Βοηθητικές Υπορουτίνες	**Περιγραφή**
nextquad()	Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί
genquad(op, x, y, z)	Παράγει την τετράδα που πρόκειται να μεταφραστεί. Το πρώτο όρισμα **op** είναι η διαδικασία που εκτελεί η τετράδα και τα **x, y, z** είναι οι εκφράσεις που επρόκειτο να εκτελεστούν. Συγκεκριμένα τα ***x, y*** είναι οι **τελεσταίοι** και το ***z*** είναι ο **προορισμός**
newtemp()	Είναι μια γεννήτρια προσωρινών μεταβλητών και καλείται εκεί που χρειάζονται κατά την μετάφραση των εντολών. Οι προσωρινές μεταβλητές είναι της μορφής **T_i (i = 1,2,3,..)**
emptylist()	Επιστρέφει μία κενή λίστα από τετράδες

makelist(x)	Δημιουργεί μία καινούργια λίστα η οποία περιέχει τετράδες της **x**
mergelist(list1, list2)	Δημιουργεί μία καινούργια λίστα η οποία αποτελείται από τις τετράδες της **list1** και **list2.** Σε περίπτωση που κάποια λίστα είναι κενή, τότε τα στοιχεία της νέας λίστας θα περιέχει μόνο της τετράδες της μη κενής λίστας
backpatch(lst, z)	Πηγαίνει στις ήδη υπάρχουσες τετράδες και προσθέτει στην τελευταία θέση το **z**, εφόσον είναι κενή

Πίνακας 2: Βοηθητικές Υπορουτίνες Παραγωγής Ενδίαμεσου Κώδικα

Η δημιουργία των τετράδων βασίζεται στην κλάση ***Quad*** η οποία παράγει ως μια πεντάδα αντικειμένων τις παραγόμενες τετράδες. Το πρώτο αντικείμενο αποτελεί τον αριθμό της κάθε τετράδας ο οποίος είναι μοναδικός και ξεκινάει από τον αριθμό 1. Στη συνέχεια τα υπόλοιπα αντικείμενα προσθέτονται στην αντίστοιχη τετράδα. Έτσι θα παραχθούν όλες οι τετράδες του προγράμματος που είναι υπό μεταγλώττιση. Η κλάση είναι η εξής:

```
class Quad:
    def __init__(self, ID, op, x, y, z):
        self.ID = int(ID)
        self.op = str(op)
        self.x = str(x)
        self.y = str(y)
        self.z = str(z)

    def myfunc(self):
        print( self.ID,":"+self.op+","+self.x +","+self.y+","+self.z)

    def metatroph(self):

        return str(self.ID) +" : "+self.op+" , "+self.x +" , "+self.y+" , "+self.z
```

Εικόνα 3: Κλάση Παραγωγής Τετράδων

Όπως παρατηρείται και στην εικόνα, η μορφή μίας τετράδας έχει την μορφή:

$$\boldsymbol{ID: op, x, y, z}$$

Βάσει των παραπάνω ρουτινών μπορούμε να ξεκινήσουμε την μετάφραση σε ενδιάμεσο κώδικα. Για να ξεκινήσει η μετάφραση πρώτα θα πρέπει να γίνουν

μετατροπές στον συντακτικό αναλυτή. Στην ουσία χρησιμοποιούμε τις παραπάνω ρουτίνες για να μετατρέψουμε τις συντακτικές μονάδες, οι οποίες παράγονται από το συντακτικό δένδρο σε τετράδες που συμβολίζουν το αντίστοιχο μεταφρασμένο κομμάτι του κώδικα.

Αρχικά η μετάφραση ξεκινάει από την συνάρτηση ***block(main_program_name)***. Η πρώτη τετράδα που θα δημιουργηθεί από την ρουτίνα ***genquad("begin_block", "main_program_name", "_", "_")*** όπου σηματοδοτεί την έναρξη της μετάφρασης του block του προγράμματος. Όταν τερματίσει η μετάφραση του προγράμματος, τότε θα παράγεται η τετράδα ***genquad("halt", "_", "_", "_")***, στην οποία τερματίζει το block. Αφού μεταφραστεί όλο το block, τότε στο τέλος πρέπει να παραχθεί η τετράδα που σηματοδοτεί την λήξη του block, η οποία γίνεται βάσει της τετράδας ***genquad("end_block", "main_program_name", "_", "_").***

Κατά την διάρκεια της μετάφρασης, γίνεται έλεγχος ύπαρξης κάποιου υποπρογράμματος. Στην περίπτωση που υπάρχει τότε καλείται εκ νέου η ***block(sub_program_name)***, όπου θα δέχεται ως όρισμα το όνομα της συνάρτησης ή της διαδικασίας. Οι τετράδες παραγωγής είναι παρόμοιες με αυτές του κύριου προγράμματος με την μόνη διαφορά πως, οι παράμετροι των συναρτήσεων δεν μεταγλωττίζονται στην δήλωση και επίσης οι τετράδες έναρξης και λήξης ***block*** γίνονται με τις εξής τετράδες ***genquad("begin_block", "sub_program_name", "_", "_")*** και ***genquad("end_block", "sub_program_name", "_", "_")***.

Στη συνέχεια, το επόμενο κομμάτι, το οποίο μεταφράζεται, είναι οι δομές της γλώσσας. Η πρώτη δομή, που θα μελετηθεί, είναι η ***assignmentStat(name_variable)***. Η τετράδα, που δημιουργείται, έχει την μορφή ***genquad(":=", name, "_", name_variable),*** όπου ο τελεστής name είναι η έκφραση η οποία μεταφράζεται και ο τελεστής name_variable είναι το όνομα και παράλληλα το αποτέλεσμα της συγκεκριμένης έκφρασης.

Η επόμενη δομή η οποία εξετάστηκε είναι η ***whileStat()***. Πρώτα απ' όλα ορίζουμε την μεταβλητή ***Bquad*** η οποία κρατάει τον αριθμό της επόμενης τετράδας. Στην συνέχεια, αφού διαβαστεί τι σύμβολο '(', τότε δημιουργούμε τις λίστες B_true και B_false, οι οποίες κρατάνε τις συνθήκες του βρόχου. Επιπλέον, εγγράφουμε τον αριθμό της επόμενης τετράδας στην λίστα S_true με την εντολή ***backpatch(S_true, nextquad())*** που θα μεταβεί εφόσον ισχύει η συνθήκη. Αφού εκτελεστούν τα

statements(), τότε δημιουργείται η τετράδα ***genquad("jump", "_", "_", str(Bquad))***, η οποία δηλώνει το άλμα στην έναρξη του βρόχου. . Τέλος, κάνουμε εγγραφή της επόμενης τετράδας στον πίνακα S_false με την εντολή ***backpatch(S_false, nextquad())***, όπου θα μεταβεί, εφόσον δεν ισχύει η συνθήκη του ***condition()***.

Οι μεταφράσεις των συνθηκών των αντίστοιχων δομών της γλώσσας γίνεται στην αντίστοιχη συνάρτηση ***condition()***. Αρχικά ορίζουμε τις δύο λίστες Q1_true και Q2_false και τις αρχικοποιούμε με τις αντίστοιχες λογικές πράξεις της ***boolterm()***. Οι παραγόμενες τετράδες θα αποθηκεύονται εκ νέου στις λίστες B_true και B_false, οι οποίες κρατάνε τις τετράδες που αληθεύει η συνθήκη ή είναι ψευδή . Όσο η μεταβλητή ***token.tokenString*** διαβάζει τη λογική πράξη 'or', τότε θα αποθηκεύουμε στη λίστα B_false τον αριθμό της επόμενης τετράδας για να μεταβεί, εφόσον δεν ισχύει η συνθήκη. Επιπλέον, αρχικοποιούμε δύο νέες λίστες Q2_true και Q2_false με τις αντίστοιχες λογικές πράξεις της ***boolterm()***. Στη συνέχεια, ενσωματώνουμε στη λίστα B_true τα στοιχεία της λίστας Q2_true καθώς θέλουμε να μεταβούμε σε τετράδες που ισχύουν οι συνθήκες. Τα στοιχεία της λίστας B_false θα ταυτίζονται με αυτά της Q2_false και έτσι θα μπορεί να μεταβεί σε άλλες τετράδες εφόσον δεν ισχύουν οι συνθήκες.

Για να λειτουργήσει η μετάφραση των παραπάνω συνθηκών θα πρέπει αρχικά να μεταφραστούν και οι αντίστοιχες λεκτικές μονάδες που παράγονται από την συνάρτηση ***boolterm()***. Αρχικά ορίζουμε τις λίστες R1_true και R2_false και τις αρχικοποιούμε με την συνάρτηση ***boolfactor()***. Στη συνέχεια ορίζουμε τις λίστες Q_true και Q_false οι οποίες θα αποθηκεύσουν τις αληθής και ψευδείς τετράδες αντίστοιχα. Όσο η μεταβλητή **token.tokenString** διαβάζει την λογική πράξη **'and',** τότε αποθηκεύουμε στην λίστα Q_true τον αριθμό της επόμενης τετράδας με την εντολή ***backpatch(Q_true, nextquad())*** που θα μεταβεί εφόσον ισχύει η συνθήκη. Στη συνέχεια, διαβάζεται η επόμενη λεκτική μονάδα. Εν συνεχεία, ορίζονται οι καινούργιες λίστες R2_true και R2_false και αρχικοποιηούντε από την συνάρτηση ***boolfactor()***. Επιπλέον, ενσωματώνουμε τα στοιχεία της λίστας R2_false στην Q_false, σε περίπτωση που κάποια συνθήκη είναι ψευδής. Τα στοιχεία της Q_true ταυτίζονται με αυτά της λίστας R2_true.

Για να μπορέσουν να μεταφραστούν οι τετράδες με τις αντίστοιχες λογικές πράξεις, θα πρέπει να συνεχίσει η μετάφραση και στην συνάρτηση ***boolfactor()***. Πρώτα

ελέγχουμε εάν η μεταβλητή **token** έχει διαβάσει την λογική πράξη ‘not’. Στην περίπτωση που την έχει διαβάσει τότε προχωράει στην επόμενη λεκτική μονάδα και αποθηκεύει τις συνθήκες στις λίστες B_true και B_false. Εφόσον έχουμε τον παραπάνω λογικό τελεστή, τότε θα αποθηκεύσουμε στην R_true τις τετράδες της B_false και στην R_false τις τετράδες της B_true. Στην περίπτωση που δεν υπάρχει η παραπάνω λογική πράξη, τότε αντίστοιχα εκτελεί τα βήματα όπως προηγουμένως με την μόνη διαφορά ότι οι λίστες τετράδων R_true και R_false παίρνουν τις τετράδες B_true και B_false αντίστοιχα. Στη συνέχεια, αποθηκεύει τις δύο εκφράσεις στις μεταβλητές exp1 και exp2. Επίσης, αποθηκεύει και τον τελεστή συσχέτισης στην μεταβλητή relop. Πρώτα ορίζουμε την λίστα R_true και αποθηκεύουμε τον αριθμό της επόμενη τετράδα στην λίστα. Επιπλέον, παράγεται η τετράδα ***genquad(relop, exp1, exp2, “_”)***, η οποία δείχνει την τετράδα που εκτελείται η λογική συνθήκη. Ύστερα ορίζουμε την λίστα R_false και αποθηκεύουμε τον αριθμό της επόμενης τετράδας. Μετά δημιουργείται η τετράδα ***genquad(“jump”, “_”, “_”, “_”)***, η οποία συμβολίζει το άλμα σε επόμενη τετράδα.

Τώρα θα δούμε πως μεταφράζονται οι εκφράσεις από την συνάρτηση ***expression()***. Αρχικά ορίζουμε μια μεταβλητή num1, η οποία θα κρατάει τον αριθμητικό τελεστή που διάβασε αρχικά. Όσο οι αριθμητικοί τελεστές είναι οι [‘+’, ‘-’], τότε κρατάει το πρόσημο της επόμενης λεκτικής μονάδας το op_sing και δημιουργείται μια νέα μεταβλητή num2, η οποία κρατάει τον αντίστοιχο αριθμητικό τελεστή της επόμενης έκφρασης. Στη συνέχεια δημιουργείται μια προσωρινή μεταβλητή temp στην οποία θα αποθηκεύουμε την πράξη μεταξύ των δύο εκφράσεων num1 και num2. Επιπλέον, παράγεται η τετράδα ***genquad(op_sing, num1, num2, temp)*** και αποθηκεύεται στην μεταβλητή num1 η προσωρινή μεταβλητή temp για να χρησιμοποιηθεί αυτή στις επόμενες τετράδες.

Για να μεταφραστούν οι εκφράσεις της συνάρτησης ***expression()***, θα πρέπει πρώτα να μεταφραστούν οι λεκτικές μονάδες που διαβάζει η συνάρτηση ***term()***. Αρχικά ορίζουμε την μεταβλητή num1 η οποία παίρνει τα αποτελέσματα της συνάρτησης ***factor()***. Όσο διαβάζει η μεταβλητή **token.tokenString** τους αριθμητικούς τελεστές [‘*’, ‘/’], τότε η επόμενη λεκτική μονάδα που θα διαβαστεί θα αποθηκευτεί στην μεταβλητή op_sing. Στη συνέχεια, ορίζουμε την μεταβλητή num2 η οποία παίρνει τα αποτελέσματα της συνάρτησης ***factor()***. Επιπλέον, ορίζουμε την προσωρινή μεταβλητή temp η οποία θα κρατάει το αποτέλεσμα των πράξεων μεταξύ των

μεταβλητών num1 και num2. Τέλος, παράγεται η τετράδα ***genquad(op_sing, num1, num2, temp)***. Εν τέλει, αποθηκεύεται στην μεταβλητή num1 την προσωρινή μεταβλητή temp που κρατάει το αποτέλεσμα της πράξης και επιστρέφεται.

Για να μεταφραστούν οι εκφράσεις, οι οποίες περιέχουν αριθμητικούς, σχεσιακούς και λογικούς τελεστές τότε θα πρέπει να καλεστεί η συνάρτηση ***factor()***. Στην περίπτωση που διαβαστεί αριθμός τότε το αποθηκεύει στην μεταβλητή num και το επιστρέφει. Αν διαβαστεί '(' τότε αποθηκεύει την έκφραση στη μεταβλητή num και την επιστρέφει. Σε περίπτωση που διαβάσει κάποια μεταβλητή, τότε αποθηκεύει το όνομα στη μεταβλητή variable. Έπειτα διαβάζει την επόμενη λεκτική μονάδα και στη συνέχεια καλεί την συνάρτηση ***idtail(variable)***. Αν η συνάρτηση αυτή επιστρέψει true στο flag ***have_func***, τότε θα ξεκινήσει η μεταγλώττιση της κληθείσας συνάρτησης. Επίσης, κρατάει και τις παραμέτρους της συνάρτησης στην λίστα parameters. Αφού εξακριβωθεί ότι υπάρχουν παράμετροι τότε ορίζουμε μια προσωρινή μεταβλητή idret και δημιουργούμε αρχικά την τετράδα ***genquad("par", param[0], param[1], "_")*** η οποία στην θέση 0 έχει το όνομα της και στην θέση 1 το είδος της, ***["cv", "ref"]***. Στη συνέχεια, δημιουργούνται οι τετράδες ***genquad("par", idret, "RET", "_")*** και ***genquad("call", func_name, "_", "_")***, όπου η προσωρινή μεταβλητή idret θα έχει το αποτέλεσμα της συνάρτησης που καλέστηκε και επιστρέφεται. Σε περίπτωση που το παραπάνω flag είναι false, τότε επιστρέφεται η μεταβλητή που διαβάστηκε.

Η επόμενη δομή είναι η ***printStat()***. Η παραγόμενη τετράδα είναι η ***genquad("out", exp, "_", "_")***, η οποία τυπώνει την έκφραση exp ως έξοδο.

Η επόμενη δομή είναι η ***returnStat()***. Η παραγόμενη τετράδα είναι η ***genquad("retv", exp, "_", "_")***, η οποία επιστρέφει την έκφραση exp.

Η επόμενη δομή είναι η ***inputStat()***. Η παραγόμενη τετράδα είναι η ***genquad("inp", idplace, "_", "_")***, η οποία εισάγει τιμή στη μεταβλητή idplace.

Η επόμενη δομή που μελετάμε είναι η συνάρτηση ***ifStat()*.** Πρώτα απ' όλα ορίζουμε μία λίστα b_true και μία λίστα b_false αντίστοιχα. Αυτές οι λίστες παίρνουν τις συνθήκες οι οποίες είναι έτοιμες να αναπαραχθούν. Όταν διαβαστεί το σύμβολο ')' τότε καλούμε την ρουτίνα ***backpatch(b_true, nextquad)*** όπου θα εγγράψει τον επόμενο αριθμό της παραγόμενης τετράδας, για να μεταβεί από τις αντίστοιχες συνθήκες στην τετράδα. Στη συνέχεια, καλούνται τα ***statements()*** όπου θα παραχθούν οι αντίστοιχες τετράδες. Το πρώτο πράγμα που συμβαίνει είναι η δημιουργία της λίστας **ifList**, η

οποία κρατάει τον επόμενο αριθμό της παραγόμενης τετράδας και δημιουργείται από την εντολή ***makelist(nextquad()).*** Στη συνέχεια παράγεται η τετράδα ***genquad("jump", "_", "_", "_")*** η οποία δηλώνει το άλμα σε επόμενη τετράδα. Ύστερα, αποθηκεύουμε στη λίστα b_false τον επόμενο αριθμό της παραγόμενης τετράδας με την εντολή ***backpatch(b_false, nextquad())*** στην οποία θα μεταβεί, εφόσον δεν ισχύουν οι συνθήκες. Τέλος αφού εκτελεστεί και το κομμάτι του elsepart(), τότε θα αποθηκεύσουμε στην λίστα ifList τον επόμενο αριθμό της παραγόμενης τετράδας όπου σηματοδοτεί το τέλος της μεταγλώττισης της δομής αυτής.

Η επόμενη δομή που είναι για μεταγλώττιση είναι η ***incaseStat()***. Αρχικά, κρατάμε μία προσωρινή μεταβλητή w και στο p1Quad τον αριθμό της επόμενης τετράδας. Στη συνέχεια, παράγεται η τετράδα ***genquad(":=", "1", "_", w)*** η οποία ενσωματώνει αυτή την τιμή στην μεταβλητή. Στη συνέχεια, δημιουργούνται δύο λίστες cond_true και cond_false η οποίες έχουν τις τετράδες των συνθηκών. Επιπλέον, στην περίπτωση που ισχύει η συνθήκη, τότε θα γίνει ***backpatch(cond_true,nextquad())*** για να μεταβούμε στην επόμενη τετράδα και εν συνεχεία τίθεται η προσωρινή μεταβλητή ίση με 0 με την εντολή ***genquad(":=", 0, "_", w)***. Στη συνέχεια, αφού εκτελεστούν τα statements(), κάνουμε ***backpatch(cond_false,nextquad())***, όπου θα μεταβουμε σε αυτή την τετράδα, εφόσον δεν ισχύσει η συνθήκη. Μόλις, τελειώσουν τα cases, τότε γίνεται έλεγχος της προσωρινής μεταβλητής. Εάν διαπιστωθεί πως έχει την τιμή 0 ,τότε επιστρέφει στην αρχή της δομής και ξανά εκτελείται, σε άλλη περίπτωση σταματάει η μεταγλώττιση της συγκεκριμένης δομής. Ο έλεγχος αυτός γίνεται με την τετράδα ***genquad("=", w, 0, p1Quad)***.

