Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

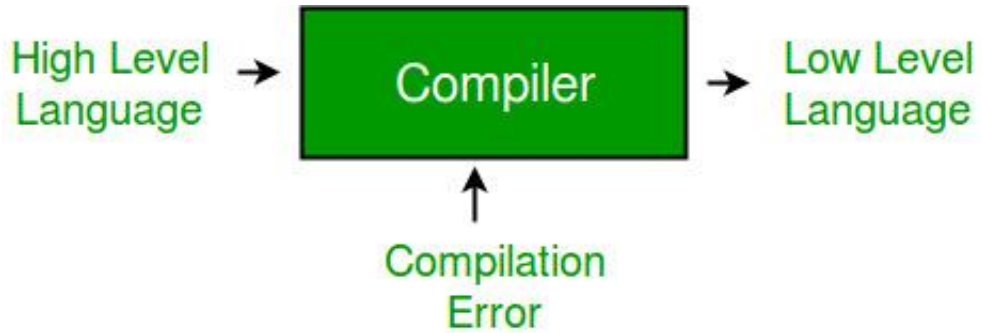
Προπτυχιακό Μάθημα: «Μεταφραστές»

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

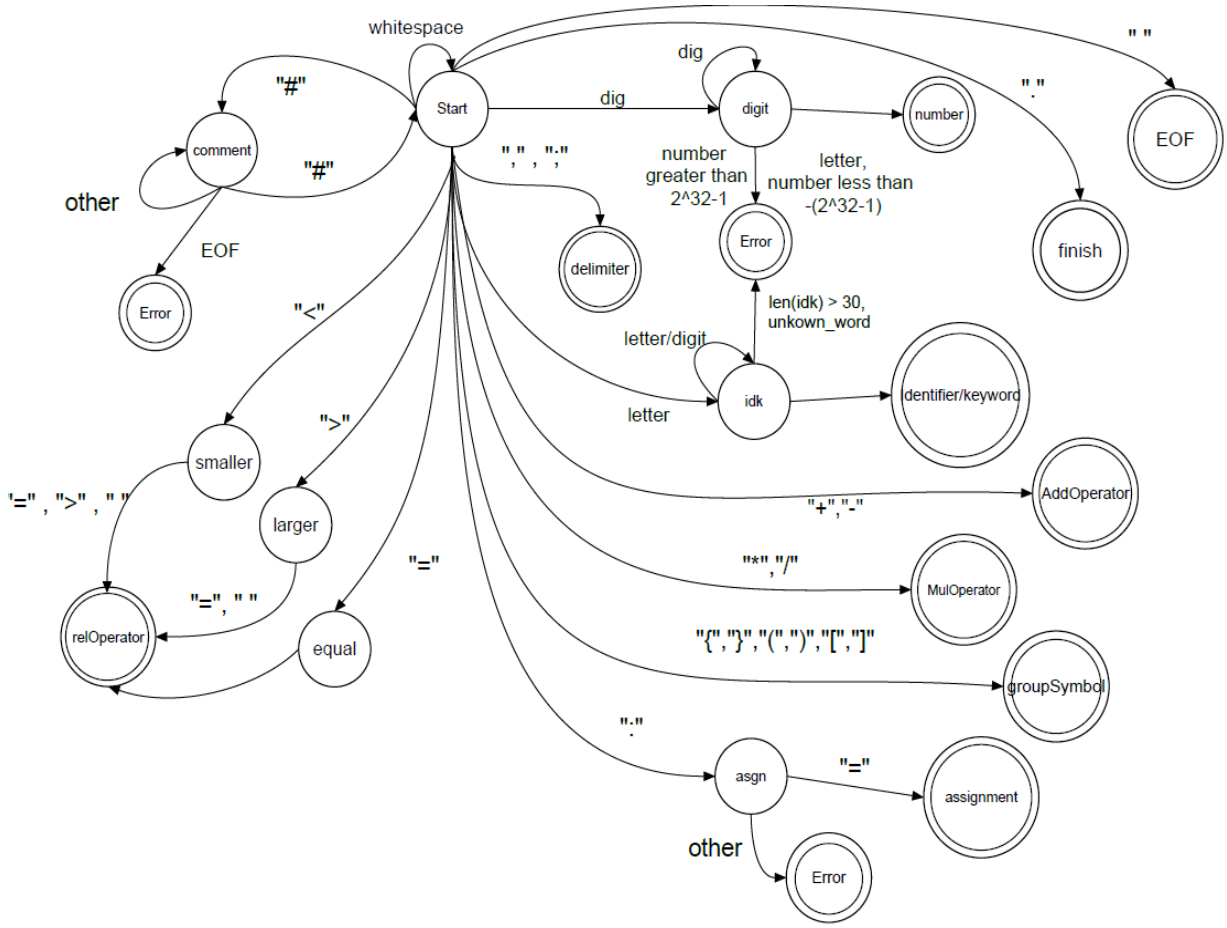
Το πρώτο μέρος ενός μεταγλωττιστή είναι ο Λεκτικός Αναλυτής. Η υλοποίησή του βασίζεται σε ένα μη ντετερμινιστικό αυτόματο πεπερασμένων καταστάσεων. Κάθε κατάσταση αποτελεί κάποιο χαρακτηριστικό της γλώσσας το οποίο ονομάζεται λεκτική μονάδα (**token**). Ένα πρόγραμμα αποτελείται από λεκτικές μονάδες, 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.

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:

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

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("} ")
```

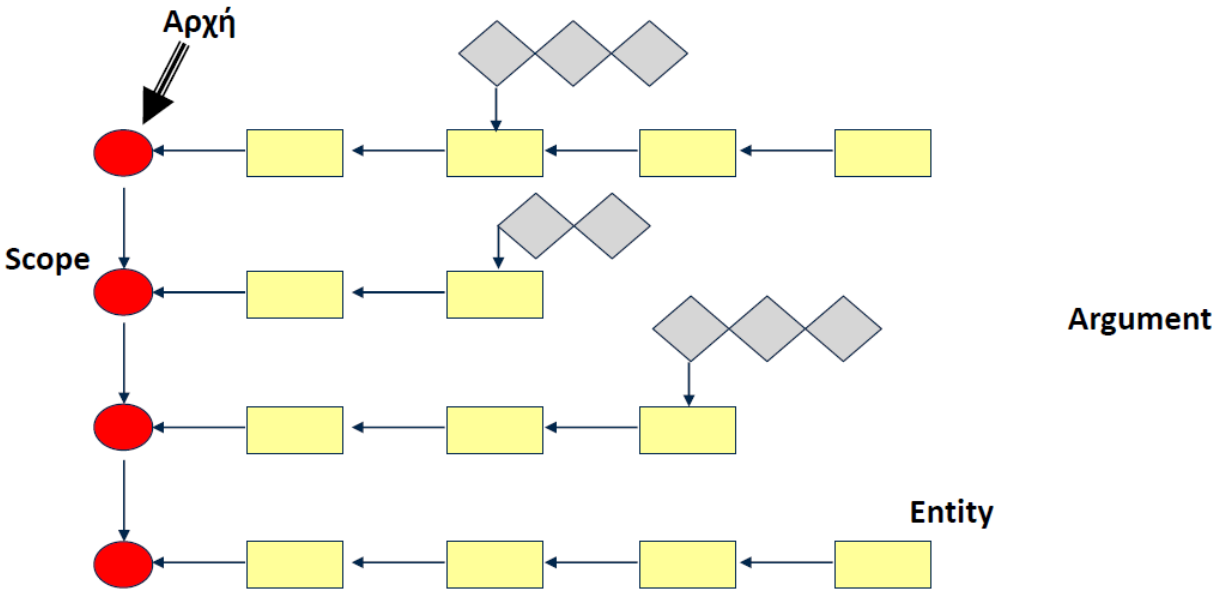
Εικόνα 6: Συνάρτηση Παραγωγής Ισοδύναμου Κώδικα C

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.

Η υλοποίηση των παραπάνω οντοτήτων βασίζεται σε κλάσεις. Αρχικά, η κλάση ***Entity*** περιέχει το είδος της οντότητας και το όνομα. Επίσης, υπάρχουν και κάποιες υποκλάσεις που δηλώνουν το είδος της οντότητας με τα αντίστοιχα πεδία τους. Αυτές οι υποκλάσεις φαίνονται παρακάτω:

Κλάσεις Οντοτήτων	**Περιγραφή**
Entity	Είναι η βασική κλάση δημιουργίας κάποιας οντότητας. Μία οντότητα καθορίζεται από τον τύπο της ***entity_type*** και από το όνομά της ***name***. Ανάλογα με τον τύπο της οντότητας και το όνομα καλείται και η αντίστοιχη κλάση
Variable(Entity)	Σε περίπτωση ανάγνωσης μεταβλητής, καλείται η κλάση δημιουργίας μεταβλητών που κληρονομεί τα αντικείμενα της κεντρικής κλάσης μαζί με τα δικά της αντικείμενα. Μία μεταβλητή έχει τύπο ***"VAR"*** και επιπλέον ένα αντικείμενο ***offset*** το οποίο προσδιορίζει την απόσταση της

	μεταβλητής από την αρχή του εγγραφήματος δραστηριοποίησης
Function(Entity)	
ConstantValue(Entity)	
Parameter(Entity)	
TemporaryVariable(Entity)	

Πίνακας 4: Υλοποίηση Κλάσεων Οντοτήτων

2.4.2 Σημασιολογική Ανάλυση

2.5 Παραγωγή Τελικού κώδικα

Μέρος - 3ο: Έλεγχος Ορθής Λειτουργίας: