Πανεπιστήμιο Ιωαννίνων

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Μάθημα: Μεταφραστές

ΑΠΟΣΤΟΛΑΚΗΣ ΕΥΑΓΓΕΛΟΣ ΑΜ:4318

Εισαγωγη:

Ο μεταγλωττιστής είναι μια εφαρμογή που λαμβάνει ένα αρχείο κώδικα γραμμένο σε μια συγκεκριμένη γλώσσα προγραμματισμού και δημιουργεί ένα αντίστοιχο τελικό αρχείο κώδικα σε διαφορετική γλώσσα προγραμματισμού, όπως το Assembly στην συγκεκριμενη περιπτωση. Κατά την εκτέλεση, μας παρέχει ειδοποιήσεις σχετικά με τυχόν λάθη που υπάρχουν στον αρχικό κώδικα, μαζί με συμβουλευτικές ειδοποιήσεις. Στην περιπτωση μας λοιπον εφτιαξα εναν μεταγλωττιστη απο γλωσσα cpy σε γλωσσα assembly του επεξεργαστη RISC-V.

Αρχικα ανελυσα με βαση τις διαφανειες του μαθηματος την ασκηση σε 5 μερη:

Α) Λεκτικο αναλυτη

Β)Συντακτικό αναλυτή

Γ)Ενδιάμεσο Κώδικα

Δ)Πινακα Συμβόλων

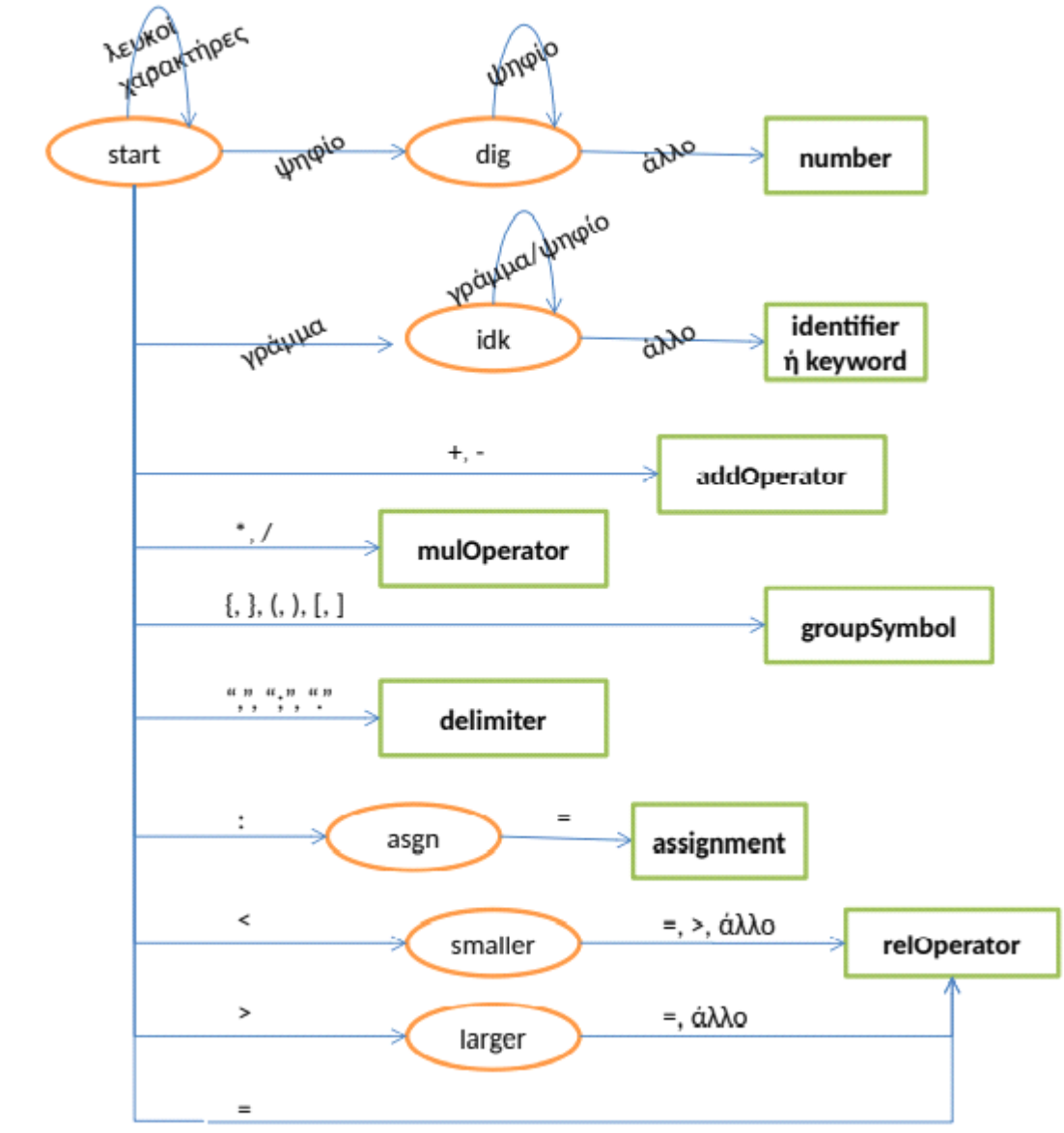
Ε)Τελικό Κώδικα

Αρχικά λοιπον στον Λεκτικό αναλυτη έπρεπε να διαβασει το προγραμμα το αρχειο cpy και να παράξει τις λεκτικές μοναδες τυπου(mod_tk)

```
keyword_tk = 210
number_tk = 211
plus_tk = 212
minus_tk = 213
mult_tk = 214
div_tk = 215
mod_tk = 216
less_than_tk = 217
more_than_tk = 218
equal_tk = 219
excl_tk = 220
comma_tk = 221
colon_tk = 222
left_parentheses_tk = 22
right_parentheses_tk = 2
left_curly_tk = 225
right_curly_tk = 226
less_equal_tk = 227
more_equal_tk = 228
not_equal_tk = 229
hashtag_tk = 230
EOF_tk = 231
change_line_tk = 232
assign_tk = 233
```

```
reserved_words = ['main' , 'def'
 'return' , 'input' , 'int' , 'a

main_tk = 110
def_tk = 111
hash_def_tk = 112
hash_int_tk = 113
global_tk = 114
if_tk = 115
elif_tk = 116
else_tk = 117
while_tk = 118
print_tk = 119
return_tk = 120
input_tk = 121
int_tk = 122
and_tk = 123
or_tk = 124
not_tk = 125
```

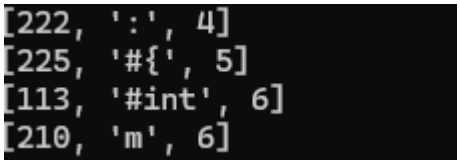
Αυτο βασίστικε στο αυτόματο που μας παρείχατε:



Σχήμα 4.2: Το αυτόματο λειτουργίας του λεκτικού αναλυτή.

Με την κληση το λεκτικου αναλυτη το αυτοματο αρχικοποιειται στην κατασταση start. Αν στην είσοδο εμφανιστει καποιο γραμμα τοτε το αυτόματο μεταβαινει στην κατασταση idk, η οποια ειναι μη τελικη.

Οσο στην εισοδο εμφανιζονται γραμματα ή αριθμους, τοτε για καθε τετοια εισοδο γινετε μεταβαση αλλα στην ιδια idk κατασταση. Απο εκει θα φυγει μολλις ερθει διαφορετικη εισοδος, οχι γραμμα ή αριθμος δλδ. Οταν γινει αυτο το αυτοματο μεταβαινει στην κατασταση keyword. Οταν γινει αυτο ο λεκτικος αναλυτης ελενχει αν το string που αναγνωρισε ειναι keyword ή δεσμευμενη λεξη και επιστρεφει στον συντακτικο αναλυτη το token του keyword ή της λεξης , το ιδιο το keyword σε μορφη string, και την γραμμη που το διαβασε. Λιγο διαφοροποιημενο απο αυτο του βιβλιου γιατι ηθελα να υπαρχει ολοκληρη η μοναδα για debugging.



Το αλφάβητο της cpy αποτελείται από:

• τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου («A»,…,«Z» και «a»,…,«z»),

• τα αριθμητικά ψηφία («0»,…,«9»),

• τα σύμβολα των αριθμητικών πράξεων («+», «-», «*», «//», «%»),

• τους τελεστές συσχέτισης «<», «>», «==», «<=», «>=», «!=»,

• το σύμβολο ανάθεσης «=»,

• τους διαχωριστές («,», «:»)

• καθώς και τα σύμβολα ομαδοποίησης («(»,«)», «#{»,«#}»)

• και σχολίων («##»).

Και οι δεσμευμένες λέξεις της γλώσσας είναι οι ακόλουθες:

main, def, #def, #int, global, if, elif, else, while, print, return, input, int, and, or, not

Το αυτοματο μεταφραζεται σε κωδικα με τη δημιουργια των απαραιτητων token και ενος πινακα δυο διαστασων με ονομα SituationBoard

```
SituationBoard = [
    #start
        [situation_start , situation_letter , situation_number , plus_tk , minus_tk , mult_tk , situation_slash , mod_t
         left_parentheses_tk , right_parentheses_tk , ERROR_PLAIN_LEFT_CURLY , ERROR_PLAIN_RIGHT_CURLY , situation_hash

    #letter
        [keyword_tk , situation_letter , situation_letter , keyword_tk , keyword_tk , keyword_tk , keyword_tk , keyword
         keyword_tk, keyword_tk , keyword_tk , keyword_tk , ERROR_UNIDENTIFIED_CHAR , keyword_tk ,keyword_tk],

    #number
        [number_tk , ERROR_LETTER_AFTER_NUM , situation_number , number_tk , number_tk , number_tk , number_tk , number
         number_tk , number_tk , number_tk , ERROR_UNIDENTIFIED_CHAR , number_tk , number_tk],
```

(variable) situation_letter: Literal[1]

Ο πινακας περιεχει επιλογες καταστασεων αναλογα τον χαρακτηρα που διαβαζει, επιλεγει μια κατασταση και προχοραει στην επομενη, επεισης περιεχει και καταστασεις σφαλματος.

Οταν αναγνωριστει μια τελικη κατασταση ο κωδικας προσθετει στη λιστα LexResult τα παραπανω τρια στοιχεια.

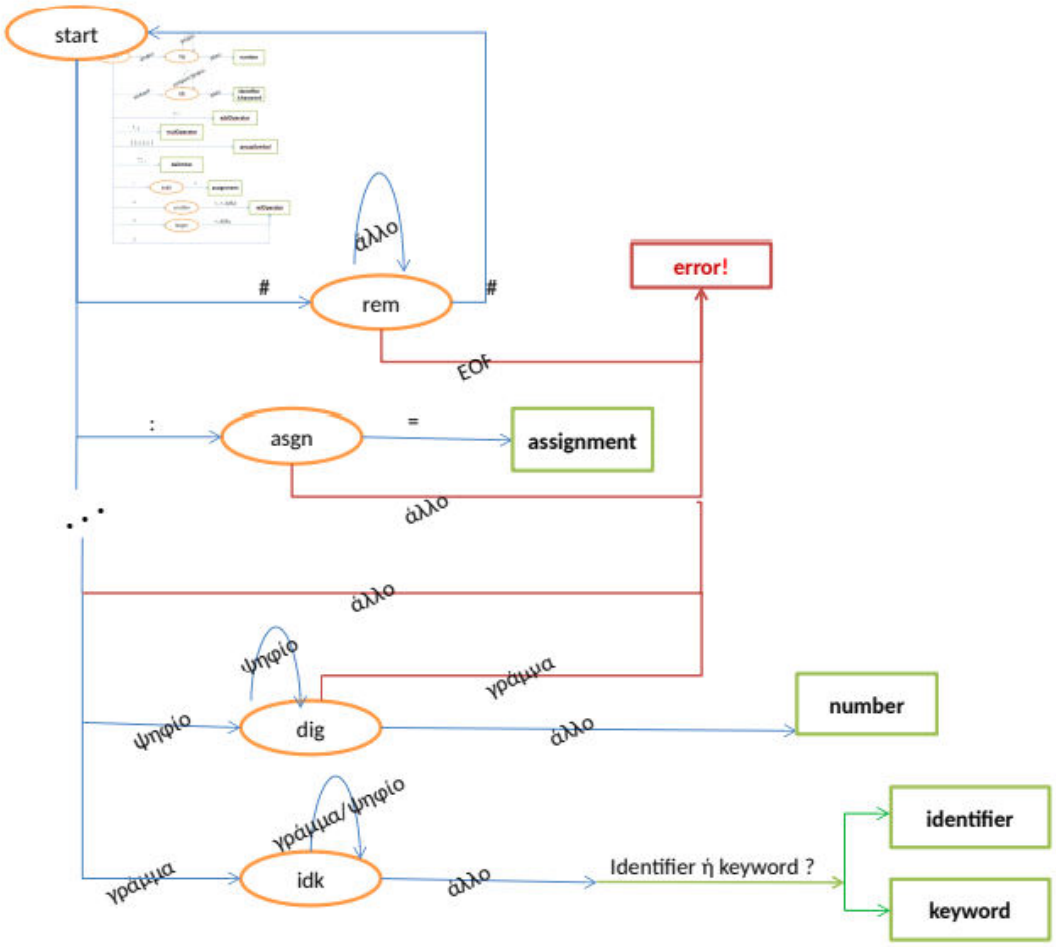
Η κληση του πινακα γινετε επιτοπου με καθε αναγνωρηση στοιχειου με την κληση

```
cr=SituationBoard[cr][char_tk]
```

Οπου cr η τωρινη κατασταση η οποια αρχικοποιηται σε situation_start και char_tk το εκαστοτε token

```
while(cr>=0 and cr<=10):
        char = file.read(1)  #rea


        if (char == ' '):
                char_tk = blank
        elif (char in abc):
                char_tk = letter
        elif (char in numbers):
                char_tk = number
```

επειτα εχουμε τη διαχειρηση σφαλματων οπου ελενχονται τα ορια που θεσατε στην εκφωνηση με βαση το αυτοματο



Σχήμα 4.4: Διαχείριση σφαλμάτων από τον λεκτικό αναλυτή.

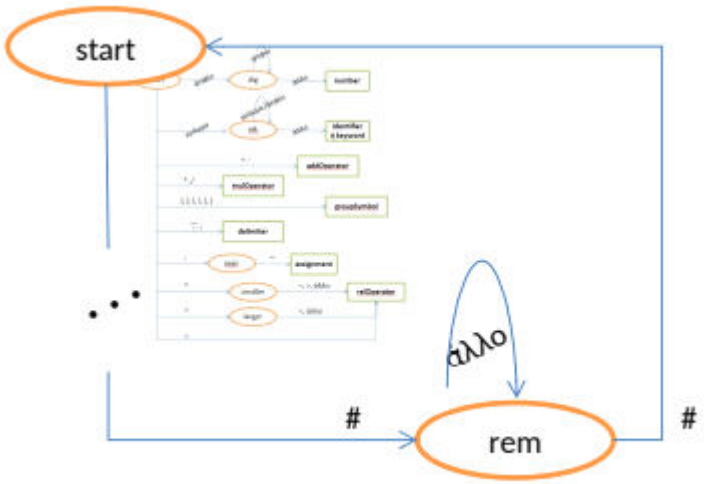
```
if (cr == number_tk):
        if (int(read_word) >= 32767):
            cr = ERROR_OUT_OF_BOUNDS
```

```
if(len(read_word)<30):
        if(cr != situation_start and cr != situation_comment and cr != situation_closing_comment):
            read_word+=char
        else:
            read_word=''

else:
        cr=ERROR_LIMIT_30
```

και τα καταλληλα error αναλογα με την καταταση που οδηγησε ο πινακας μας

```
 if(cr == ERROR_UNIDENTIFIED_CHAR):
        print("ERROR: This symbol cannot be used")
```

Σημαντικο κομματι ειναι και η διαχειρηση των σχολιων η οποια γινετε συμφωνα με το αυτοματο εντος του πινακα μας



Σχήμα 4.3: Διαχείριση σχολίων από τον λεκτικό αναλυτή.

```
#comments
    [situation_comment , situation_comment , situation_comment , situation_comment, situation_comment , situation_comment , situation_comment , situation_comment , situation_co
     situation_comment , situation_comment , situation_comment , situation_comment, situation_comment , situation_comment , situation_closing_comment , ERROR_UNIDENTIFIED_CHAR
```

ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

Σχήμα 5.1: Ο συντακτικός αναλυτής στη διαδικασία της μεταγλώττισης.

Στη συγκεκριμενη ασκηση ενα δυσκολο κομματι ηταν η δημιουργεια της γραμματικης του προγραμματος. Βασιστικα αρκετα στην γραμματικη της Cimple που εχετε στο βιβλιο σας αλλα και στην γραμματικη της περσινης ασκησης που αναρτησατε στο msteams. Για τους περισσοτερους κανονες της γραμματικης δημιουργησα μια ξεχωριστη συναρτηση συμφωνα με τη μεθοδο υης αναδρομικης καταβασης. Σε καθε εισοδο οταν βρισκει ενα μη τερματικο συμβολο καλει την αντιστοιχη συναρτηση. Αν παλι βρει τερματικο συμβολο ελενχει αν τεριαζει με την επομενη μοναδα, αν ταιριαζει προχωραει στην αναγνωση της και συνεχιζει στον κωδικα. Αν ομως δεν ταιριαζει και η γραμματικη δεν μας δινει το κενο σαν ενναλακτικη, μας εμφανιζει μυνημα σφαλματος και τερματιζει το προγραμμα. Αν παλι απο την

ενναλακτικη του κενου μπορει να προχωρησει, συνεχιζει το προγραμμα.

Θα αναλυσω τον κανονα της if μιας και ηταν μια απο τις βασικες διαφορες που ειχαμε να διαχειριστουμε με την προσθηκη του elif.

```
# if statement
ifStat    →    if ( condition )
                   statements(1)
               elsepart
# else part is optional
elsepart  →    else
                   statements(2)
          |    ε
```

Να σημειωθει εδω οτι στη γραμματικη μας δεν ψαχνουμε για παρενθεση, αλλα απευθειας την ανω κατω τελεια, απλα η δομη παραμενει ιδια.

```
def if_stat():
        global LexRes
        global line
        if(LexRes[0] == if_tk):

                LexRes = lex()
                line = LexRes[2]
                #condition()
                C = condition()
                backpatch(C[0], nextQuad())

                if(LexRes[0] == colon_tk):
                                        LexRes = lex()
                                        line = LexRes[2]

                                        code_blocks()

                                        backpatch(C[1],nextQuad())
                                        ListIf = makeList(nextQuad())
                                        genQuad('jump','_','_',ListIf[0])


                                        while(LexRes[0] == elif_tk):
                                                LexRes = lex()
                                                line = LexRes[2]
                                                #conditon()
                                                C = condition()
                                                if(LexRes[0] == colon_tk):
                                                                        LexRes = lex()
                                                                        line = LexRes[2]

                                                                        code_blocks()

                                                                        backpatch(C[1],nextQuad())
                                                                        genQuad('jump','_','_',ListIf[0])


                                                else:
                                                        print("Missing: ':' after elif",line)
                                                        exit(-1)
                                        if(LexRes[0] == else_tk):
                                                LexRes = lex()
                                                line = LexRes[2]
                                                if(LexRes[0] == colon_tk):
                                                        LexRes = lex()
                                                        line = LexRes[2]

                                                        code_blocks()
                                                        backpatch(C[1],nextQuad())
                                                else:
                                                        print("Missing ':' after else statement")
                                                        exit(-1)
                else:
                        print("Missing ':' after if statement")
                        exit(-1)
        else:
                print("Missing if statement")
                exit(-1)
```

Οπως παρατηρειτε στον παραπανω κωδικα αρχικά ελέγχουμε αν η λεκτική μονάδα που επιστρέφει ο λεκτικός αναλυτής αντιστοιχεί σε αυτή της if. Σε αυτη την περιπτωση καλουμε την συνάρτηση condition() μιας και συναντήσαμε τον αντίστοιχο κανόνα τον. Έπειτα ελέγχουμε αν ακολουθεί η άνω-κάτω τελεία.Επειτα καλουμε τη συναρτηση code_blocks() η δικη μου εκδοχη της statemets.

```
statement  →  assignStat
           |  ifStat
           |  whileStat
           |   ...
           |  inputStat
           |  printStat
           |  ε
```

```
def code_block():
	global LexRes
	global line
	if(LexRes[0] == keyword_tk or LexRes[0] == print_tk or LexRes[0] == return_tk):
		simple_code_block()
	elif(LexRes[0] == if_tk or LexRes[0] == while_tk):
		structured_code_block()
	else:
		print("Keywords Error", line)
		exit(-1)

def code_blocks():
	global LexRes

	code_block()
	while(LexRes[0] == keyword_tk or LexRes[0] == print_tk or LexRes[0] == return_tk or LexRes[0] == if_tk or LexRes[0] == while_tk):
		code_block()
```

Θα παρατηρησετε εδω και την αναφορα στην global_id_table η οποια εχει δημιουργηθει για την διαχηρηση των global.

```
def simple_code_block():
	global LexRes
	global global_id_table
	if(LexRes[0] == keyword_tk):
		identifier = LexRes[1]
		if(identifier in global_id_table):
			LexRes = lex()
			if(LexRes[0] == assign_tk):
				LexRes = lex()
				value = expression()
				global_id_table[identifier] = value
		else:
			assignment_stat()

	elif(LexRes[0] == print_tk):
		print_stat()
	elif(LexRes[0] == return_tk):
		return_stat()
def structured_code_block():
	global LexRes
	if(LexRes[0] == if_tk):
		if_stat()
	elif(LexRes[0] == while_tk):
		while_stat()
```

Μετα ελενχουμε αν η επομενη λεκτικη μοναδα αντιστοιχει σε αυτη της elif και ακολουθουμε την ιδια διεδικασια. Αξιζει να σημειωθει οτι στην elif εχουμε μια while η οποια συνεχιζει να κανει την παραπανς διεδικασια μεχρι να ερθει η λεκτικη μοναδα else και αυτο γιατι δεν υπαρχει καποιος περιορισμος στις εμφανισεις του elif, οποτε το else εχει και ρολο τερματισμου μιας loop. Η else με τη σειρα της ακολουθει την παραπανω πορεια ενω σε καθε ελενχο που κανουμε τσεκαρουμε παραλληλα για σφαλματα και αν υπαρχουν το προγραμμα τερματιζει με το αντιστοιχο output διευκολινοντας ετσι το debugging.

Με την ιδια λογικη εχω φτιαξει και τις υπολοιπες συναρτησεις και εχς μερικα παραδειγματα:

```
def bool_term():
        global LexRes
        global line
        #bool_factor()

        Btrue = emptyList()
        Bfalse = emptyList()

        B1 = bool_factor()

        Btrue = B1[0]
        Bfalse = B1[1]

        while(LexRes[0] == and_tk):
                LexRes = lex()
                line = LexRes[2]
                backpatch(Btrue,nextQuad())
                #bool_factor()

                B2 = bool_factor()

                Bfalse = merge(Bfalse,B2[1])
                Btrue = B2[0]

        return [Btrue,Bfalse]
```

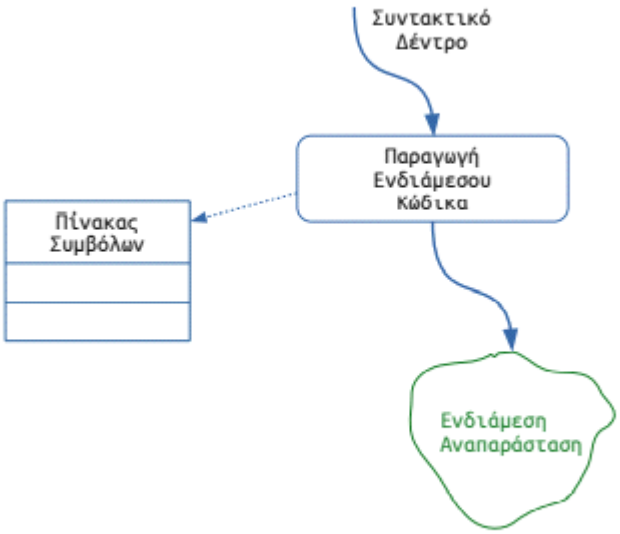
```
def while_stat():
        global LexRes
        global line
        if(LexRes[0] == while_tk):
                LexRes = lex()
                line = LexRes[2]

                Cquad = nextQuad()
                C = condition()
                backpatch(C[0], nextQuad())
                #condition()

                if(LexRes[0] == colon_tk):
                                        LexRes = lex()
                                        line = LexRes[2]
                                        if(LexRes[0] == left_curly_tk):
                                                LexRes = lex()
                                                line = LexRes[2]
                                                code_blocks()

                                                genQuad('jump','_','_',Cquad)
                                                backpatch(C[1],nextQuad())
                                                if(LexRes[0] == right_curly_tk):
                                                    LexRes = lex()
                                                    line = LexRes[2]
                                                else:
                                                    print("Missing '#}' after while statement",line)
                                                    exit(-1)
                                        else:
                                                code_block()

                                                genQuad('jump','_','_',Cquad)
                                                backpatch(C[1],nextQuad())
                else:
                                         print("Missing ':' after while statement",line)
                                         exit(-1)

        else:
                print("Missing while statement",line)
                exit(-1)
```

```
def ADD_OP():
        global LexRes
        global line
        if(LexRes[0] == plus_tk):
                Add_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == minus_tk):
                Add_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        return Add_Op_Symb
```

ΕΝΔΟΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

Το επομενο βημα ηταν η σχεδιαση του ενδοιμασου κωδικα.

ΜΕ



Σχήμα 6.2: Παραγωγή κώδικα.

Σκοπος του ενδοιαμεσου κωδικα ειναι να χωρισει τον κωδικα του προγραμματος σε quads τα οποια διαβαζονται την ετικετας τους, στην περιπτωση μας ενας αριθμος. Η καθε τετραδα εχει εναν τελεστη που καθοριζει και την ενεργεια που θα γινει και 3 τελουμενα. Στην θεση 0 εχουμε την ετικετα στη θεση 1 τον τελεστη και απο 2-4 τα τελουμενα. Τα τελουμενα μπορουνα να ειναι και κενα (_). Ολα τα quads αποθηκεβονται σε μια λιστα, η οποια ειναι και η ουσιαστικη εξοδος του ενδοιαμεσου κωδικα προς τον τελικο.

Οποτε θα αναλυσω τισ συναρτησεις του συντακτικου κωδικα.

Πριν κανω κατι αλλο δημιουργω τη λιστα που θα αποθηκευονται τα quads η οποια προς το παρον τηα ειναι κενη.

```
global listOfAllQuads
listOfAllQuads = []
countQuad = 1
```

Αρχικα η genQuad(op,x,y,z) η οποια παραγει μια τετραδα οπου op ειναι ο τελστης και x,y,z τα 3 τελουμενα.

```
def genQuad(op,x,y,z):
        global countQuad
        global listOfAllQuads
        global FinalQuadsList
        list = []

        list = [nextQuad()]
        list.extend([op, x, y, z])

        countQuad += 1
        listOfAllQuads.append(list)
        FinalQuadsList.append(list)
        return list
T_i = 1
TempVarList = []
```

Μετα η nextQuad() η οποια μας δινει τον αριθμο της επομενης τετραδας

```
def nextQuad():
        global countQuad
        return countQuad

FinalQuadsList = []
```

Μετα ειναι το newTemp() η οποια επιστρεφει το ονομα της επομενης προσωρινης μεταβλητης. (T_2, T_3,...)

```
def newTemp():
        global T_i
        global TempVarList
        temp = 'T_' + str(T_i)
        T_i += 1
        TempVarList.append(temp)

        entity = Entity()
        entity.name = temp
        entity.type = 'TEMP'
        entity.tempVar.offset = compute_offsets()
        new_entity(entity)

        return temp
```

Η emptyList() δημιουργει και επιστρεφει μια νεα κενη λιστα στην οποια στη συνεχεια θα τοποθετηθουν

ετικετες τετραδων.

```
def emptyList():
        return []
```

Η makeList(x) οπου x η ετικετα της τετραδας. Δημειουργει και επιστρεφει μια νεα λιστα η οποια εχει μοναδικο στοιχειο το χ.

```
def makeList(x):
        return [x]
```

Η mergeList(list1, list2) δημιουργει μια λιστα και συνενωνει τις list1 και list2 σε αυτη.

```
def merge(list1, list2):
        return list1 + list2
```

Και η backpatch(list, z) οπου z ειναι το label. Διαβαζει μια μια τις τετραδες που υπαρχουν στην λιστα list και για την τετραδα που αντιστοιχει στο z, συμπληρωνει το τελευταιο της πεδιο με το z. Με δυο loop διατρεχουμε την List και την λιστα ολων των quads. Ελενχουμε αν ειναι η ιδια τετραδα και αν το 5ο στοιχειο της ειναι κενο, αν ειναι τοτε περναμε το z εκει.

```
def backpatch(list, z):
        global listOfAllQuads
        for i in range(len(list)):
                for j in range(len(listOfAllQuads)):
                        if(list[i] == listOfAllQuads[j][0] and listOfAllQuads[j][4] == '_'):
                                listOfAllQuads[j][4] = z
                                break; #didnt enter next i(next loop)
        return
```

Το επομενο βημα ηταν να βαλω στον συντακτικο αναλυτη τις καταλληλες κλησεις ωστε να παραγονται οι τετραδες του ενδοιαμεσου. Αρχικα εβαλα τα begin_block και end_block στην συναρτηση def_function() η οποια διαχειριζεται και το μεγαλυτερο μερος των συναρτησεων. Ετσι στο ανοιγμα μιας εχουμε μετα το συμβολο #{ μια genQuad('begin_block',name,'_','_') και μετα το }# μια genQuad('end_block',name,'_','_').

```
        #line = LexRes[2]
if(LexRes[0] == colon_tk):
                LexRes = lex()
                line = LexRes[2]
                if(LexRes[0] == left_curly_tk):
                        LexRes = lex()
                        line = LexRes[2]
                        new_scope(name)
                        add_parameter()
                        global_id_list(0)
                        declarations()
                        while(LexRes[0] == def_tk):
                                def_function()

                        compute_startQuad()

                        genQuad('begin_block',name,'_','_')
                        code_blocks()
                        compute_frameLength()
                        genQuad('end_block',name,'_','_')

                        final()
                        delete_scope()

                        if(LexRes[0] == right_curly_tk):
                                LexRes = lex()
                                line = LexRes[2]
                        else:
                                print("Missing: '#}'",line)
                                exit(-1)
                        if(LexRes[0] == hash_int_tk or LexRes[0] == global_tk):
                                declarations()
```

Το ιδιο συμβαινει και με την call_main_part(): καθως εχει διαφορετικα αναγνωριστικα, απλα εκει τον operator τον συμπληρωνουμε εμεις.

```
def call_main_part():
        global LexRes
        global line

        if(LexRes[0] == hash_def_tk):
                LexRes = lex()
                line = LexRes[2]
                if(LexRes[0] == main_tk):
                        LexRes = lex()
                        line = LexRes[2]

                        genQuad('begin_block','main','_','_')
                        main_function_call()

                        while(LexRes[0] == keyword_tk):
                                main_function_call()
                        genQuad('halt','_','_','_')
                        genQuad('end_block','main','_','_')

                        final()
                        delete_scope()
                else:
                        print("Missing 'main' call")
                        exit(-1)
        else:
                print("Missing '#def' call")
                exit(-1)
```

Μετα εκανα παρεμβασεις στις συναρτησεις για τις αριθμιτικες πραξεις. Καθενας κανονας επιστρεφει τον κανονα που τον ενεργοποιησε σε μια μεταβλητη ως αποτελεσμα.

```
def MUL_OP():
        global LexRes
        global line
        if(LexRes[0] == mult_tk):
                Mul_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == div_tk):
                Mul_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == mod_tk):
                Mul_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        return Mul_Op_Symb
```

```
def ADD_OP():
        global LexRes
        global line
        if(LexRes[0] == plus_tk):
                Add_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == minus_tk):
                Add_Op_Symb = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        return Add_Op_Symb
```

Επειτα συμπληρωσα και το assignment_stat και το print_stat ωστε να παραγει εισοδο και εξοδο δεδομενων συμφωνα με το 6.6 κεφαλαιο στη σελιδα 117. Αντοιστοιχα για εξοδο χρησημοποιουμε τη μεταβλητη Eplace και για εισοδο thisId.

```
def print_stat():
        global LexRes
        global line
        if(LexRes[0] == print_tk):
                LexRes = lex()
                line = LexRes[2]
                if(LexRes[0] == left_parentheses_tk):
                        LexRes = lex()
                        line = LexRes[2]
                        #expression()
                        Eplace = expression()
                        genQuad('out',Eplace,'_','_')

                        if(LexRes[0] == right_parentheses_tk):
                                LexRes = lex()
                                line = LexRes[2]
                                if(LexRes[0] == EOF_tk):
                                        pass
```

```
def assignment_stat():
        global LexRes
        global line
        if(LexRes[0] == keyword_tk):
                thisId = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
                if(LexRes[0] == assign_tk):
                        LexRes = lex()
                        line = LexRes[2]
                        if(LexRes[0] == int_tk):
                                LexRes = lex()
                                line = LexRes[2]

                                genQuad('inp',thisId,'_','_')

                                if(LexRes[0] == left_parentheses_tk):
                                        LexRes = lex()
```

Και επειτα τροποποιησα αναλογως τις expression, term, factor. Στην γραμμη1283 κραταω το αποτελεσμα της ADD_AP(), φτιαχνω μια νεα προσωρινη μεταβλητη w. Δημιουργω την τετραδα που προσθετει το αποτελεσμα στο T2 και περναω τα αποτεσματα στο T1. Οταν ολοκληρωθει το αποτελεσμα βρισκετε στο T1. Το ιδιο γινετε και στο term().

```
def expression():
        global LexRes
        global line
        optional_sign()
        T1p = term()
        while(LexRes[0] == plus_tk or LexRes[0] == minus_tk):
                Plus_or_Minus = ADD_OP()
                T2p = term()

                w = newTemp()
                genQuad(Plus_or_Minus,T1p,T2p,w)
                T1p = w
        Eplace = T1p
        return Eplace
```

```
def term():
        global LexRes
        global line
        F1p = factor()
        while(LexRes[0] == mult_tk or LexRes[0] == div_tk or LexRes[0] == mod_tk):
                Mul_Or_Div = MUL_OP()
                F2p = factor()

                w = newTemp()
                genQuad(Mul_Or_Div,F1p,F2p,w)
                F1p = w
        Tp = F1p
        return Tp
```

Στην return_stat() συμφωνα με τις διαφανειες τοσο του ενδοιαμεσου οσο και του τελικου καλω την genQuad() ωστε να δημιουργηθει μια τετραδα για την επιστροφη με ονομα retv και μια μεταβλητη με ονομα Eplace.

```
def return_stat():
        global LexRes
        global line
        if(LexRes[0] == return_tk):
                LexRes = lex()
                line = LexRes[2]

                #expression()
                Eplace = expression()
                genQuad('retv',Eplace,'_','_')

        else:
                print("Missing return",line)
                exit(-1)
```

Στην συναρτηση REL_OP() εβαλα μια μεταβλητη να επιστρεφει τον τελεστη συγκρισης

```
def REL_OP():
        global LexRes
        global line
        if(LexRes[0] == equal_tk):
                r = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == less_than_tk):
                r = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == more_than_tk):
                r = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == less_equal_tk):
                r = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == more_equal_tk):
                r = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        elif(LexRes[0] == not_equal_tk):
                r = LexRes[1]
                LexRes = lex()
                line = LexRes[2]
        else:
                print("PLease use one of (=, <, >, <=, >=, !=)")
                exit(-1)
        return r
```

Στην bool_factor εβαλα δυο πινακες Btrue, Bfalse οι οποιοι αποθηκευουν τα αποτελεσματα της condition(). Οταν παρει τα τελικα αποτεσματα δημιουργει τη λιστα Btrue και φτιαχνει την τετραδα. Το ιδιο κανει και η Bfalse αλλα με την τετραδα jump αν δεν ισχθει η συγκριση.

```
def bool_factor():
        global LexRes
        global line

        Btrue = emptyList()
        Bfalse = emptyList()

        if(LexRes[0] == not_tk):
                LexRes = lex()
                line = LexRes[2]
                if(LexRes[0] == left_parentheses_tk):
                        LexRes = lex()
                        line = LexRes[2]
                        #condition()
                        C = condition()
                        if(LexRes[0] == right_curly_tk):
                                LexRes = lex()
                                line = LexRes[2]

                                Btrue = C[1]
                                Bfalse = C[0]

                        else:
                                print("Missing ')' at boolfactor")
                                exit(-1)
                else:
                        print("Missing '(' at boolfactor")
        elif(LexRes[0] == left_parentheses_tk):
                LexRes = lex()
                line = LexRes[2]
                #condition()
                C = condition()

                if(LexRes[0] == right_parentheses_tk):
                        LexRes = lex()
                        line = LexRes[2]

                        Btrue = C[0]
                        Bfalse = C[1]

                else:
                        print("Missing ')' in boolfactor")
                        exit(-1)
        else:
                #expression()
                E1 = expression()
                #REAL_OP()
                R = REL_OP()
                #expression()
                E2 = expression()
```

Μετετρεψα αναλογα και τις bool_term() και condition()

```
def condition():
        global LexRes
        global line

        Btrue = emptyList()
        Bfalse = emptyList()

        B1 = bool_term()

        Btrue = B1[0]
        Bfalse = B1[1]
        #bool_term()
        while(LexRes[0] == or_tk):
                LexRes = lex()
                line = LexRes[2]
                backpatch(Bfalse,nextQuad())
                #bool_term()
                B2 = bool_term()

                Btrue = merge(Btrue,B2[0])
                Bfalse = B2[1]
        return [Btrue,Bfalse]

def bool_term():
        global LexRes
        global line
        #bool_factor()

        Btrue = emptyList()
        Bfalse = emptyList()

        B1 = bool_factor()

        Btrue = B1[0]
        Bfalse = B1[1]

        while(LexRes[0] == and_tk):
                LexRes = lex()
                line = LexRes[2]
                backpatch(Btrue,nextQuad())
                #bool_factor()

                B2 = bool_factor()

                Bfalse = merge(Bfalse,B2[1])
                Btrue = B2[0]

        return [Btrue,Bfalse]
```

Στην συναρτηση if_stat() συμπληρωσα τις τετραδες της backpatch με την επομενη τετραδα, ακομα στο κομματι If και elif δημιουργησα μια τετραδα Jump ωστε αν ισχυει καποιο if, elif να μην μπει στα υπολοιπα.

```
def if_stat():
        global LexRes
        global line
        if(LexRes[0] == if_tk):

                LexRes = lex()
                line = LexRes[2]
                #condition()
                C = condition()
                backpatch(C[0], nextQuad())

                if(LexRes[0] == colon_tk):
                                        LexRes = lex()
                                        line = LexRes[2]

                                        code_blocks()

                                        backpatch(C[1],nextQuad())
                                        ListIf = makeList(nextQuad())
                                        genQuad('jump','_','_',ListIf[0])


                                        while(LexRes[0] == elif_tk):
                                                LexRes = lex()
                                                line = LexRes[2]
                                                #conditon()
                                                C = condition()
                                                if(LexRes[0] == colon_tk):
                                                                        LexRes = lex()
                                                                        line = LexRes[2]

                                                                        code_blocks()
                                                                        ListIf = makeList(nextQuad())
                                                                        backpatch(C[1],nextQuad())
                                                                        genQuad('jump','_','_',ListIf[0])
```

Με την ιδια λογικη στην while_stat(), κραταω την τετραδα που πραγματοποιητε η while και γυρναει σε αυτη με την τετραδα Jump.

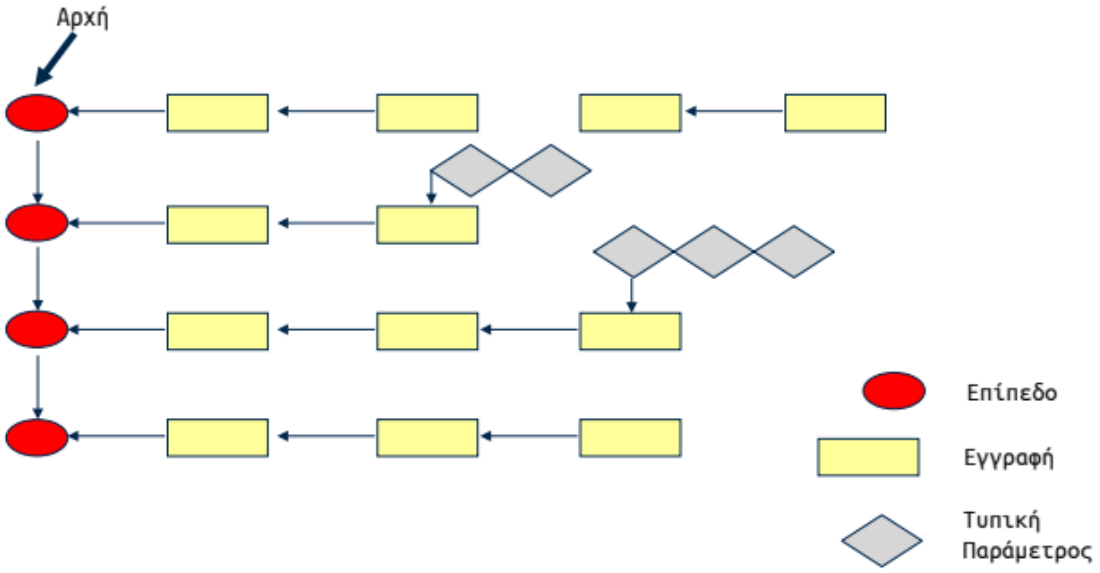
```
def while_stat():
        global LexRes
        global line
        if(LexRes[0] == while_tk):
                LexRes = lex()
                line = LexRes[2]

                Cquad = nextQuad()
                C = condition()
                backpatch(C[0], nextQuad())
                #condition()

                if(LexRes[0] == colon_tk):
                                LexRes = lex()
                                line = LexRes[2]
                                if(LexRes[0] == left_curly_tk):
                                        LexRes = lex()
                                        line = LexRes[2]
                                        code_blocks()

                                        genQuad('jump','_','_',Cquad)
                                        backpatch(C[1],nextQuad())
                                        if(LexRes[0] == right_curly_tk):
                                            LexRes = lex()
                                            line = LexRes[2]
                                        else:
                                            print("Missing '#}' after while statement",line)
                                            exit(-1)
                                else:
                                        code_block()

                                        genQuad('jump','_','_',Cquad)
                                        backpatch(C[1],nextQuad())
```

Τελος με τη συναρτηση intCode(), γραφω σε ενα αρχειο το οποιο ανοιγω παρακατω το οποιο αν υπαρχει σβηνει ολες τις εγγραφες με το w στην εντολη open(το screenshot ειναι του τελικου οπου για debbuging εχω βαλει a οπου κραταει τις εγγραφες), και μεσα σε αυτο τυπωνω την λιστα των τετραδων, περισσοτερο για να ξερω οτι δουλευει.

```
def intCode(intF):
        #this writes the list of quads to the intermediate code file named intFile.int
        for i in range(len(listOfAllQuads)):
                intF.write(str(listOfAllQuads[i][0]) + ' ' + str(listOfAllQuads[i][1]) + ' ' + str(listOfAllQuads[i][2]) + ' ' + str(listOfAllQuads[i][3]) + ' ' + str(listOfAllQuads[i][4]) + '\n')
intF = open('intCodeFile.int','a')
intCode(intF)
intF.close()
listOfAllQuads = []
```

Πινακας Συμβολων

Στον πινακα συμβολων αποθηκευουμε πληροφοριες σχετικα με τα συμβολικά ονοματα που χρησιμοποιουμε σε ενα προγραμμα οπως μεταβλητες, συναρτησεις, διδαδικασιες, παραμετρους κλπ. Ενας πινακας συμβολων οπτικα εχει την παρακατω μορφη:



Σχήμα 8.2: Η δομή του πίνακα συμβόλων.

Εμεις εχουμε να δημειουργησουμε 3 κλασεις την scope()(Επιπεδο) την Entity()(εγγραφη) και την Argument()(τυπικη παραμετρος).

Η Scope() εχει ενα ονομα,μια λιστα με ολες τις εγγραφες που αποθηκευονται στο εγγραφημα δραστηριοτητας και το βαθος φωλιασματος. Για καθε συναρτηση δημιουργειτε νεο Scope() και στο τελος της διαγραφετε.

```
class Scope():
        #Red circle
        def __init__(self, enclosingScope=None):
                self.enclosingScope = enclosingScope
                self.entityList = []
                self.name = ''
                self.nestingLvl = 0
                #self.enclosingScope = None
                #self.entityList = []
```

Η Entity() εχει 4 τυπους, εναν για μεταβλητη (Variable), εναν για υποπρογραμμα(SubProgram), εναν για μια παραμετρους(Parameter) και εναν για προσωρινες μεταβλητες(TempVar).

```
class Entity():
        #yellow box
        def __init__(self, name = None, type = None):
                self.name = name
                self.type = type
                self.variable = self.Variable()
                self.subprogram = self.SubProgram()
                self.parameter = self.Parameter()
                self.tempVar = self.TempVar()

        class Variable():
                def __init__(self):
                        self.type = 'Int'
                        self.offset = 0

        class SubProgram():
                def __init__(self):
                        self.type = 'Function'
                        self.startQuad = 0
                        self.frameLength = 0
                        self.arguments = []
                        self.nestingLvl = 0

        class Parameter():
                def __init__(self):
                        self.mode = 'CV'
                        self.offset = 0

        class TempVar():
                def __init__(self):
                        self.type = 'Int'
                        self.offset = 0
```

Και τριτη η Argument() που εχει ενα ονομα και ειναι τυπου int αφου στη γλωσσα αμς υπαρχουν μονο ακαιραιοι.

```
class Argument():
        #triangle
        def __init__(self):
                self.name = ''
                self.type = 'Int'
```

Μετα ειναι οι συναρτησεις δημειοθυργειας των 3 κλασεων:

```
def new_argument(object):
        global topScope
        topScope.entityList[-1].subprogram.arguments.append(object)

def new_entity(object):
        global topScope
        topScope.entityList.append(object)

topScope = None

def new_scope(name):
        global topScope
        nextScope = Scope()
        nextScope.name = name
        nextScope.enclosingScope = topScope

        if(topScope == None):
                nextScope.nestingLvl = 0
        else:
                nextScope.nestingLvl = topScope.nestingLvl + 1

        topScope = nextScope
```

Να σημειωθει οτι φτιαχνετε μια μεταβλητη topScope το οποιο στη συναρτηση new_scope χρησημοποιω ωστε να ελενχω αν ειναι το κορυφαιο επιπεδο κενο αλλιως να ορισω το σωστο βαθος φολιασματος.

delete_scope(): Χρησιμευει στην διαγραφη ενος επιπεδου οταν τελειωσει η μεταφραση μιας συναρτησης, οριζοντας το επομενο σαν κορυφαιο.

```
def delete_scope():
        global topScope

        freeScope = topScope
        topScope = topScope.enclosingScope
        del freeScope
```

compute_offset(): Την συναρτηση αυτη τη χρησιμοποιοω για να υπολογισω την αποσταση μιας μεταβλητης, παραμετρου ή προσωρινης μεταβλητης. Διατρεχει τον πινακα εγγραφων και για καθε εγγραφη αυξανει τον μετρητη κατα 1. Επειδη εχει μονο ακεραιους αριθμους που καταλαμβανουν 4 bytes ο καθενας, πολλαπλασιαζω τον μετρητη με το 4 και προσθετω 12.

```
def compute_offsets():
        global topScope
        counter = 0
        if topScope.entityList:
                for entity in topScope.entityList:
                        if (entity.type == 'VAR' or entity.type == 'TEMP' or entity.type == 'PARAM'):
                                counter += 1
        offset = (4*counter) + 12

        return offset
```

add_parameters(): Κανει προσθηκη τυπικων παραμετρων μιας συναρτησης

```
def add_parameter():
        global topScope
        for arg in topScope.enclosingScope.entityList[-1].subprogram.arguments:
                entity = Entity()
                entity.type = 'PARAM'
                entity.name = arg.name
                entity.parameter.mode = 'CV'
                entity.parameter.offset = compute_offsets()
                new_entity(entity)
```

Τελος η find_entity(name): Αλλαξα το ονομα γιατι την χρησημοποιησα για debugging αλλα τελικα δουλεψε και επειδη οτι δουλευει δε το χαλαμε εμεινε :) . Με αυτη ψαχνουμε ενα ζευγαρι επιπεδου και εγγραφης,(ή στη δικια μας περιπτωση γιατι η loadvr επερνε σαν εισοδο None και γιατι δεν ειχε γινει η δημειουργεια entity στην ωρα της (mind not the commends here)).

```
def find_entity(name):
        global topScope
        #print(f"Looking for entity: {name}")
        #print(f"Current scope: {topScope}")
        #print(f"All entities in current scope: {[e.name for e in topScope.entityList]}")

        currentScope = topScope
        while currentScope != None:
                for entity in currentScope.entityList:
                        if entity.name == name:
                                return currentScope,entity
                currentScope = currentScope.enclosingScope
        print("Error: Entity not found. Name: " + str(name))
        exit()
```

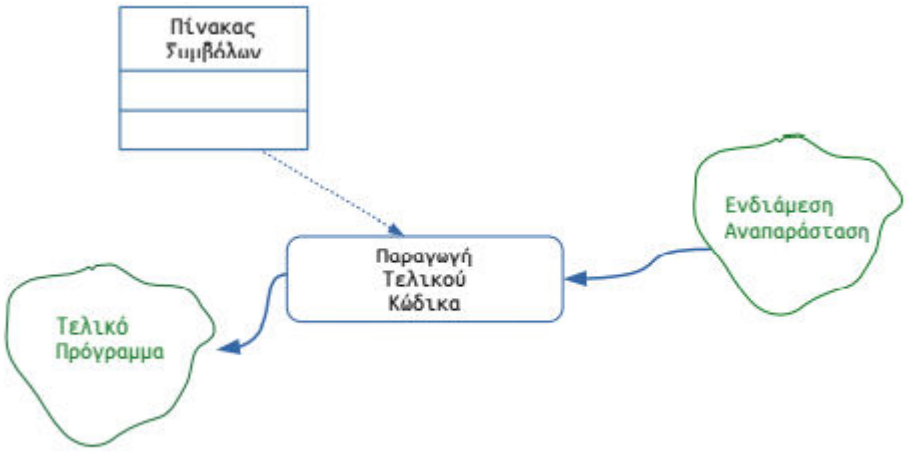
Τελος εγιναν αλλαγες στον συντακτικο αναλυτη ωστε κατα την αναγνωση του προγγραμματος να δημειουργειτε ο πινακας συμβολων. Οπως για παραδειγμα στο startRule() η στην συναρτηση def_functio().

```
def startRule():
        new_scope('main')
        def_main_part()
        call_main_part()
```

```
def def_function():
        global LexRes
        global line
        global topScope
        declarations()
        if(LexRes[0] == def_tk):
                LexRes = lex()
                line = LexRes[2]
                if(LexRes[0] == keyword_tk):
                        name = LexRes[1]
                        LexRes = lex()
                        line = LexRes[2]
                        if(LexRes[0] == left_parentheses_tk):
                                LexRes = lex()
                                line = LexRes[2]

                                newScope = Scope(enclosingScope=topScope)
                                topScope = newScope

                                while True:
                                        print(f'Current LexRes[0]: {LexRes[1]}')  # Debugging print statement
                                        if LexRes[0] == right_parentheses_tk:
                                                LexRes = lex()
                                                line = LexRes[2]
                                                break
                                        elif LexRes[0] == keyword_tk:
                                                newEntity = Entity(name=LexRes[1], type='VAR')
                                                topScope.entityList.append(newEntity)
                                                LexRes = lex()
                                                line = LexRes[2]
                                        elif LexRes[0] == comma_tk:
                                                LexRes = lex()
                                                line = LexRes[2]


                                entity = Entity()
                                entity.name = name
                                entity.type = 'SUBPROGRAM'
                                entity.subprogram.nestingLvl = topScope.nestingLvl + 1
                                entity.subprogram.type = 'Function'
                                new_entity(entity)
```

ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

Το τελευταιο κομματι της εργασιας ειναι ο τελικος κωδικας. Ειναι το σταδιο της μεταγλωττισης οπου παραγεται ο κωδικας σε γλωσσα μηχανης. Αυτο το μερος εγινε αρκετα με βαση τις διαφανειες του msteams αλλα και του βιβλιου που υπαρχει εκει, δεν ξερω αν τα αποτελεσματα ειναι 100% σωστα γιατι δεν εχω καθολου καλη σχεση με την assmbly.



Σχήμα 9.1: Η παραγωγή τελικού κώδικα στη διαδικασία της μεταγλώττισης.

Αρχικα ανοιγω ενα αρχειο asm(Assembler source file) οπου θα γραφονται οι εντολες σε assmbly.

```
ascFile = open('ascFile.asm', 'w')
ascFile.write('        \n\n\n')
```

Αρχικα η gnvlcode()

Βοηθητικές Συναρτήσεις - gnvlcode

- μεταφέρει στον t0 την διεύθυνση μιας **μη τοπικής** μεταβλητής
- από τον **πίνακα συμβόλων** βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από **τον σύνδεσμο προσπέλασης** την εντοπίζει

```
lw t0,-4(sp)                στοίβα του γονέα
όσες φορές χρειαστεί:
        lw t0,-4(t0)        στοίβα του προγόνου που έχει τη μεταβλητή
addi t0,t0,-offset          διεύθυνση της μη τοπικής μεταβλητής
```

Εδω λοιπον στην αρχη μεταφερω το περιεχομενο της στιβας που ειναι στη θεση -4 στον καταχωρητη t0.

Μετα φορτωνει στον ιδιο καταχωρητη τις τοπικες μεταβλητες των προγωνων του καλλοντας τη find_entity() για να βρει το επιπεδο οπου υπαρχει η εγγραφη. Επειτα βρισκει το ποσες φορες θα γινει κατι τετοιο αφαιροντας το βαθος φωλιασματος του γονεα με το βαθος φωλιασματος που αυτη βρηκε. Τελος ελενχει τον τυπο της μεταβλητης και κανει τη καταλληλη εγγραφη.

```
def gnlvcode(var):

        global topScope
        global ascFile
        ascFile.write('lw t0,-4(sp)\n')

        (scope, entity) = find_entity(var)

        level = topScope.nestingLvl - scope.nestingLvl
        level = level-1

        for i in range(0, level):
                ascFile.write('lw t0,-4(t0)\n')

        if entity.type == 'VAR':
                ascFile.write('addi t0,t0,-' + str(entity.variable.offset) + '\n')
        elif entity.type == 'PARAM':
                ascFile.write('addi t0,t0,-' + str(entity.parameter.offset) + '\n')
```

loadvr(v,r)

Βοηθητικές Συναρτήσεις – loadvr

- **μεταφορά δεδομένων στον καταχωρητή r**
- η μεταφορά μπορεί να γίνει από τη μνήμη (στοίβα)
- ή να εκχωρηθεί στο r μία σταθερά
- η σύνταξη της είναι loadvr(v,r)
- διακρίνουμε περιπτώσεις

H

συναρτηση αυτη εχει τις εξης περιπτωσεις

Το v να ειναι σταθερα: Ο ελενχος γινετε με τη χρηση της v.isdigit(). Αν ειναι η τιμη μεταφερεται απευθειας στον καταχωριτη

Εαν ειναι μεταβλητη που εχει δηλωθει σε συναρτηση που εκτελειται τη δεδομενη στιγμη (scope.nestingLvl == topScope.nestingLvl). Ελενχεται ο τυπος της μεταβλητης και γινετε στον καταχωρητη η αναλογη εγγραφη.

```
if scope.nestingLvl == topScope.nestingLvl:
        if entity.type == 'VAR':
                ascFile.write('lw t' + str(r) + ', -' + str(entity.variable.offset) + '(sp)\n')
        elif entity.type == 'PARAM' and entity.parameter.mode == 'CV':
                ascFile.write('lw t' + str(r) + ', -' + str(entity.parameter.offset) + '(sp)\n')
        elif entity.type == 'TEMP':
                ascFile.write('lw t' + str(r) + ', -' + str(entity.tempVar.offset) + '(sp)\n')
```

Εαν ειναι καθολικη μεταβλητη (scope.nestingLvl == 0) γινετε ελενχος του τυπου της και τυπωνεται η καταλληλη εγγραφη.

```
    elif scope.nestingLvl == 0 and entity.type == 'VAR':
            ascFile.write('lw t' + str(r) + ', -' + str(entity.variable.offset) + '(s0)\n')

    elif scope.nestingLvl == 0 and entity.type == 'TEMP':
            ascFile.write('lw t' + str(r) + ',-' + str(entity.tempVar.offset) + '(s0)\n')
```

Εαν ειναι μεταβλητη που εχει δηλωθει σε καποιο προγονο (scope.nestingLvl < topScope.nestingLvl) γινετε και παλι ελενχος και τυπονεται η καταλληλη εγγραφη.

```
    elif scope.nestingLvl < topScope.nestingLvl:
            if entity.type == 'VAR':
                    gnlvcode(v)
                    ascFile.write('lw t' + str(r) + ', 0(t0)\n')
            elif entity.type == 'PARAM' and entity.parameter.mode == 'CV':
                    gnlvcode(v)
                    ascFile.write('lw t' + str(r) + ', 0(t0)\n')
```

Η storevr(r,v)

Βοηθητικές Συναρτήσεις – storerv

- **μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v)**
- η σύνταξη της είναι storerv(r,v)
- διακρίνουμε περιπτώσεις

Η οποια εχει τις ιδιες περιπτωσεις εκτος του να ειναι η v σταθερα.

```
def storerv(r,v):

        global topScope
        global ascFile

        (scope,entity) = find_entity(v)

        if scope.nestingLvl == topScope.nestingLvl:
                if entity.type == 'VAR':
                        ascFile.write('sw t' + str(r) + ', -' + str(entity.variable.offset) + '(sp)\n')
                elif entity.type == 'PARAM' and entity.parameter.mode == 'CV':
                        ascFile.write('sw t' + str(r) + ', -' + str(entity.parameter.offset) + '(sp)\n')
                elif entity.type == 'TEMP':
                        ascFile.write('sw t' + str(r) + ', -' + str(entity.tempVar.offset) + '(sp)\n')

        elif scope.nestingLvl == 0 and entity.type == 'VAR':
                ascFile.write('sw t' + str(r) + ', -' + str(entity.variable.offset) + '(sp)\n')
        elif scope.nestingLvl == 0 and entity.type == 'TEMP':
                ascFile.write('sw t' + str(r) + ', -' + str(entity.tempVar.offset) + '(sp)\n')

        elif scope.nestingLvl < topScope.nestingLvl:
                if entity.type == 'VAR':
                        gnlvcode(v)
                        ascFile.write('sw t' + str(r) + ', 0(t0)\n')
                elif entity.type == 'PARAM' and entity.parameter.mode == 'CV':
                        gnlvcode(v)
                        ascFile.write('sw t' + str(r) + ', 0(t0)\n')
```

Και τελος ξεκιναει η μετατροπη του κωδικα απο quads σε assmbly. Διατρεχω τη λιστα ολων των quads και ξεκιναω να δουλευω την καθε περιπτωση. Τις εντολες τις πηρα απευθειας απο τις διαφανειες και αναλογα την εντολη ειτε τραβαω ειτα αποθηκευω στους καταχωριτες τις μεταβλητες μου.

Παραθετω μερικα παραδειγματα:

```
if listOfAllQuads[i][1] == 'jump':
        ascFile.write('j label' + str(listOfAllQuads[i][4]) + '\n') #4 as jump target
```

```
elif listOfAllQuads[i][1] == '!=':
        loadvr(listOfAllQuads[i][2],1)
        loadvr(listOfAllQuads[i][3],2)
        ascFile.write('bne t1,t2,label' + str(listOfAllQuads[i][4]) + '\n')
```

```
elif listOfAllQuads[i][1] == '+':
        loadvr(listOfAllQuads[i][2],1)
        loadvr(listOfAllQuads[i][3],2)
        ascFile.write('add t1,t1,t2\n')
        storerv(1,listOfAllQuads[i][4])
```

Ιδιαιτερη θεωρω την συναρτηση για το inp καθως ο καταχωρητης a7 ειναι για να ερχονται system calls, και συγκεκριμενα το 5 ειναι για να διαβασει εναν ακαιραιο απο τον χρηστη.

```
elif listOfAllQuads[i][1] == 'inp':
        ascFile.write('li a7,5\n') # in as
        ascFile.write('ecall\n')
        ascFile.write('mv t1,a0\n')
        storerv(1,listOfAllQuads[i][2])
```

Επεισης ιδιαιτερες ειναι και οι περιπτωσεις του par και call

για το par:

για αρχη τοποθετω τον frame pointer να δειχνει στην στοιβα της συναρτησης που θα δημειουργηθει

```
ascFile.write('addi fp,sp,-' + str(entity.subprogram.frameLength) + '\n')
```

το frameLength ομως πρεπει να το βρω, αρα διατρεχω την λιστα των τετραδων μεχρι να βρω το call. Περνω το ονομα της συναρτησης και με το find_entity() βρισκω το επιπεδο και την εγγραφη της και ετισ βρηκα το frameLength. Στη συνεχεια θετω τον μετρητη (Cntr) που ειχα πριν στο -1 ως 0 γιατι απο εδω και περα θελω να μετραω το πληθος των παραμετρων. Ελενχω αν προκειτε για παραμετρο που περναει με τιμη (CV) ή για παραμετρο επιστροφης (RET). Στην πρωτη περιπτωση φωρτωνω τον καταχωριτη και τον αποθηκευω στη θεση -(12*4Cntr).

```
elif listOfAllQuads[i][1] == 'par':
        if Cntr == -1:
                x = 0
                while x < len(listOfAllQuads): # loop till it find call or reach end of list

                        if listOfAllQuads[x][1] == 'call':
                                result = str(listOfAllQuads[x][2]) #2 for the name
                                break

                        x += 1
                (scope,entity) = find_entity(result)
                ascFile.write('addi fp,sp,-' + str(entity.subprogram.frameLength) + '\n')
                Cntr = 0

        if listOfAllQuads[i][3] == 'CV':
                loadvr(listOfAllQuads[i][2],0)

                ascFile.write('sw t0,-' + str(12 + 4*Cntr) + '(sp)\n')


        #elif listOfAllQuads[i][3] == 'REF':
                #loadvr(listOfAllQuads[i][2],0)
                #gnlvcode(listOfAllQuads[i][2])
                #ascFile.write('sw t0,-' + str(12 + 4*Cntr) + '(sp)\n')

        elif listOfAllQuads[i][1] == 'RET':
                (scope,entity) = find_entity(listOfAllQuads[i][2])
                ascFile.write('addi t0,sp' + str(entity.tempVar.offset) +'\n')
                ascFile.write('sw t0,-8(fp)\n')
```

Στην περιπτωση call:

πρεπει να γεμισω το δευτερο πεδιο του εγγραφηματος δραστηριοποιησης της συναρτησης με την διευθυνση του εγγραφηματος δραστηριοποιησης του γονεα. Με αυτο τον τροπο η συναρτηση θα ξερει που να ψαξει σε περιπτωση που χρειαστει να προσπελασει μια μεταβλητη που δεν της ανοικει. Πρωτα ψαχνει με το find_antity(). Αν εχουν το ιδιο βαθος φωλιασματος θα εχουν και ιδιο γονεα, αρα μεταφερω στον καταχωριτη t0 το περιεχομενο του sp θεση -4, και το αποθηκευω στην ιδια του fp. Αν το βαθος ειναι διαφορετικο τοτε απλα αποθηκευω στη -4 του fp το περιεχομενο της sp. Τελος μεταφερω τον δεικτη κανω την κληση της συναρτησης και επιστρεφω τον δεικτη παλι πισω στο δεικτη στοιβας.

```
elif listOfAllQuads[i][1] == 'call':
        Cntr = -1
        (scope,entity) = find_entity(listOfAllQuads[i][2])
        if topScope.nestingLvl == entity.subprogram.nestingLvl:
                ascFile.write('lw t0,-4(sp)\n')
                ascFile.write('sw t0,-4(fp)\n')

        elif topScope.nestingLvl < entity.subprogram.nestingLvl:
                ascFile.write('sw sp,-4(fp)\n')

        ascFile.write('addi sp,sp,' + str(entity.subprogram.frameLength) + '\n')
        ascFile.write('jal label' + str(entity.subprogram.startQuad) + '\n')
        ascFile.write('addi sp,sp,-' + str(entity.subprogram.frameLength) + '\n')
```

Τελος υπαρχουν δυο περιπτωσεις για το begin_block μια να ειναι σε καποια συναρτηση οπου θα αποθηκευει στην πρωτη θεση του εγγραφηματος δραστηριοτητας την διευθυνση επιστροφης. Και μια να ειναι στο κυριως προγραμμα που πρεπει να ανεβασει τον δεικτη στην αρχη του αρχειου να γραψει j label' + str(listOfAllQuads[i][0]) ως δεικτη του κυριως προγραμματος. Επειτα παει παλι στο τελος του αρχειου και κατεβαζει τον sp οσο το framelength της main και συμειωνω στον gp το εγγραφημα δραστηριοτητας.

```
elif listOfAllQuads[i][1] == 'begin_block' and topScope.nestingLvl == 0:
        ascFile.seek(0, os.SEEK_SET)
        ascFile.write('j label' + str(listOfAllQuads[i][0]) + '\n')
        ascFile.seek(0, os.SEEK_END)
        #This will move the pointer to thestart of the file then label to main and the return to where it was.
        ascFile.write('addi sp,sp,' + str(compute_offsets()) + '\n')
        ascFile.write('move gp,sp\n')
```

Τελος η σωστη λειτουργεια επιβαιβεωνετε με τους διαφορους ελενχους που εχουμε ανα τακτα διαστηματα ξεκινωντασ απο τον λεκτικο αναλυτη και συντακτικο αναλυτη

```
[119, 'print', 118]
[223, '(', 118]
[210, 'isPrime', 118]
[223, '(', 118]
[210, 'i', 118]
[224, ')', 118]
[224, ')', 118]
[210, 'i', 119]
[233, '=', 119]
[210, 'i', 119]
```

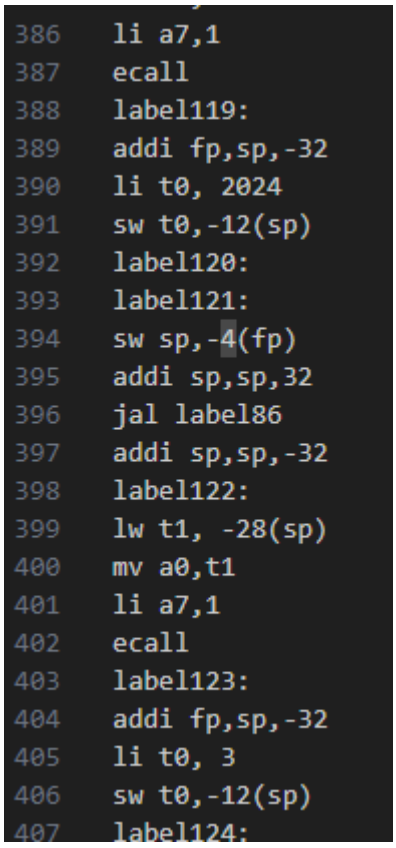
τον ενδοιαμεσο κωδικα

```
8 jump _ _ 8
9 > y x 11
10 jump _ _ 14
11 > y z _
12 jump _ _ 14
13 = y _ m
14 jump _ _ 8
15 = z _ m
16 retv m _ _
17 end_block max3 _ _
18 begin_block fib _ _
19 + counterFunctionCalls 1 T_2
20 < x 0 22
21 jump _ _ 23
22 retv 1 _ _
23 jump _ _ 23
24 == x 0 _
25 jump _ _ 26
26 == x 1 _
```

τον πινακα συμβολων

```
COPE: name:quad nestingLevel:1
      ENTITIES:
      ENTITY:  name:y  type:VAR        variable-type:Int       offset:12
COPE: name: nestingLevel:0
      ENTITIES:
      ENTITY:  name:x  type:VAR        variable-type:Int       offset:0
```

αλλα και τον τελικο



```
386   li a7,1
387   ecall
388   label119:
389   addi fp,sp,-32
390   li t0, 2024
391   sw t0,-12(sp)
392   label120:
393   label121:
394   sw sp,-4(fp)
395   addi sp,sp,32
396   jal label86
397   addi sp,sp,-32
398   label122:
399   lw t1, -28(sp)
400   mv a0,t1
401   li a7,1
402   ecall
403   label123:
404   addi fp,sp,-32
405   li t0, 3
406   sw t0,-12(sp)
407   label124:
```

Αν και εχω εξοδο σε γλωσσα μηχανης αυτην δεν ειναι 100% σωστη καθως εχουν τυπωθει καποια κενα μετα απο καποια labels ομως δεν καταφερα να βρω και να διορθωσω την πηγη του προβληματος.