Η επόμενη δομή η οποία εξετάζεται είναι ***forcaseStat().*** Αρχικά ορίζουμε την μεταβλητή p1Quad η οποία κρατάει το τον αριθμό της επόμενης τετράδας. Στη συνέχεια, κάθε φορά που διαβάζεται η λεκτική μονάδα ***case***, τότε έχουμε τις συνθήκες οι οποίες θα αποθηκευτούν στις λίστες cond_true και cond_false. Αρχικά, αποθηκεύουμε στην λίστα cond_true τον αριθμό της επόμενης τετράδας με την εντολή ***backpatch(cond_true, nextquad())*** όπου θα μεταβεί εφόσον ισχύει η συνθήκη. Επιπλέον, αφού εκτελεστούν τα ***statements()***, παράγεται η τετράδα ***genquad("jump", "_", "_", p1Quad)***, η οποία υποδηλώνει το άλμα αρχική τετράδα της δομής. Έτσι θα φτιαχτεί ο βρόχος της δομής. Εν συνεχεία, ενσωματώνουμε στην λίστα cond_false την επόμενη τετράδα που θα πάει εφόσον δεν ισχύει η συνθήκη. Όταν δεν ισχύει καμία

συνθήκη, τότε απλά μεταβαίνει στις τετράδες που παράγονται μετά την δεσμευμένη λέξη ***default***.

Η επόμενη δομή που θα μελετηθεί είναι ***switchcaseStat()***. Αρχικά, δημιουργείται μία κενή λίστα τετράδων ***exitlist*** με την εντολή ***emptylist()***. Στη συνέχεια, αφού διαβαστεί η λεκτική μονάδα ***case***, τότε αρχικοποιούμε τις λίστες συνθηκών cond_true και cond_false, οι οποίες θα κρατάνε στις συνθήκες που είναι είτε αληθείς είτε ψευδείς, αντίστοιχα. Σε περίπτωση που αληθεύουν, τότε ενσωματώνουμε στην cond_true τον αριθμό της τετράδας που πρέπει να μεταβεί με την εντολή ***backpatch(cond_true,nextquad())***. Εν συνεχεία εκτελούνται τα statements() και παράγονται οι αντίστοιχες τετράδες. Αφού παραχθούν, δημιουργείται μία νέα λίστα ***e*** που περιέχει τις επόμενες τετράδες με την εντολή ***makelist(nextquad())***. Έπειτα δημιουργείται η τετράδα ***genquad("jump", "_", "_", "_")*** η οποία δηλώνει άλμα σε κάποια τετράδα. Ύστερα γίνεται συγχώνευση των λιστών ***exitlist*** και ***e*** με την εντολή ***mergelist(exitlist,e)***. Εφόσον ισχύει η συνθήκη, τότε σηματοδοτεί το τέλος της μεταγλώττισης και βγαίνει από την δομή. Στην περίπτωση που δεν ισχύει κάποια συνθήκη, τότε κάνουμε ***backpatch(cond_false,nextquad())*** για να μεταβεί στην επόμενη τετράδα, μέχρις ότου βρεθεί κάποια αληθής. Αν είναι όλες ψευδείς, τότε θα πάει στην ***default***, θα εκτελεστούν τα statements και ύστερα θα γίνει ***backapatch(exitlist,nextquad())***, όπου θα έχουμε μετάβαση στην επόμενη τετράδα. Στην ουσία η ***exitlist*** κρατά τις τετράδες που επρόκειτο να παραχθούν από την συγκεκριμένη δομή.

Η επόμενη και τελευταία δομή είναι της ***callStat()***. Η δομή αυτή καλείται εφόσον έχει καλεστεί μία διαδικασία στο κύριο πρόγραμμα. Για την μεταγλώττιση θα χρειαστούμε τις παραμέτρους που παίρνει και το όνομα της. Το όνομα της το αποθηκεύουμε σε μία μεταβλητή ***call_name*** που διαβάζεται από τον λεκτικό αναλυτή. Στην συνέχεια, έχουμε την λίστα parameters η οποία καλεί την συνάρτηση ***actualparlist(call_name)*** για να αποθηκευτούν οι παράμετροι και το είδος τους. Εφόσον, επιστραφεί η λίστα, τότε την χρησιμοποιούμε για την μεταγλώττιση. Αρχικά παράγονται οι τετράδες των παραμέτρων με την εντολή ***genquad("par", param[0], param[1], "_")*** όπου το ***param[0]*** θα έχει τα ονόματα των παραμέτρων και το ***param[1]*** το είδος τους, ***["cv", "ref"]***. Τέλος, παράγεται και η τετράδα ***genquad("call", call_name, "_", "_")***, όπου θα καλείται το όνομα της διαδικασίας.

Για την παραγωγή των τετράδων έχουμε κάποιες βοηθητικές μεταβλητές οι οποίες καθορίζουν τα αντίστοιχα block του κώδικα ως προς μεταγλώττιση. Δηλαδή, εάν είναι το κύριο μέρος ενός προγράμματος ή κάποια συνάρτηση ή διαδικασία. Στον παρακάτω πίνακα φαίνονται οι συγκεκριμένες μεταβλητές:

Βοηθητικές Μεταβλητές	Περιγραφή
counter_next_quad	Είναι ο μετρητής των τετράδων και κάθε φορά καλείται από την ***nextquad()***, παράγει τον αριθμό της επόμενης τετράδας
quadList	Είναι μία λίστα που περιέχει τις τετράδες από κάθε μεταγλώττιση γραμμής. Κατά την μεταγλώττιση καλείται η κλάση ***Quad*** η οποία θα δημιουργήσει την πεντάδα των αντικειμένων και τα στοιχεία της αποθηκεύονται στην λίστα μέχρι να τελειώσει η μετάφραση
main_program_name	Κρατάει το όνομα του προγράμματος που επρόκειτο να μεταγλωττιστεί
have_sub_program	Είναι μία ***boolen*** μεταβλητή η οποία είναι ***true*** σε περίπτωση ύπαρξης κάποιας μεταβλητής ή διαδικασίας και ***false*** σε περίπτωση που δεν υπάρχει κάτι τέτοιο. Σε περίπτωση ***false***, μπορεί να παραχθεί το αρχείο σε ισοδύναμη γλώσσα ***C*** το οποίο θα περιέχει τις τετράδες του ενδιάμεσου κώδικα
list_of_variables	Είναι μία λίστα που κρατάει όλες τις μεταβλητές του πηγαίου κώδικα και τις προσωρινές μεταβλητές που εμφανίζονται στον ενδιάμεσο κώδικα. Χρησιμοποιείται για την παραγωγή του ισοδύναμου αρχείου σε γλώσσα ***C***
subprogram_name	Κρατάει το όνομα του εκάστοτε υποπρογράμματος που επρόκειτο να μεταγλωττιστεί

Πίνακας 3:Βοηθητικές Μεταβλητές Ενδιάμεσου κώδικα

Επιπλέον, υπάρχουν και δύο βοηθητικές συναρτήσεις, οι οποίες παράγουν αρχεία ενδιάμεσου κώδικα (***.int***) και ισοδύναμα σε ***C***. Αυτές οι συναρτήσεις είναι η ***intFileGen()*** και ***cFileGen()***. Για την παραγωγή του ισοδύναμου κώδικα ***C***, υλοποιήθηκε και μία έξτρα συνάρτηση ***find_variable()***, η οποία διατρέχει όλη την λίστα ***quadList*** και προσθέτει τις μεταβλητές που βρήκε στην λίστα ***list_of_variables***.

Επίσης, καταγράφονται και οι προσωρινές μεταβλητές στην ίδια λίστα. Οι συναρτήσεις είναι οι εξής:

```
def intFileGen():
    global intFile, quadList

    intFile.write("//Name: Lampros Vlaxopoulos\t AM: 2948\t username: cse52948\n")
    intFile.write("//Name: Georgios Krommydas\t  AM: 3260\t username: cse63260\n\n")

    for i in range(len(quadList)):
        intFile.write(str(quadList[i].metatroph()))
        intFile.write('\n')
```

Εικόνα 4: Συνάρτηση Παραγωγής Αρχείων .int

```
def find_variable():

    global list_of_variables
    for q in quadList:
        if (str(q.op) in ("+", "-", "*", "/", ":=", "<", ">", ">=", "<=", "<>", "=")):
            if ((not (q.z in list_of_variables )) and (str(q.z) != "_") and (not str(q.z).isdigit())):
                list_of_variables.append(str(q.z))
```

```
list_of_variables.append("T_" + str(T_i))
```

Εικόνα 5: Εισαγωγή μεταβλητών list_of_variables

```
def cFileGen():
    global cFile,list_of_variables

    cFile.write("//Name: Lampros Vlaxopoulos\t AM: 2948\t username: cse52948\n")
    cFile.write("//Name: Georgios Krommydas\t  AM: 3260\t username: cse63260\n\n")
    cFile.write("#include <stdio.h>\n\n")
    cFile.write("int main(){\n")
    cFile.write("\tint ")
    find_variable()

    for i in range(len(list_of_variables)):
        cFile.write(str(list_of_variables[i]))
        if (len(list_of_variables) == i+1):
            cFile.write(";\n\n")
        else:
            cFile.write(", ")
    for q in quadList:
        if (q.op == "begin_block"):
            cFile.write("\tL_"+str(q.ID)+ ": ")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op == ":="):
            cFile.write("\tL_"+str(q.ID) + ": " + str(q.z) + " = " + str(q.x) + "; ")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op in('+','-','*','/')):
            cFile.write("\tL_"+str(q.ID) + ": " + str(q.z) + " = " + str(q.x) + q.op + str(q.y) + "; ")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op == "jump"):
            cFile.write("\tL_"+str(q.ID) + ": " + "goto L_" + str(q.z) +"; ")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op in ('=', '<>', '<', '<=', '>', '>=')):
            op=q.op
            if op == '=':
                op = '=='
            elif op == '<>':
                op = '!='
            cFile.write("\tL_"+str(q.ID) + ": " + "if (" + str(q.x) + op + str(q.y) + ") goto L_" + str(q.z) + "; ")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op == "out"):
            cFile.write("\tL_"+str(q.ID) + ": " + "printf(\"" + str(q.x) + " = %d\\n\", " + str(q.x) + ");")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op == "inp"):
            cFile.write("\tL_"+str(q.ID) + ": " + "scanf(\"%d" + "\", &" + q.x + ");")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op == "halt"):
            cFile.write("\tL_"+str(q.ID) + ": " + "return 0; ")
            cFile.write("\t//" + q.metatroph() + "\n")
        elif (q.op == "end_block"):
            cFile.write("\tL_"+str(q.ID) + ": {}")
            cFile.write("\t//" + q.metatroph() + "\n")
            cFile.write("} ")
```

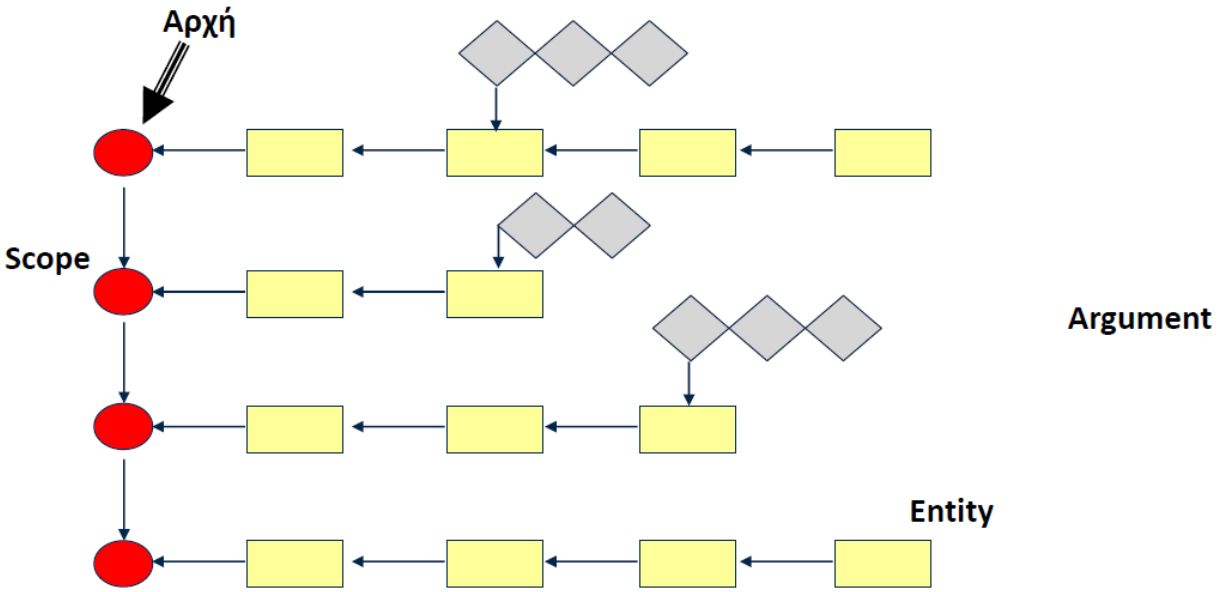
Εικόνα 6: Συνάρτηση Παραγωγής Ισοδύναμου Κώδικα C

2.4 Πίνακας Συμβόλων και Σημασιολογική Ανάλυση

Κατά την μεταγλώττιση ενός πηγαίου κώδικα χρειαζόμαστε, εκτός από την ορθή σύνταξη των λεκτικών μονάδων, και έναν πίνακα συμβόλων ο οποίος αποθηκεύει όλες τις μεταβλητές που χρησιμοποιήθηκαν και τις συναρτήσεις του προγράμματος. Επίσης, χρειαζόμαστε και κάποιους επιπλέον σημασιολογικούς κανόνες να ακολουθήσουμε για την ορθή και σωστή μεταγλώττιση των πηγαίων προγραμμάτων σε τελικό κώδικα.

2.4.1 Πίνακας Συμβόλων

Ο πίνακας συμβόλων αποτελεί μία δομή η οποία αποθηκεύει τις μεταβλητές ενός προγράμματος, είτε είναι αυτές τοπικές είτε καθολικές. Αντίστοιχα εάν έχουμε κάποια συνάρτηση ή διαδικασία, επίσης αποθηκεύεται μαζί με τις αντίστοιχες παραμέτρους της. Γενικά ένας πίνακας συμβολών μπορεί να έχει την παρακάτω μορφή:



Εικόνα 7: Μορφή Πίνακα Συμβόλων

Υπάρχουν γενικά τρεις βασικές έννοιες στον πίνακα. Αυτές είναι η Οντότητα***(Entity)***, το ***Scope*** και το ***Argument***. Αρχικά το ***Scope*** αποτελείται από μία λίστα που περιέχει οντότητες οι οποίες αποθηκεύονται είτε προσωρινά είτε μόνιμα στο εγγράφημα δραστηριοποίησης. Όταν ένα πρόγραμμα που είναι υπό μεταγλώττιση περιέχει συναρτήσεις ή διαδικασίες, τότε το βάθος φωλιάσματος της λίστας αυξάνεται για να αποθηκευτούν προσωρινά οι μεταβλητές του εκάστοτε υποπρογράμματος.

Η επόμενη έννοια είναι η Οντότητα***(Entity)***, η οποία προσδιορίζει το είδος κάποιου εγγραφήματος στον πίνακα συμβολών. Αντίστοιχα, αποθηκεύονται πρώτα οι καθολικές μεταβλητές του προγράμματος στο βάθος φωλιάσματος 0 και στη συνέχεια, αν κληθεί μία συνάρτηση ή διαδικασία, τότε αυξάνεται και το βάθος για τις υπόλοιπες μεταβλητές και παραμέτρους του προγράμματος. Η αποθήκευσης γίνεται προσωρινά στην στοίβα με χρήση ενός δείκτη, του ***stack pointer(sp)***.

Η τρίτη και τελευταία έννοια είναι το ***Argument*** όπου είναι ουσιαστικά μία λίστα που κρατάει τις παραμέτρους μίας συνάρτησης ή διαδικασίας, ανάλογα με την μορφή που περνάει. Δηλαδή εάν είναι με τιμή (***in***) ή με αναφορά (***inout***). Αυτή η λίστα γεμίζει σε περίπτωση που διαβαστεί ένα υποπρόγραμμα. Σε περίπτωση που υπάρχει ένα υποπρόγραμμα μέσα σε ένα υποπρόγραμμα, τότε η μετάφραση του εσωτερικού μπλοκ υποπρογράμματος διαβάζεται σε διαφορετικό βάθος φωλιάσματος με την δικιά του λίστα από παραμέτρους.

Οι παραπάνω έννοιες έχουν υλοποιηθεί σε κλάσεις. Η κάθε κλάση αποτελείται από κάποιες συναρτήσεις οι οποίες θα εξηγηθούν παρακάτω. Αρχικά έχουμε την κλάση ***Entity***, η οποία προσδιορίζει την οντότητα του προγράμματος. Δηλαδή, τις μεταβλητές, τα υποπρογράμματα και τις παραμέτρους. Παρακάτω φαίνονται η συναρτήσεις της γλώσσας:

Συναρτήσεις κλάσης Entity	**Περιγραφή**
__init__(self, name, typeofEntity, offset)	Αρχικοποιεί το αντικείμενο οντότητα με βάσει κάποια πεδία. Αρχικά κρατά το όνομα της οντότητας, τον τύπο της, το ***offset*** που έχει μέσα στην στοίβα(απόσταση από την κορυφή), την αρχική τετράδα που εμφανίζεται(αν πρόκειται για συνάρτηση ή διαδικασία), τον τύπο των παραμέτρων(***parMode***), την λίστα που θα αποθηκευτούν οι παράμετροι, την επόμενη οντότητα που εμφανίζεται, την τύπο μιας μεταβλητής ο οποίος είναι πάντα ***“Int”*** και το

	framelength μιας συνάρτησης ή διαδικασίας
setframelength(self, fl)	Θέτει το ***framelength*** ενός υποπρογράμματος, δηλαδή το μήκος που δεσμεύει ένα υποπρόγραμμα μέσα στην στοίβα
setParMode(self, par)	Θέτει τον τύπο των παραμέτρων ενός υποπρογράμματος
setArgument(self, nextA)	Εισάγει την επόμενη παράμετρο που βλέπει μέσα στην λίστα από τα ***arguments***
setstartQuad(self, starQ)	Θέτει την αρχική τετράδα που ξεκινά ένα υποπρόγραμμα να μεταγλωττίζεται
changeoffset(self, newoffset)	Αλλάζει την θέση μιας οντότητας μέσα στην στοίβα
returnoffset(self)	Επιστρέφει την τωρινή θέση μιας οντότητας μέσα στην στοίβα
returnArgumentList(self)	Επιστρέφει την παραγόμενη λίστα από ***arguments*** που έχουν διαβαστεί μέσα σε ένα πρόγραμμα

Πίνακας 4: Συναρτήσεις Κλάσης Entity

Η επόμενη κλάση που δημιουργήθηκε είναι το ***Scope***. Αυτή η κλάση προσδιορίζει τις οντότητες που εγγράφηκαν στον πίνακα συμβόλων αλλά και το βάθος τους που έγινε η εγγραφή. Παρακάτω φαίνονται οι συναρτήσεις της κλάσης:

Συναρτήσεις Κλάσης Scope	**Περιγραφή**
__init__(self, nestingLevel)	Αρχικοποιεί το αντικείμενο ***Scope*** το οποίο θα περιέχει μία λίστα από τις οντότητες (***listofEntitys***) και το βάθος φωλιάσματος στο οποίο έχει αποθηκευτεί η οντότητα (***nestingLevel***)
addentity(self, entitytoadd)	Προσθέτει μία οντότητα μέσα ***Scope*** και στην λίστα από τις οντότητες

printScope(self)	Τυπώνει τα βάθη φωλιάσματος που υπάρχουν μέσα στον πίνακα συμβόλων
setListOfEntities(self, entLi)	Θέτει την λίστα από τις οντότητες που υπάρχει μέσα στο ***Scope***
getTotalOffset(self)	Επιστρέφει το συνολικό ***offset*** του εγγραφήματος δραστηριοποίησης. Δηλαδή, την απόσταση της τελευταίας οντότητας από την κορυφή της στοίβας
returnListOfEntitys(self)	Επιστρέφει την λίστα με τις οντότητες
varLocal(self, Varname)	Ελέγχει εάν μία μεταβλητή είναι καθολική ή τοπική με το όνομά της ***Varname***

Πίνακας 5: Συναρτήσεις Κλάσης Scope

Η τρίτη και τελευταία κλάση είναι η ***Argument*** η οποία κρατάει πληροφορία για τις παραμέτρους των υποπρογραμμάτων. Παρακάτω φαίνονται οι συναρτήσεις που την υλοποιούν:

Συναρτήσεις Κλάσης Argument	**Περιγραφή**
__init__(self, parMode, typeOfArg)	Αρχικοποιεί το αντικείμενο ***Argument*** το οποίο θα περιέχει ως πεδίο τον τρόπο περάσματος μιας παραμέτρου (***"in"***, ***"inout"***) και τον τύπο της παραμέτρου(***typeOfArg***)
printerArg(self)	Τυπώνει τις παραμέτρους ενός υποπρογράμματος μαζί με τον τύπο τους

Πίνακας 6: Συναρτήσεις Κλάσης Argument

Για την εγγραφή μέσα στον πίνακα συμβόλων χρησιμοποιούμε και κάποιες βοηθητικές μεταβλητές. Αυτές φαίνονται παρακάτω:

Βοηθητικές Μεταβλητές	**Περιγραφή**
scopeList	Είναι μία λίστα που αποθηκεύει προσωρινά όλες τις οντότητες του προγράμματος βάσει του βάθους φωλιάσματος

nesting	Είναι μία μεταβλητή η οποία κρατά το βάθος φωλιάσματος που γίνεται η εγγραφή των οντοτήτων. Αυξάνεται και μειώνεται ανάλογα με το ***block*** που γίνεται ανάγνωση
currentScope	Είναι μία μεταβλητή που μας δείχνει σε ποιο ***Scope*** βρισκόμαστε μέσα στην λίστα
List_after_delete_scope	Είναι μια λίστα που αποθηκεύονται τα ***scopes*** εφόσον έχουν διαγραφεί από τον πίνακα συμβόλων.

Πίνακας 7: Βοηθητικές Μεταβλητές Πίνακα Συμβόλων

Οι παραπάνω συναρτήσεις θα χρησιμοποιηθούν στις συναρτήσεις του συντακτικού αναλυτή, έτσι ώστε να κατασκευαστεί ο πίνακας συμβόλων ενός προγράμματος. Με το που ξεκινά η μεταγλώττιση ενός προγράμματος αρχικοποιείται μέσα στην ***program()*** το αντικείμενο ***Scope*** με αρχικό βάθος φωλιάσματος ***nesting***. Η πρώτη οντότητα που θα αποθηκεύσει στην θέση 8 του εγγραφήματος δραστηριοποίησης είναι το όνομα του κύριου προγράμματος και στην συνέχεια το προσθέτει μέσα στην λίστα ***scopeList*** στην αρχική θέση. Στη συνέχεια, αφού ο συντακτικός αναλυτής έχει αναγνωρίσει τις μεταβλητές που αρχικοποιήθηκαν, τότε τις προσθέτει μέσα στην λίστα των ***Scope*** από τις θέσεις 12 και ύστερα. Κάθε μεταβλητή χρησιμοποιεί 4 byte για την εγγραφή μέσα στην στοίβα και ως οντότητα έχει τύπο ***var***. Επίσης, αυτές οι μεταβλητές είναι καθολικές μεταβλητές ενός προγράμματος.

Το επόμενο βήμα είναι ο έλεγχος ύπαρξης υποπρογραμμάτων. Σε περίπτωση που υπάρχει κάποιο, τότε γίνεται η εγγραφή ως οντότητα μέσα στο ***Scope***. Αρχικά, ελέγχουμε και τις παραμέτρους του υποπρογράμματος και τις εισάγουμε μέσε στην λίστα των ***arguments***. Στη συνέχεια αυξάνεται το βάθος φωλιάσματος μέσα στον πίνακα συμβόλων κατά 1. Τώρα ξεκινά η μεταγλώττιση του υποπρογράμματος στο επόμενο βάθος φωλιάσματος. Αρχικά, κρατάμε και τον αριθμό της πρώτης τετράδας του υποπρογράμματος, έτσι ώστε να γίνει εγγραφή στον πίνακα και να υπολογιστεί και το ***framelength*** της συνάρτησης. Κρατάμε το όνομα και τον τύπο του υποπρογράμματος στην θέση 8 της στοίβας. Έπειτα, αποθηκεύουμε στο ***Scope*** τις

παραμέτρους στα αντίστοιχα ***offset*** μαζί με τον τρόπο περάσματος που έχουν. Εάν είναι ***"in"*** τότε έχουμε ***"cv"***, εάν είναι ***"inout"*** τότε έχουμε ***"ref"***.

Επίσης, ελέγχουμε πρώτα εάν έχουν αρχικοποιηθεί μεταβλητές μέσα στο υποπρόγραμμα. Σε περίπτωση που έχουμε, τότε ξεκινά η εισαγωγή των οντοτήτων των μεταβλητών μέσα στην στοίβα. Αυτές οι μεταβλητές είναι τοπικές και έχουν εμβέλεια μόνο μέσα στο υποπρόγραμμα. Οι μεταβλητές αυτές πιάνουν επίσης 4 byte και κατά την εγγραφή το ***offset*** γίνεται συν 4 μέχρις ότου να διαβαστούν όλες οι τοπικές μεταβλητές. Στη συνέχεια ελέγχει για προσωρινές μεταβλητές μέσα στις τετράδες του υποπρογράμματος. Σε περίπτωση που βρει κάποια τότε την προσθέτει και αυτή την οντότητα στην λίστα με τα ***scopes***. Τέλος, αφού γίνει η μεταγλώττιση του υποπρογράμματος, τότε αλλάζει το συνολικό ***offset*** μέσα στην στοίβα και επιστρέφεται το ***framelength*** του υποπρογράμματος. Το επόμενο βήμα είναι να μειώσουμε το βάθος φωλιάσματος κατά 1 και να διαγράψουμε τις οντότητες και τα ***arguments*** του υποπρογράμματος που εγγράφηκαν κατά την μεταγλώττιση. Επιπλέον, κρατάμε τις οντότητας που διαγράφηκαν στην λίστα ***List_after_delete_scope***, για να μπορέσουμε να τυπώσουμε ολόκληρο τον πίνακα συμβόλων.

Αφού τελειώσει η μεταγλώττιση ενός υποπρογράμματος, τότε ξεκινά του κύριου υποπρογράμματος. Από την στιγμή που έχουν δηλωθεί ήδη οι global μεταβλητές στην αρχή, το μόνο που μένει είναι να αποθηκευτούν και οι προσωρινές μεταβλητές εάν υπάρχουν στον πίνακα συμβόλων. Όταν διαβαστούν όλες οι μεταβλητές και του κυρίους προγράμματος μαζί με των υποπρογραμμάτων που καλούνται στο κύριο πρόγραμμα, τότε σταματά η μεταγλώττιση και διαγράφεται και το ***scope*** με τα ***entities***. Κατά την διαγραφή αποθηκεύεται και αυτό το βάθος φωλιάσματος στην λίστα ***List_after_delete_scope***. Έτσι θα πάρουμε τον τελικό πίνακα συμβόλων με όλες τις μεταβλητές που εμφανίζονται στο πρόγραμμα.

Για να τυπωθεί ο συνολικός πίνακας συμβόλων δημιουργήθηκε η συνάρτηση ***printSymbolTable()***, η οποία θα τυπώσει στον οθόνη όλα τα ***scopes*** του προγράμματος, ενώ παράλληλα θα τα αποθηκεύσει σε ένα αρχείο ***.txt***. Για να τα τυπώσουμε θα χρειαστούμε την λίστα ***List_after_delete_scope*** η οποία έχει κρατήσει όλο τον πίνακα συμβόλων. Παρακάτω φαίνεται η συνάρτηση:

```
def printSymbolTable():## kanei print ta scopes

    global List_after_delete_scope,SymbolTableFile


    for xo in List_after_delete_scope:
        print("\n-------------------------------------------------------------------------")
        SymbolTableFile.write("\n-------------------------------------------------------------\n")
        SymbolTableFile.write(xo.printScope()+"\n")
        print("\n=========================================================================\n")

        SymbolTableFile.write("\n=============================================================================\n")
        ls=xo.returnListOfEntitys()
        for l in ls:
            SymbolTableFile.write(str(l.printer())+"\n")
            a=l.returnArgumentList()
            for arg in a:
                SymbolTableFile.write(arg.printerArg()+"\n")
        print("-----------------------------------------\n")
        SymbolTableFile.write("-------------------------------------------------------------\n")
```

Εικόνα 8: Συνάρτηση Τύπωσης Πίνακα Συμβόλων

2.4.2 Σημασιολογική Ανάλυση

Κατά την διάρκεια της μεταγλώττισης, χρειαζόμαστε και κάποιους σημασιολογικούς κανόνες να ελέγχουμε. Η σημασιολογική ανάλυση χωρίζεται σε δύο φάσεις. Αρχικά, ελέγχουμε τις περιπτώσεις κατά την σύνταξη ενός υποπρογράμματος και στην συνέχεια και τον πίνακα συμβόλων.

Ένα υποπρόγραμμα μπορεί να είναι είτε μία συνάρτηση (***function***) είτε μία διαδικασία (***procedure***). Θα πρέπει να γίνεται έλεγχος ένα διαβάζεται τουλάχιστον μία φορά η λεκτική μονάδα ***return***. Στην περίπτωση μίας συνάρτησης, θα πρέπει να ελέγχεται η ύπαρξη της συγκεκριμένης λεκτικής μονάδας. Σε περίπτωση που δεν διαβάζεται από τον λεκτικό αναλυτή η συγκεκριμένη λέξη κατά την μεταγλώττιση, τότε θα πρέπει να εμφανιστεί ένα μήνυμα λάθους. Στην περίπτωση που το υποπρόγραμμα είναι μία διαδικασία, τότε δεν θα πρέπει ο λεκτικός αναλυτής να διαβάζει την λεκτική μονάδα ***return***. Σε αντίθετη περίπτωση, δηλαδή αναγνωριστεί η συγκεκριμένη λεκτική μονάδα, τότε θα πρέπει να εμφανιστεί ένα μήνυμα λάθους. Καθώς, έτσι παραβιάζεται ο συγκεκριμένος σημασιολογικός κανόνας.

Ο σημασιολογικός αναλυτής εφαρμόζεται και στον πίνακα συμβόλων. Αρχικά, ελέγχουμε στο ίδιο βάθος φωλιάσματος εάν εμφανίζεται μεταβλητή, διαδικασία ή συνάρτηση και τον αριθμό εμφανίσεων. Στην περίπτωση που διαβαστεί μία μεταβλητή στο ίδιο βάθος, η οποία έχει ξανά διαβαστεί, τότε θα πρέπει να εμφανίσει μήνυμα λάθους. Διότι, δεν επιτρέπεται η δήλωση της ίδιας μεταβλητής στο ίδιο βάθος φωλιάσματος πάνω από μία φορά. Αντίστοιχα ισχύει και για μία διαδικασία και για μία συνάρτηση. Το επόμενο βήμα, θα πρέπει να γίνεται ο έλεγχος των οντοτήτων αυτών

και την αντίστοιχη λειτουργία τους. Δηλαδή, ο τύπος μιας μεταβλητής θα πρέπει να είναι ή ***"VAR"*** ή ***"PAR"*** ή ***"TMPVAR"*** και κατ' επέκταση να λειτουργεί ως μεταβλητή. Σε άλλη περίπτωση θα εμφανιστεί μήνυμα λάθους για την συγκεκριμένη οντότητα. Αν είναι κάποια συνάρτηση ή διαδικασία, τότε θα πρέπει ο τύπος να είναι ***"FUNC"***. Η λειτουργία του υποπρογράμματος θα πρέπει αντίστοιχα να λειτουργεί έτσι όπως δηλώθηκε στον πίνακα συμβόλων. Σε αντίθετη περίπτωση, θα πρέπει να εμφανιστεί μήνυμα λάθους. Το τελικό βήμα είναι ο έλεγχος των παραμέτρων των συναρτήσεων και των διαδικασιών. Η δήλωση τους γίνεται στην συνάρτηση του συντακτικού αναλυτή ***formalparlist*** και η εκτέλεση τους γίνεται στην συνάρτηση του συντακτικού αναλυτή ***actualparlist***. Θα πρέπει οι παράμετροι που διαβάστηκαν και κλήθηκαν θα πρέπει να είναι ίδιοι. Σε οποιαδήποτε άλλη περίπτωση θα πρέπει εμφανίσει μήνυμα λάθους, καθώς οι δύο αυτές λίστες των παραμέτρων θα πρέπει να ταυτίζονται.

2.5 Παραγωγή Τελικού κώδικα

Το τελευταίο στάδιο της μεταγλώττισης είναι η παραγωγή του τελικού κώδικα, ο οποίος είναι σε γλώσσα ***assembly***. Αποτελεί τον κώδικα προορισμού(*objective code*). Η μεταγλώττιση ξεκινά μετά την παραγωγή του ενδιάμεσου κώδικα και την δήλωση των μεταβλητών και των συναρτήσεων και των διαδικασιών στον πίνακα συμβολών. Χρειαζόμαστε την πληροφορία από τον πίνακα συμβόλων, διότι οι δηλώσεις γίνονται σε διαφορετικά επίπεδα φωλιάσματος και σε συγκεκριμένη θέση στην στοίβα.

Για την μεταγλώττιση του ενδιάμεσου κώδικα σε τελικό, χρειαζόμαστε πρώτα κάποιες βοηθητικές υπορουτίνες για την μεταφορά των δεδομένων στους καταχωρητές.

Βοηθητικές Υπορουτίνες	**Λειτουργία**

Πίνακας 8: Βοηθητικές Υπορουτίνες Τελικού Κώδικα

Μέρος - 3ο: Έλεγχος Ορθής Λειτουργίας: