Translation of CISCAL file to ASSEMBLY implemented in PYTHON

Ο κώδικας υλοποίησης μας αποτελείται από 4 φάσεις.

Η 1η είναι η Λεκτική και Συντακτική Ανάλυση, η 2η η Παραγωγή Ενδιάμεσου Κώδικα,
η 3η η Σημασιολογική Ανάλυση και ο Πίνακας Συμβόλων, και η 4η η Παραγωγή Τελικού Κώδικα.

```
1   import sys
```

Αρχικά εισάγουμε το module **sys** (system-specific parameters and functions),

```
7   #LEKTIKOS ANALYTHS
8
9   #alphabet characters
10  alphabet =['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
11  |       |       |       'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
12
13  #natural numbers
14  numbers=['0','1','2','3','4','5','6','7','8','9']
```

Ξεκινάμε με τη δημιουργία του, **λεκτικού αναλυτή**, ο οποίος διαβάζει τους χαρακτήρες ενός αρχικού προγράμματος και επιστρέφει τις λεκτικές του μονάδες. Αποθηκεύουμε τα γράμματα της αγγλικής αλφαβήτας, σε κεφαλαία και μικρά, στη **λίστα alphabet** και τους φυσικούς αριθμούς στη **λίστα numbers**.

```
16  #open test file
17  file = open(str(sys.argv[1]),'r')
```

Παίρνουμε το αρχείο από τη γραμμή εντολών, ώστε να διαβαστεί το κείμενο, μέσω της μεταβλητής **file**. Το ανοίγουμε για λειτουργία **read** στη θέση 1, καθώς υπάρχουν 2 ορίσματα.

```
20  #transition table characters -> xarakthrew
21  |
22  |       |       |       |   # Symbol
23
24  whitecharacter=0                # " "
25
26  letters=1                   # [a,b,..,w,z]
27
28  number=2                    # [1,2,...,8,9]
29
30  plus=3                      # +
31  minus=4                     # -
32  multiply=5                  # *
33  divide=6                    # /
34
35  equal=7                     # =
36  lessThan=8                  # <
37  greaterThan=9               # >
38
39  EndOfFile=10                #
```

Θέτουμε σε μεταβλητές τους χαρακτήρες που αναγνωρίζει το αυτόματο καταστάσεων.

```
41  notAcceptableSymbol=11      #
42
43  comma=12                    # ,
44  questionMark=13             # ;
45
46  leftParenthesis=14          # (
47  rightParenthesis=15         # )
48  leftBracket=16              # [
49  rightBracket=17             # ]
50
51  blockOpening=18             # {
52  blockClosing=19             # }
53
54  lineChange= 20              # (Enter)
55
56  colon=21                    # :
57  period=22                   # .
58  hashtag=23
```

```
61  #Status -> Katastaseis
62
63  Status_start=0              # start
64
65  Status_letter=1             # a,b,...,y,z
66  Status_number=2             # 1,2,...
67
68  Status_lessThan=3           # <
69  Status_greaterThan=4        # >
70
71  Status_assignment=5         # :=
72  Status_comments=6           # #
```

Θέτουμε σε μεταβλητές τις **καταστάσεις αναγνώρισης**, <u>που είναι:</u>

- Αρχική κατάσταση
- Γράμμα
- Αριθμός
- Μικρότερο από - Μεγαλύτερο από
- Καταχώρηση
- Σχόλιο

```
74  #Tokens
75
76  identifier_Token=50         # name of anything we need
77  number_Token=51             # [1,2,...,8,9]
78
79  plus_Token=52               # +
80  minus_Token=53              # -
81  multiply_Token=54           # *
82  divide_Token=55             # /
83
84  equal_Token=56              # =
85  lessThan_Token=57           # <
86  greaterThan_Token=58        # >
87
88  EndOfFile_Token=59          #
```

Θέτουμε τα αναγνωριστικά tokens, που αντιστοιχούν στους χαρακτήρες που θέσαμε προηγουμένως.

```
90  comma_Token=60              # ,
91  questionMark_Token=61       # ;
92
93  leftParenthesis_Token=62    # (
94  rightParenthesis_Token=63   # )
95  leftBracket_Token=64        # [
96  rightBracket_Token=65       # ]
97
98  blockOpening_Token=66       # {
99  blockClosing_Token=67       # }
100 lessORequal_Token=68         # <=
101 greaterORequal_Token=69      # >=
102
103 colon_Token=70               # :
104 assignment_Token=71          # :=
105 notEqual_Token=72            # !=
106 period_Token=73              # .
```

```
109 #DESMEUMENES LEKSEIS PROGRAMMATOS
110
111 desmeumenes_lexeis=['program' ,'declare','if','else',
112                      'while','switchcase',
113                      'forcase','incase','default','case',
114                      'not','and','or',
115                      'function','procedure','call','return','in','inout',
116                      'input','print']
```

Θέτουμε τις δεσμευμένες λέξεις, που αναγνωρίζει το αυτόματο καταστάσεων.

```
120 program_Token=100
121 declare_Token=101
122
123 if_Token=102
124 else_Token=103
125 while_Token=104
126
127 switchcase_Token=105
128 incase_Token=106
129 forcase_Token=107
130 case_Token=108
131
132 default_Token=109
```

Θέτουμε τα tokens που αντιστοιχούν στις δεσμευμένες λέξεις, που θέσαμε προηγουμένως.

```
133 procedure_Token=110
134 function_Token=111
135 call_Token=112
136 return_Token=113
137
138 in_Token=114
139 inout_Token=115
140
141 and_Token=116
142 or_Token=117
143 not_Token=118
144
145 input_Token=119
146 print_Token=120
```

```
#Errors

not_Acceptable_Symbol_Error=-1
digit_letter_Error=-2
colon_Error=-3
number_Excepts_Space_Error=-4
over_30_characters_Error=-5
statements_Open_on_EndOfFile_Error=-6
```

Θέτουμε τα errors που ενδέχεται να συναντήσουμε.
Συγκεκριμένα, αν εντοπίστηκε :

- μη αποδεκτό σύμβολο,
- αριθμός μετά από γράμμα
- : που δεν ακολουθείται από =
- αριθμός εκτός των διαχειρίσιμων ορίων
- λέξη με παραπάνω από 30 χαρακτήρες
- σχόλια που ανοίγουν, αλλά δεν κλείνουν.

```
transitionTable=[
        #Status_start -> we await a character
        [Status_start,Status_letter,Status_number,                  plus_Token,minus_Token,multiply_Token, divide_Token,

        equal_Token,Status_lessThan,Status_greaterThan,             EndOfFile_Token,not_Acceptable_Symbol_Error,

        comma_Token,questionMark_Token,

        leftParenthesis_Token,rightParenthesis_Token,leftBracket_Token,rightBracket_Token,blockOpening_Token,blockClosing_Token,

        Status_start,Status_assignment,                             period_Token,Status_comments],                  # we move to the next status, by checking the next symbol

    #Status_letter
        [identifier_Token,Status_letter,Status_letter,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token
        identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier_Token,identifier

    #Status_number
        [number_Token,digit_letter_Error, Status_number,number_Token,number_Token,number_Token,
        number_Token,number_Token,number_Token,number_Token,number_Token,not_Acceptable_Symbol_Error,
        number_Token,number_Token,number_Token,number_Token,number_Token,number_Token,number_Token,number_Token,
        number_Token,number_Token,number_Token,number_Token],
```

Ορίζουμε τον πίνακα μεταβάσεων, μέσω του οποίου επιλέγεται η επόμενη κατάσταση,
με βάση το σύμβολο που εξετάζουμε. Στην αρχική κατάσταση, αναγνωρίζουμε το πρώτο σύμβολο του κειμένου, και στις υπόλοιπες αναγνωρίζουμε γράμματα, αριθμούς, τα σύμβολα < , > , : , { .
Αν δεν εντοπίσουμε ως επόμενο χαρακτήρα κάποιον από τους αναμενόμενους, υπάρχει λάθος.

```
    #Status_lessThan
        [lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,
        lessThan_Token,lessOREqual_Token,lessThan_Token,notEqual_Token,lessThan_Token,not_Acceptable_Symbol_Error,
        lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token,
        lessThan_Token,lessThan_Token,lessThan_Token,lessThan_Token],

    #Status_greaterThan
        [greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,
        greaterThan_Token,greaterOREqual_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,not_Acceptable_Symbol_Error,
        greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token,
        greaterThan_Token,greaterThan_Token,greaterThan_Token,greaterThan_Token],

    #Status_assignment
        [colon_Error,colon_Error,colon_Error,colon_Error,colon_Error,colon_Error,
        colon_Error,assignment_Token,colon_Error,colon_Error,colon_Error,not_Acceptable_Symbol_Error,
        colon_Error,colon_Error,colon_Error,colon_Error,colon_Error,colon_Error,colon_Error,colon_Error,
        colon_Error,colon_Error,colon_Error,colon_Error],

    #Status_comments
        [Status_comments,Status_comments,Status_comments,Status_comments,Status_comments,Status_comments,
        Status_comments,Status_comments,Status_comments,Status_comments,statements_Open_on_EndOfFile_Error,Status_comments,
        Status_comments,Status_comments,Status_comments,Status_comments,Status_comments,Status_comments,Status_comments,Status_comments,
        Status_start,Status_comments,Status_comments,Status_start]

        ]
```

```
207 line=1          # initialize line counter
208
209 def characterCheck(character, linecounter):
210
211         if (character == ' ' or character == '\t'):
212                 character_Token = whitecharacter
213
214         elif (character in alphabet):
215                 character_Token = letters
216         elif (character in numbers):
217                 character_Token = number
218
219         elif (character == '+'):
220                 character_Token = plus
221         elif (character == '-'):
222                 character_Token = minus
223         elif (character == '*'):
224                 character_Token = multiply
225         elif (character == '/'):
226                 character_Token = divide
```

```
228         elif (character == '='):
229                 character_Token = equal
230         elif (character == '<'):
231                 character_Token = lessThan
232         elif (character == '>'):
233                 character_Token = greaterThan
234
235         elif (character == ':'):
236                 character_Token = colon
237         elif (character == ','):
238                 character_Token = comma
239         elif (character == ';'):
240                 character_Token = questionMark
241
242         elif (character == '('):
243                 character_Token = leftParenthesis
244         elif (character == ')'):
245                 character_Token = rightParenthesis
246         elif (character == '['):
247                 character_Token = leftBracket
248         elif (character == ']'):
249                 character_Token = rightBracket
250         elif (character == '{'):
251                 character_Token = blockOpening
252         elif (character == '}'):
253                 character_Token = blockClosing
```

Αρχικοποιούμε τη μεταβλητή line στο 1, ώστε να την αξιοποιήσουμε παρακάτω.

Αποθηκεύουμε στη μεταβλητή **character_Token** το χαρακτήρα που αναγνωρίσαμε.

```
271 def lex():                                  #token finder (code,token,line) [(int,string,int)]
272
273         global line                         #current line
274         producedWord=''                     #produced token
275         current= Status_start               #sets situation to start
276
277         linecounter= line                   #holds the number of checked lines
278         resultlex=[]                        #list for the 3 parts of the token
279
280         while(current>=0 and current<=6):                   #check if token is found
281                 character = file.read(1)                        #reads the next character
282
283                 character_Token = characterCheck(character,linecounter)         #calls characterCheck
284
285                 current=transitionTable[current][character_Token]                       #move from last status to next
286
287                 if(len(producedWord)<30):
288                         if(current!=Status_start and current!=Status_comments):         #ignore start and comments
289                                 producedWord+=character                                         #add the current character to the produced word
290
291                 else:
292                         current=over_30_characters_Error                        #show error if more than acceptable length
```

Η μέθοδος **lex()** είναι ο λεκτικός αναλυτής.

Βρίσκει και επιστρέφει τη λεκτική μονάδα (**token)** με τις πληροφορίες της.

Αποθηκεύουμε το παραγόμενο token, που προέκυψε από την λεκτική μονάδα που διαβάστηκε, στην μεταβλητή **producedWord**, την σειρά που βρισκόμαστε στη **linecounter**, την κατάσταση με βάση τον πιο πρόσφατο χαρακτήρα που αναγνωρίσαμε. Τις τοποθετούμε στο **resultlex** στο τέλος της μεθόδου.

Η μεταβλητή **current** διατηρεί την πιο πρόσφατη κατάσταση, οπότε την Αρχικοποιούμε με την αρχική, **Status_start**. Αγνοούμε τα σχόλια, και την αρχική και τελική κατάσταση, και αναγνωρίζουμε τους χαρακτήρες, με τη σειρά που εμφανίζονται. Αναμένουμε την επόμενη κατάσταση με βάση των πίνακα μεταβάσεων.

```
297         if(current==identifier_Token or current==number_Token or current==lessThan_Token or current==greaterThan_Token ):
298             if (character == '\n'):
299                 linecounter -= 1
300             character=file.seek(file.tell()-1,0)              #returns last character read on File
301 
302             producedWord = producedWord[:-1]                    #cuts the last character
```

```
304         if(current==identifier_Token):
305             if(producedWord in desmeumenes_lexeis):             #check for commited words
306 
307                 if(producedWord=='program'):                    # program
308                     current=program_Token
309                 elif(producedWord=='declare'):                  # declare
310                     current=declare_Token
311                 elif (producedWord == 'if'):                    # if
312                     current = if_Token
313                 elif (producedWord == 'else'):                  # else
314                     current = else_Token
315                 elif (producedWord == 'while'):                 # while
316                     current = while_Token
317                 elif (producedWord == 'switchcase'):            # switchcase
318                     current = switchcase_Token
319                 elif (producedWord == 'forcase'):               # forcase
320                     current = forcase_Token
321                 elif (producedWord == 'incase'):                # incase
322                     current = incase_Token
323                 elif (producedWord == 'case'):                  # case
324                     current = case_Token
325                 elif (producedWord == 'default'):               # default
326                     current = default_Token
```

Σβήνουμε από το παραγόμενο token τον τελευταίο χαρακτήρα που διαβάστηκε.

Αφού έχουμε εντοπίσει τον χαρακτήρα ανάθεσης, σε περίπτωση που το παραγόμενο token ανήκει στις δεσμευμένες λέξεις, αποθηκεύουμε στην μεταβλητή **current** την αντίστοιχη μεταβλητή του token.

```
327                 elif (producedWord == 'procedure'):          # procedure
328                     current = procedure_Token
329                 elif (producedWord == 'function'):           # function
330                     current = function_Token
331                 elif (producedWord == 'call'):               # call
332                     current = call_Token
333                 elif (producedWord == 'return'):             # return
334                     current = return_Token
335                 elif (producedWord == 'in'):                 # in
336                     current = in_Token
337                 elif (producedWord == 'inout'):              # inout
338                     current = inout_Token
339                 elif (producedWord == 'and'):                # and
340                     current = and_Token
341                 elif (producedWord == 'or'):                 # or
342                     current = or_Token
343                 elif (producedWord == 'not'):                # not
344                     current = not_Token
345                 elif (producedWord == 'input'):              # input
346                     current = input_Token
347                 elif (producedWord == 'print'):              # print
348                     current = print_Token
```

Ελέγχουμε για τα πιθανά errors, που θέσαμε παραπάνω, και ενημερώνουμε τον χρήστη εμφανίζοντας το κατάλληλο μήνυμα λάθους.

```
351         if (current == number_Token):                        #check for number belonging in [-32767,32767]
352             if (producedWord.isdigit() >= pow(2,32)):            #pow(2,32)=2^32
353                 current = number_Excepts_Space_Error         #note as error if outside accepted prices
```

```
356         #CHECK FOR POSSIBLE ERRORS
357         if(current==not_Acceptable_Symbol_Error):
358             print("FOUND ERROR: Symbol is not recognised")
359         elif(current==digit_letter_Error):
360             print("FOUND ERROR: Letter spoted after digit")
361         elif(current==colon_Error):
362             print("FOUND ERROR: Colon symbol : is not followed by equal symbol =")
363         elif(current==number_Excepts_Space_Error):
364             print("FOUND ERROR: The number is outside the space [-(2^32-1),2^32-1]")
365         elif(current==statements_Open_on_EndOfFile_Error):
366             print("FOUND ERROR: Comments { were opened, but were not closed properly")
367         elif(current==over_30_characters_Error):
368             print("FOUND ERROR: Current word consists of over 30 digits")
```

```
370         # line 0 _Token, line 1 word, 
371 
372         resultlex.append(current)
373         resultlex.append(producedWord)
374         resultlex.append(linecounter)
375         line=linecounter
376         return resultlex
```

Αφού έχουμε σχηματίσει επιτυχώς την λέξη που συναντήσαμε, προσθέτουμε στο **resultlex** την τελευταία κατάσταση (current), την παραγόμενη λεκτική μονάδα (producedWord), και τον αριθμό των σειρών που μετρήσαμε (linecounter). Ενημερώνουμε τη line με την τελευταία σειρά όπου αναγνωρίσαμε token, και επιστρέφουμε το resultlex.

```
380 # Endiamesos kodikas ->
381 
382 global cFile
383 
384 global listOfTotalQuads
385 listOfTotalQuads = []
386 countQuad = 1
```

Συνέχεια έχει ο ενδιάμεσος κώδικας, ο οποίος είναι υπεύθυνος για την κατασκευή 4αδων, ώστε να παράγει ο συντακτικός μια def συνάρτηση, για τον κάθε κανόνα της γραμματικής. Η **listOfTotalQuads** περιέχει τις παραγόμενες 4αδες, τύπου **label**: **op,x,y,z**, και η **countQuad** διατηρεί τον αριθμό ταυτοποίησης τους, που αντιστοιχεί στον αριθμό εμφάνισης. Ορίζεται ως global μέσα στις μεθόδους, όπως και οι περισσότερες μεταβλητές στον παρών αρχείο, ώστε να διατηρούνται οι αλλαγές στην τιμή της.

```
389 def nextQuad():
390 
391     global countQuad
392 
393     return countQuad
```

Η **nextQuad** με την κλήση της ενημερώνει και επιστρέφει

την τιμή της **countQuad**, δηλαδή τον αριθμό 4αδας που θα παραχθεί.

(δηλαδή το **label**)

```
397 def generateQuad(first, second, third, fourth):      #cr
398 
399     global countQuad                                #We end
400     global listOfTotalQuads                         #the lis
401     list = []                                       #list fo
402 
403     list = [nextQuad()]
404     list += [first] + [second] + [third] + [fourth]
405 
406     countQuad +=1                                   #increas
407     listOfTotalQuads += [list]                      #Puts qu
408     return list                                     #return
```

Η **generateQuad(first, second, third, fourth)** είναι υπεύθυνη για τη δημιουργία της επόμενης 4αδας.

Κατά την κλήση της δέχεται 4 ορίσματα, τα οποία προσθέτει στην list, μετά που λάβει την **countQuad**, από την **nextQuad,** και την τοποθετήσει στην πρώτη θέση της λίστας.

Έπειτα αυξάνει τον μετρητή των 4αδων, και προσθέτει τη λίστα που δημιούργησε στο τέλος της λίστας των συνολικών 4αδων, και επιστρέφει την λίστα που περιέχει την 4αδα που δημιούργησε.

```
410 T_i = 1
411 listOfTemporaryVariables = []
```

Αρχικοποιούμε τη μεταβλητή **T_i,** η οποία διατηρεί την προσωρινή τιμή, και τη λίστα **listOfTemporaryVariables**, η οποία διατηρεί αυτές τις προσωρινές τιμές.

```
414  def newTemp():                              #creates ar
415
416          global T_i                          #counter fo
417          global listOfTemporaryVariables          #list
418
419          list = ['T_']
420          list.append(str(T_i))               #creates st
421          tempVariable="".join(list)          #puts space
422          T_i +=1
423
424          listOfTemporaryVariables += [tempVariable]
425
426          return tempVariable
```

Η **newTemp()** είναι υπεύθυνη για τη δημιουργία των **T_**, προσωρινών μεταβλητών, και την προσθήκη τους στη λίστα.

Αρχικά δημιουργεί μια λίστα που περιέχει το **string 'T_i'**, και στη συνέχεια το ενώνει με τον αριθμό που έχει κρατήσει η μεταβλητή **T_i**, και το προσθέτει στη λίστα των προσωρινών μεταβλητών.

Αυξάνει το **T_i** κατά 1, ώστε να καταχωρίσει την επόμενη τιμή, όταν ξανακληθεί η μέθοδος.

```
429  def emptyList():
430
431          pointerList = []
432
433          return pointerList
```

Η **emptyList()** είναι υπεύθυνη για την αρχικοποίηση και επιστροφή της κενής λίστας **pointerList**, ώστε να της τοποθετηθούν σε άλλη μέθοδο τα 4 ετικετών (**tags**).

```
436  def makeList(x):
437
438          listThis = [x]
439
440          return listThis
```

Η **makeList()** είναι υπεύθυνη για την αρχικοποίηση και επιστροφή της λίστας **listThis**, που θα περιέχει μόνο το **x** tag, που θα ορίζεται κατά την κλήση της μεθόδου.

```
442  def merge(list1, list2):
443
444          list=[]
445          list += list1 + list2
446
447          return list
448
```

Η **emptyList()** είναι υπεύθυνη για την αρχικοποίηση και επιστροφή της λίστας **list**, στην οποία τοποθετούμε τη συνένωση των επιμέρους λιστών **list1** και **list2**, που ορίζονται κατά την κλήση της μεθόδου.

```
449  def backPatch(list, z):                    #backPatch checks each quad and adds quad z.'
450
451          global listOfTotalQuads             #list is constructed by listOfTotalQuads quads of
452
453
454          for i in range(len(list)):
455                  for j in range(len(listOfTotalQuads)):
456                          if(list[i]==listOfTotalQuads[j][0] and listOfTotalQuads[j][4]=='_'):
457                                  listOfTotalQuads[j][4] = z
458                                  break;
459          return
```

Η **backpatch(list,z)** τοποθετεί την ετικέτα z στην 4η θέση όποιας λίστας, που περιέχεται στην λίστα 4αδων ετικετών **listofTotalQuads**, και έχει κενή την 4η της θέση, επειδή δεν γνωρίζουμε στο στάδιο της αρχικής κλήσης, που θα μεταφερθεί στη συνέχεια η εκτέλεση.

```
461  def syntax_an():
462
463          global line
464          global lexres
465          lexres= lex()
466          line = lexres[2]
```

Η **syntax_an()** καλεί την μέθοδο **lex()**, και αποθηκεύει την 3αδα που λαμβάνει στην **lexres**.

Επιπλέον, τοποθετεί την 2η θέση της **lexres,** δηλαδή την τιμή της linecounter, της πιο πρόσφατης σειράς, στην **line**.

```
468  def program():                          # start of program
469
470          global line
471          global lexres
472
473          if(lexres[0] == program_Token):                        #we identify the expected program word
474                  lexres = lex()
475                  line = lexres[2]
476
477                  if(lexres[0] == identifier_Token):             #then set the expected identifier
478                          id = lexres[1]
479                          lexres = lex()
480                          line = lexres[2]
481
482                          block(id,1)
483
484                          if(lexres[0] == period_Token):         #then we find the expected . symbol
485                                  lexres = lex()
486                                  line = lexres[2]
487
488                                  return
489                          else:
490                                  print("FOUND ERROR: Period was not found", line)
491                                  exit(-1)
492
493                  else:
494                                  print("FOUND ERROR: Can not locate file or program name",line)
495                                  exit(-1)
496          else:
497                  print("FOUND ERROR: The word 'program' is not found on the start of the program",line)
498                  exit(-1)
```

Η μέθοδος **program()** διαχειρίζεται το κυρίως πρόγραμμα, και είναι κομμάτι του συντακτικού αναλυτή.

Αρχικά πρέπει να βρει την λέξη token **program** στο κείμενο, στη συνέχεια αναγνωρίζει το όνομα του προγράμματος id, και αναμένει την τελεία.

Αν κάποια από τις παραπάνω συνθήκες δεν ισχύει, ενημερώνουμε τον χρήστη με μήνυμα λάθους.

Για την αναγνώριση των συμβόλων και λέξεων καλεί την μέθοδο **lex()**, όταν βρει το όνομα ταυτοποίησης προγράμματος, καλεί την **block().**

```
500  def block(name, flag):                # initiates program or subprogram
501
502          global line
503          global lexres
504
505          if(lexres[0] == blockOpening_Token):                    #we find the [ symbol
506                  lexres = lex()
507                  line  =  lexres[2]
508
509                  declarations()
510
511                  subprograms()
512
513                  generateQuad('begin_block',name,'_','_')        #initiates program or subprogram called
514
515                  blockstatements()
516
517                  if(flag==1):
518                          generateQuad('halt','_','_','_')        #ends program
519                          generateQuad('end_block',name,'_','_')  #ends program or subprogram called name
520
521                  if(lexres[0] == blockClosing_Token):            #then we find the expected ] symbol
522                          lexres = lex()
523                          line = lexres[2]
524                          return
```

Η **block(name,flag)** καλείται μετά που εντοπιστούν η λέξη **program**, και το όνομα του προγράμματος, προκειμένου να αναγνωρίσει το άνοιγμα και το κλείσιμο των **block**, εντός των οποίων βρίσκεται το πρόγραμμα ή κάποιο υποπρόγραμμα.

Αφού εντοπίσει το άνοιγμα των αγκύλων, καλεί την μέθοδο **declarations()** η οποία θα αναγνωρίσει τα ονόματα των μεταβλητών, και θα τα καταγράψει στο αρχείο c.

Καλεί τη μέθοδο **generateQuad('begin_block',name,'_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στο σύμβολο **{** , και έπειτα καλεί την **blockStatements()** για να διαχειριστεί τις διαδικασίες που έπονται του εντοπισμού προγράμματος και υποπρογράμματος.

Αν η flag ισούται με 0, καλεί την **generateQuad('halt','_','_','_')** ώστε να ενημερώσει για τον τερματισμό του προγράμματος, και την **generateQuad('end_block',name,'_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στο σύμβολο **}** .

```
526  def declarations():                          #in case of declaration token we call it
527
528          global lexres
529          global cFile
530
531          while(lexres[0] == declare_Token):                    #we find the word declare
532                  lexres = lex()
533                  line = lexres[2]
534
535                  cFile.write("int ")
536                  varlist()
537                  cFile.write(";\n\t")
538
539                  if(lexres[0] == questionMark_Token):          #then we find the expected ; symbol
540                          lexres = lex()
541                          line = lexres[2]
542
543                  else:
544                          print("FOUND ERROR: questionMark symbol was not found on the end of the varlist", line)
545                          exit(-1)
546          return
```

Η **declerations()** ψάχνει τη λεκτική μονάδα **declare**, και όσο παραμένει στο στάδιο του κανόνα, γράφει στο **αρχείο c** που θα παραχθεί τη λέξη **int**, και μετά που κληθεί η μέθοδος **varlist(),** η οποία ελέγχει αν υπάρχουν πολλαπλά ονόματα ταυτοποίησης, και γράψει το υπόλοιπο κείμενο, γράφει το σύμβολο **;** .

Αν δεν αναγνωριστεί στη συνέχεια σύμβολο **;** , ενημερώνουμε το χρήστη με μήνυμα λάθους.

```
548  def varlist():                               #in case of multiple identifiers we check the token and call lex()
549
550          global lexres
551          global cFile
552
553          if(lexres[0] == identifier_Token):                    #we set the identifier
554                  cFile.write(lexres[1])
555                  lexres = lex()
556                  line = lexres[2]
557
558                  while(lexres[0] == comma_Token):              #then we find the expected , symbol
559                          cFile.write(lexres[1])
560                          lexres = lex()
561                          line = lexres[2]
562
563                          if(lexres[0] == identifier_Token):    #then we set the expected next identifier
564                                  cFile.write(lexres[1])
565                                  lexres = lex()
566                                  line = lexres[2]
567
568                          else:
569                                  print("FOUND ERROR: Coma was not found before the identifier or there are 2 or mo
570                                  exit(-1)
571
572          return
```

Η **varlist()** ελέγχει αν υπάρχουν μια ή παραπάνω δηλώσεις, με ονόματα ταυτοποίησης χωρισμένα με το σύμβολο **,** . Καταγράφει στο **αρχείο c** τα ονόματα που αναγνωρίζει ανάμεσα στα κόμματα, και αν δεν βρει επαρκή, ενημερώνει τον χρήστη με μήνυμα λάθους.

```
574  def subprograms():              #in case of multiple subprograms, repeat subprrogram call
575  |
576  |       global lexres
577  |
578  |       while(lexres[0] == procedure_Token or lexres[0] == function_Token ):     #we find the word procedure or function
579  |       |       subprogram()
580  |       return
```

Η **subprograms()** καλεί την **subprogram()** όσο εξακολουθούν να υπάρχουν λειτουργίες, όσο δηλαδή βρίσκει **function** ή **procedure**.

```
582  def subprogram():              #in case of subprogram, we call block, in order to initiate it
583  |
584  |       global lexres
585  |
586  |       if(lexres[0]==procedure_Token):                                  #we find the word procedure
587  |       |       lexres=lex()
588  |       |       line=lexres[2]
589  |       |
590  |       |       if(lexres[0]==identifier_Token):                         #then we set the expected identifier
591  |       |       |       id = lexres[1]
592  |       |       |       lexres = lex()
593  |       |       |       line = lexres[2]
594  |       |       |
595  |       |       |       if(lexres[0] == leftParenthesis_Token):          #then we find the expected ( symbol
596  |       |       |       |       lexres = lex()
597  |       |       |       |       line = lexres[2]
598  |       |       |       |
599  |       |       |       |       formalparlist()
600  |       |       |       |
601  |       |       |       |       if(lexres[0] == rightParenthesis_Token):  #the we find the expected ) symbol
602  |       |       |       |       |       lexres = lex()
603  |       |       |       |       |       line = lexres[2]
604  |       |       |       |       |       block(id,0)
605  |       |       |       |       |
606  |       |       |       |       |       return
607  |       |       |       |       else:
608  |       |       |       |       |       print("FOUND ERROR: Right parenthesis is not closed properly after the formalparlist",line)
609  |       |       |       |       |       exit(-1)
610  |       |       |       else:
611  |       |       |       |       print("FOUND ERROR: Left parenthesis is not opened properly before the formalparlist",line)
612  |       |       |       |       exit(-1)
613  |       |       else:
614  |       |       |       print("FOUND ERROR: We await the identifier after the function", line)
615  |       |       |       exit(-1)
```

Η **subprogram()** αρχικά εντοπίζει την **procedure** ή την **function**, και αφού αποθηκεύσει την τελευταία κατάσταση στην id, αν βρει την αριστερή παρένθεση, καλεί την **formalparlist(),** κι εφόσον βρει και την δεξιά, καλεί την **block(id,0)** για να διαχειριστεί τις διαδικασίες που έπονται του εντοπισμού προγράμματος και υποπρογράμματος, με id το όνομα της διαδικασίας που αναγνώρισε.

Αν δεν εντοπιστεί όνομα ταυτοποίησης, ή κάποια από τις παρενθέσεις, ενημερώνουμε τον χρήστη με μήνυμα λάθους.

```
617         elif(lexres[0]== function_Token):
618             lexres = lex()
619             line = lexres[2]
620 
621             if(lexres[0]==identifier_Token):                            #we set the expected identifier
622                 id = lexres[1]
623                 lexres = lex()
624                 line = lexres[2]
625 
626                 if(lexres[0] == leftParenthesis_Token):                 #then we find the expected ( symbol
627                     lexres = lex()
628                     line = lexres[2]
629 
630                     formalparlist()
631 
632                     if(lexres[0] == rightParenthesis_Token):            #then we find the expected ) symbol
633                         lexres = lex()
634                         line = lexres[2]
635                         block(id,0)
636 
637                         return
638                     else:
639                         print("FOUND ERROR: Right parenthesis is not closed properly after the formalparlist",line)
640                         exit(-1)
641                 else:
642                     print("FOUND ERROR: Left parenthesis is not opened properly before the formalparlist",line)
643                     exit(-1)
644             else:
645                 print("FOUND ERROR: We await the identifier after the function ", line)
646                 exit(-1)
```

```
648 def formalparlist():           #in case of multiple items, we repeat formalparitem() call
649 
650     global lexres
651     global line
652     formalparitem()
653 
654     while(lexres[0] == comma_Token):                #we find the , symbol
655         lexres  = lex()
656         line = lexres[2]
657         formalparitem()
658 
659     return
```

Η **formalparlist()** αναγνωρίζει το σύμβολο , και καλεί μια φορά την **formalparitem().**

Όσο εντοπίζει κόμμα, επαναλαμβάνει την κλήση της.

```
661 def formalparitem():           #we save the variables, after the words in or inout, as identifiers
662
663         global lexres
664         global line
665
666         if(lexres[0] == in_Token):                    #we find the word in
667                 lexres = lex()
668                 line = lexres[2]
669
670                 if(lexres[0]== identifier_Token):     #then we set the expected variables name as identifier
671                         lexres = lex()
672                         line = lexres[2]
673
674                 else:
675                         print("FOUND ERROR: We await the variable name after the 'in' ", line)
676                         exit(-1)
677         elif(lexres[0] == inout_Token):               #we find word inout
678                 lexres = lex()
679                 line = lexres[2]
680
681                 if(lexres[0] == identifier_Token):    #then we set the expected variables name as identifier
682                         lexres = lex()
683                         line = lexres[2]
684
685                 else:
686                         print("FOUND ERROR: We await the variable name after the 'inout' ", line)
687                         exit(-1)
688
689         return
```

Η **formalparitem()** αναγνωρίζει τις δεσμευμένες λέξεις **in** ή **inout**, και αποθηκεύει το όνομα των μεταβλητών που εντοπίζει.

Αν δεν βρούμε κανένα από τα **in** ή **inout**, ενημερώνουμε τον χρήστη με μήνυμα λάθους.

```
691  def statements():              #saves the statement and calls statement() to identify it
692
693          global lexres
694          global line
695
696          if(lexres[0] == blockOpening_Token):                  #we find the [ symbol
697                  lexres = lex()
698                  line  =  lexres[2]
699
700                  statement()
701
702                  while(lexres[0] == questionMark_Token):       #then we find the expected ? symbol
703                          lexres = lex()
704                          line = lexres[2]
705
706                          statement()
707
708                  if(lexres[0] == blockClosing_Token):          #then we find the expected ] symbol
709                          lexres = lex()
710                          line = lexres[2]
711                          return
712
713                  else:
714                          print("FOUND ERROR: The block is not closed properly after the statements", line)
715                          exit(-1)
716          else:
717
718                  statement()
719
720                  if(lexres[0] == questionMark_Token):          #we find the expected ? symbol
721                          lexres = lex()
722                          line = lexres[2]
723                          return
724                  else:
725                          print("FOUND ERROR: There is no questionmark after the statement", line)
726                          exit(-1)
```

Η **statements()** αν εντοπίσει αγκύλες, καλεί μια φορά την μέθοδο **statement(),** η οποία αναγνωρίζει τη δεσμευμένη λέξη που διαβάστηκε, και όσο βρίσκει το σύμβολο **?,** και ενημερώνει το χρήστη με μήνυμα λάθους αν δεν το βρει.

Ειδάλλως καλεί μόνο μια φορά την **statement(),** και αναζητά το το σύμβολο **?,** και ενημερώνει το χρήστη με μήνυμα λάθους αν δεν το βρει.

```
728  def blockstatements():            #in case of multiple statements, we repeat statement() call
729  |
730  |       global lexres
731  |       global line
732  |       statement()
733  |
734  |       while(lexres[0] == questionMark_Token):              #we find the ? symbol
735  |       |       lexres = lex()
736  |       |       line = lexres[2]
737  |       |       statement()
```

Η **blockstatements()** καλεί την **statement()** μια φορά, και επαναλαμβάνει την κλήση της όσο βρίσκει το σύμβολο **?**.

```
739  def statement():                  #initiate t
740  |
741  |       global lexres
742  |
743  |       if(lexres[0]==identifier_Token):
744  |       |       assignmentStat()
745  |       elif(lexres[0]==if_Token):
746  |       |       ifStat()
747  |       elif(lexres[0]==while_Token):
748  |       |       whileStat()
749  |       elif(lexres[0]==switchcase_Token):
750  |       |       switchcaseStat()
751  |       elif(lexres[0]==forcase_Token):
752  |       |       forcaseStat()
753  |       elif(lexres[0]==incase_Token):
754  |       |       incaseStat()
755  |       elif(lexres[0]==call_Token):
756  |       |       callStat()
757  |       elif(lexres[0]==return_Token):
758  |       |       returnStat()
759  |       elif(lexres[0]==input_Token):
760  |       |       inputStat()
761  |       elif(lexres[0]==print_Token):
762  |       |       printStat()
763  |
764  |       return
```

Η **statement()** αναγνωρίζει την δεσμευμένη λέξη, και καλεί την αντίστοιχη μέθοδο.

```
766  def assignmentStat():          #saves the word after the identifier
767  |
768  |       global lexres
769  |       global line
770  |
771  |       if(lexres[0] == identifier_Token):             #we find the identifier
772  |       |       myid = lexres[1]
773  |       |       lexres = lex()
774  |       |       line = lexres[2]
775  |       |
776  |       |       if(lexres[0] == assignment_Token):      #then we find the expected := sumbol
777  |       |       |       lexres = lex()
778  |       |       |       line = lexres[2]
779  |       |       |
780  |       |       |       Eplace = expression()
781  |       |       |       generateQuad(':=', Eplace, '_', myid)
782  |       |       |
783  |       |       |       return
784  |       |       else:
785  |       |       |       print("FOUND ERROR: There must be an assignment symbol after the variable symbol.", line)
786  |       |       |       exit(-1)
787  |       else:
788  |       |       print("FOUND ERROR: Does not exist",line)
789  |       |       exit(-1)
```

Η assignmentStat() αποθηκεύει το κείμενο μετά το σύμβολο καταχώρησης := , καλεί την **expression()** και την αποθηκεύει στην **Eplace**, και καλεί την **generateQuad**, με τις κατάλληλες μεταβλητές. Ενημερώνουμε για λάθη, με τα κατάλληλα μηνύματα.

```
791  def ifStat():                  #if case
792  |
793  |       global lexres
794  |       global line
795  |
796  |       if(lexres[0] == if_Token):
797  |       |       lexres= lex()
798  |       |       line = lexres[2]
799  |       |
800  |       |       if(lexres[0] == leftParenthesis_Token):
801  |       |       |       lexres = lex()
802  |       |       |       line = lexres[2]
803  |       |       |
804  |       |       |       C = condition()
805  |       |       |       backPatch(C[0], nextQuad())
```

Η **ifStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **if,** και αν βρει την αριστερή παρένθεση καλεί και αποθηκεύει την **condition** στην **C**, και καλεί την **Backpatch(C[0],nextQuad()).**

```
807                     if(lexres[0]== rightParenthesis_Token):          #then we find the expected ] symbol
808                             lexres = lex()
809                             line = lexres[2]
810
811                             statements()
812
813                             ifList = makeList(nextQuad())
814                             generateQuad('jump', '_', '_', '_')
815                             backPatch(C[1], nextQuad())
816
817                             elsepart()
818
819                             backPatch(ifList, nextQuad())
820
821                             return
822                     else:
823                             print("FOUND ERROR: Parenthesis is not closed properly after the if case", line)
824                             exit(-1)
825             else:
826                     print("FOUND ERROR: Parenthesis is not opened properly before the if case", line)
827                     exit(-1)
828     else:
829             print("FOUND ERROR: A problem appeared while entering the if case",line)
830             exit(-1)
```

Αφού κλείσει η αριστερή παρένθεση, εκτελούμε τις υπόλοιπες διαδικασίες, και καλούμε τις **statements(),** και την **generateQuad('jump, '_','_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στο σύμβολο **{,** και την **elsepart(),** για να ελέγξουμε αν υπάρχουν μια ή παραπάνω συνθήκες **else.**

Αν βρούμε κάποιο λάθος, ενημερώνουμε τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
832 def elsepart():                 #else
833
834         global lexres
835         global line
836
837         if(lexres[0] == else_Token):
838                 lexres = lex()
839                 line = lexres[2]
840
841                 statements()
842
843         return
```

Η **elsepart()** καλείται όταν εντοπίσουμε την λέξη **else**, ή από την **ifStat()**, ώστε να εντοπίσουμε αν υπάρχει, τη δεσμευμένη λέξη **else**.

Αν εντοπίσουμε την **else**, καλούμε την **statements()**.

```
845 def whileStat():                #while case
846 
847         global lexres
848         global line
849 
850         if(lexres[0]== while_Token):                                    #we find the word while
851                 lexres = lex()
852                 line = lexres[2]
853 
854                 if(lexres[0] == leftParenthesis_Token):                  #then we find the expected [ symbol
855                         lexres = lex()
856                         line = lexres[2]
857 
858                         Cquad=nextQuad()
859                         C = condition()
860                         backPatch(C[0], nextQuad())
861 
862                         if(lexres[0] == rightParenthesis_Token):         #then we find the expected ] symbol
863                                 lexres = lex()
864                                 line = lexres[2]
865 
866                                 statements()
867 
868                                 generateQuad('jump', '_', '_', Cquad)     #moves to Cquad
869                                 backPatch(C[1], nextQuad())
870 
871                                 return
872                         else:
873                                 print("FOUND ERROR: The parenthesis on while case has not closed properly", line)
874                                 exit(-1)
875                 else:
876                         print("FOUND ERROR: The parenthesis on while case has not opened properly",line)
877                         exit(-1)
878         else:
879                 print("FOUND ERROR: A problem appeared on the while case", line)
880                 exit(-1)
```

Η **whileStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **while**.

Ακολουθούμε παρόμοια διαδικασία με την μέθοδο **ifStat(),** δηλαδή αποθηκεύουμε το αποτέλεσμα που θα επιστρέψει η **nextQuad()** στην μεταβλητή **Cquad**, και το αποτέλεσμα που θα επιστρέψει η **condition()** στην μεταβλητή **C**.

Καλεί την **generateQuad('jump, '_','_','CQuad')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην **CQuad.**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
882 def switchcaseStat():                # checks the conditions after the word case, and executes statements
883                                     # when finished leaves from switchcase
884     global lexres
885     global line
886
887     if(lexres[0] == switchcase_Token):                              #we find the word swithcase
888         lexres = lex()
889         line = lexres[2]
890         outList=emptyList()
891         while(lexres[0] == case_Token):                              #then we find the expected word case
892             lexres = lex()
893             line = lexres[2]
894             if(lexres[0] == leftParenthesis_Token):                  #then we find the expected [ symbol
895                 lexres = lex()
896                 line = lexres[2]
897                 C = condition()
898                 backPatch(C[0], nextQuad())
899                 if(lexres[0] == rightParenthesis_Token):             #then we find the expected ] symbol
900                     lexres = lex()
901                     line = lexres[2]
902                     statements()
903                     outJump = makeList(nextQuad())
904                     generateQuad('jump', '_', '_', '_')
905                     outList = merge(outList, outJump)
906                     backPatch(C[1], nextQuad())
907                     #return
908                 else:
909                     print("FOUND ERROR: Right parenthesis was not found on forcase", line)
910                     exit(-1)
911             else:
912                 print("FOUND ERROR: Left parenthesis was not found on forcase", line)
913                 exit(-1)
914
915         if(lexres[0] == default_Token):                              #we find the word default
916             lexres = lex()
917             line = lexres[2]
918
919             statements()
920
921             backPatch(outList, nextQuad())
922         else:
923             print("FOUND ERROR: Default has not been started properly on forecase", line)
924             exit(-1)
925     else:
926         print("FOUND ERROR: Forcase does not start properly", line)
927         exit(-1)
```

Η **switchcaseStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **switchcase.**

Καλεί την **generateQuad('jump, '_','_','_)** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην **merge(outList,outJump).**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
929 def forcaseStat():                    # checks the conditions after the word case, and executes statements
930         |       |       |       |     # when finished returns to the start of forcaseStat
931         global lexres
932         global line
933 
934         if(lexres[0] == forcase_Token):                                  #we find the word forcase
935                 lexres = lex()
936                 line = lexres[2]
937                 quad=nextQuad()
938 
939                 while(lexres[0] == case_Token):                          #then we find the expected word case
940                         lexres = lex()
941                         line = lexres[2]
942 
943                         if(lexres[0] == leftParenthesis_Token):          #then we find the expected [ symbol
944                                 lexres = lex()
945                                 line = lexres[2]
946                                 C = condition()
947                                 backPatch(C[0], nextQuad())
948 
949                                 if(lexres[0] == rightParenthesis_Token): #then we find the expected ] symbol
950                                         lexres = lex()
951                                         line = lexres[2]
952 
953                                         statements()
954 
955                                         generateQuad('jump','_','_',quad)   #moves to quad
956                                         backPatch(C[1], nextQuad())
957 
958 
959                                 else:
960                                         print("FOUND ERROR: Right parenthesis was not found on forcase", line)
961                                         exit(-1)
962                         else:
963                                 print("FOUND ERROR: Left parenthesis was not found on forcase", line)
964                                 exit(-1)
965 
966                 if(lexres[0] == default_Token):                          #we find the word default
967                         lexres = lex()
968                         line = lexres[2]
969 
970                         statements()
971                 else:
972                         print("FOUND ERROR: Default has not been started properly on forecase", line)
973                         exit(-1)
974         else:
975                 print("FOUND ERROR: Forcase does not start properly", line)
976                 exit(-1)
```

Η **forcaseStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **forcase.**

Καλεί την **generateQuad('jump, '_','_',quad)** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση **nextQuad().**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
978  def incaseStat():              # checks the conditions after the word case, and executes statements
979  |       |       |       |     # when finished, if at least on statement was executed, returns to the start of incaseStat, else leaves
980          global lexres
981          global line
982
983          if(lexres[0] == incase_Token):                                  #we find the word incase
984  |               lexres = lex()
985  |               line = lexres[2]
986  |
987  |               quad=nextQuad()
988  |               w = newTemp()
989  |               generateQuad(':=','0','_',w)
990  |
991  |               while(lexres[0] == case_Token):                         #then we find the word case
992  |  |                    lexres = lex()
993  |  |                    line = lexres[2]
994  |  |
995  |  |                    if(lexres[0] == leftParenthesis_Token):         #then we find the expected [ symbol
996  |  |  |                         lexres = lex()
997  |  |  |                         line = lexres[2]
998  |  |  |
999  |  |  |                         C = condition()
1000 |  |  |                         backPatch(C[0], nextQuad())
1001 |  |  |
1002 |  |  |                         if(lexres[0] == rightParenthesis_Token):    #then we find the expected ] symbol
1003 |  |  |  |                              lexres = lex()
1004 |  |  |  |                              line = lexres[2]
1005 |  |  |  |
1006 |  |  |  |                              statements()
1007 |  |  |  |
1008 |  |  |  |                              generateQuad(':=','1','_',w)
1009 |  |  |  |                              backPatch(C[1], nextQuad())
1010 |  |  |  |
1011 |  |  |  |                              #return
1012 |  |  |
1013 |  |  |                         else:
1014 |  |  |  |                              print("FOUND ERROR: Right parenthesis was not found on forcase", line)
1015 |  |  |  |                              exit(-1)
1016 |  |                    else:
1017 |  |  |                         print("FOUND ERROR: Left parenthesis was not found on forcase", line)
1018 |  |  |                         exit(-1)
1019 |
1020 |               generateQuad('=',w,'1',quad)
1021
1022         else:
1023 |               print("FOUND ERROR: Forcase does not start properly", line)
1024 |               exit(-1)
```

Η **incaseStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **incase.**

Καλεί την **generateQuad(':=, '0','_',w)** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση **newTemp(),** και αφού καλέσει την statements() καλεί την **generateQuad(':=, '1','_',w).**

Αν έχει εκτελεστεί τουλάχιστον ένα **statement** όταν έχει ολοκληρωθεί η διαδικασία, επιστρέφει στην αρχή της μεθόδου, ειδάλλως φεύγει από την επαναληπτική διαδικασία της **while**.

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1026  def returnStat():              #returns the text inside the parenthesis
1027  |
1028  |       global lexres
1029  |       global line
1030  |
1031  |       if(lexres[0] == return_Token):                              #we find the word return
1032  |       |       lexres = lex()
1033  |       |       line = lexres[2]
1034  |       |
1035  |       |       if(lexres[0] == leftParenthesis_Token):             #then we find the expected [ symbol
1036  |       |       |       lexres = lex()
1037  |       |       |       line = lexres[2]
1038  |       |       |
1039  |       |       |       Eplace = expression()
1040  |       |       |       generateQuad('retv', Eplace, '_', '_')
1041  |       |       |
1042  |       |       |       if(lexres[0] == rightParenthesis_Token):    #then we find the expected ] symbol
1043  |       |       |       |       lexres = lex()
1044  |       |       |       |       line = lexres[2]
1045  |       |       |       |       return
1046  |       |       |       else:
1047  |       |       |       |       print("FOUND ERROR: The parenthesis does not close properly on return",line)
1048  |       |       |       |       exit(-1)
1049  |       |       else:
1050  |       |       |       print("FOUND ERROR: The parenthesis does not close properly on return", line)
1051  |       |       |       exit(-1)
```

Η **returnStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **return.**

Αποθηκεύει την τιμή που επιστρέφει η **return** στην μεταβλητή **Eplace**.

Καλεί την **generateQuad('retv', 'Eplace','_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση επιστροφής του κειμένου εντός της παρένθεσης (return value).

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1053  def callStat():                #calls the identifier
1054
1055          global lexres
1056          global line
1057
1058          if(lexres[0] == call_Token):                                    #we find the word call
1059                  lexres = lex()
1060                  line = lexres[2]
1061
1062                  if(lexres[0] == identifier_Token):                      #then we set the expected identifier
1063                          idName = lexres[1]
1064                          lexres = lex()
1065                          line = lexres[2]
1066
1067                          if(lexres[0] == leftParenthesis_Token):         #then we find the expected [ symbol
1068                                  lexres = lex()
1069                                  line = lexres[2]
1070
1071                                  actualparlist()
1072                                  generateQuad('call', idName, '_', '_')  #we initiate idName function
1073
1074                                  if(lexres[0] == rightParenthesis_Token):        #then we find the expected ] symbol
1075                                          lexres = lex()
1076                                          line = lexres[2]
1077                                          return
1078                                  else:
1079                                          print("FOUND ERROR: The parenthesis does not properly close on call",line)
1080                                          exit(-1)
1081                          else:
1082                                  print("FOUND ERROR: The parenthesis does not properly open on call", line)
1083                                  exit(-1)
1084
1085                  else:
1086                          print("FOUND ERROR: Identifier was not found on call", line)
1087                          exit(-1)
1088          else:
1089                  print("FOUND ERROR: Call does not properly start",line)
1090                  exit(-1)
1091
1092          return
```

Η **callStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **call.**

Καλεί την μέθοδο **actualparlist()**, και στη συνέχεια

Καλεί την **generateQuad('call', idName,'_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση **idName.**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1094  def printStat():                #prints the text in parenthesis
1095
1096          global lexres
1097          global line
1098
1099          if(lexres[0] == print_Token):                                  #we find the word print
1100                  lexres = lex()
1101                  line = lexres[2]
1102
1103                  if(lexres[0] == leftParenthesis_Token):                #then we find the expected [ symbol
1104                          lexres = lex()
1105                          line = lexres[2]
1106
1107                          Eplace = expression()
1108                          generateQuad('out', Eplace, '_', '_')
1109
1110                          if(lexres[0] == rightParenthesis_Token):       #then we find the expected ] symbol
1111                                  lexres = lex()
1112                                  line = lexres[2]
1113
1114                          else:
1115                                  print("FOUND ERROR: The parenthesis does not properly close on print",line)
1116                                  exit(-1)
1117                  else:
1118                          print("FOUND ERROR: The parenthesis does not properly open on print", line)
1119                          exit(-1)
1120          else:
1121                  print("FOUND ERROR: The print does not properly start", line)
1122                  exit(-1)
1123          return
```

Η **printStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **print**.

Αποθηκεύει την τιμή που επιστρέφει η **expression()** στην μεταβλητή **Eplace**.

και στη συνέχεια

Καλεί την **generateQuad('out', Eplace,'_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση καταγραφής του κειμένου εντός της παρένθεσης**.**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1125  def inputStat():              #saves the identifier in parenthesis
1126  |
1127  |       global lexres
1128  |       global line
1129  |
1130  |       if(lexres[0] == input_Token):                                   #we find the word input
1131  |       |       lexres = lex()
1132  |       |       line = lexres[2]
1133  |       |
1134  |       |       if(lexres[0] == leftParenthesis_Token):                 #then we find the expected [ symbol
1135  |       |       |       lexres = lex()
1136  |       |       |       line = lexres[2]
1137  |       |       |
1138  |       |       |       if(lexres[0] == identifier_Token):              #then we set the expected identifier
1139  |       |       |       |       myid = lexres[1]
1140  |       |       |       |       generateQuad('inp',myid,'_','_')
1141  |       |       |       |       lexres = lex()
1142  |       |       |       |       line = lexres[2]
1143  |       |       |       |
1144  |       |       |       |       if(lexres[0] == rightParenthesis_Token):  #then we find the expected ] symbol
1145  |       |       |       |       |       lexres = lex()
1146  |       |       |       |       |       line = lexres[2]
1147  |       |       |       |       |       return
1148  |       |       |       |
1149  |       |       |       |       else:
1150  |       |       |       |       |       print("FOUND ERROR: The parenthesis does not properly close on input",line)
1151  |       |       |       |       |       exit(-1)
1152  |       |       |       else:
1153  |       |       |       |       print("FOUND ERROR: Identifier was not found on input",line)
1154  |       |       |       |       exit(-1)
1155  |       |       else:
1156  |       |       |       print("FOUND ERROR: The parenthesis does not properly open on input", line)
1157  |       |       |       exit(-1)
1158  |       else:
1159  |       |       print("FOUND ERROR: The input does not properly start", line)
1160  |       |       exit(-1)
```

Η **inputStat()** καλείται όταν εντοπίσουμε τη δεσμευμένη λέξη **input.**

Αφού εντοπίσει το όνομα ταυτοποίησης, το αποθηκεύει στην **myid**.

και στη συνέχεια

Καλεί την **generateQuad('inp', myid,'_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση εγγραφής του κειμένου (δηλαδή το όνομα ταυτοποίησης) εντός της παρένθεσης**.**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1162  def actualparlist():              #in case
1163
1164          global lexres
1165          global line
1166
1167          actualparitem()
1168
1169          while(lexres[0] == comma_Token):
1170                  lexres  = lex()
1171                  line = lexres[2]
1172
1173                  actualparitem()
1174
1175          return
```

Η **actualparlist**() καλεί την **actualparitem(),** και όσο εντοπίζει κόμμα, μετά από την ολοκλήρωση ης λειτουργίας της, επαναλαμβάνει την κλήση της.

```
1177  def actualparitem():              #sends the name and value of the identifier
1178
1179          global lexres
1180          global line
1181
1182          if(lexres[0] == in_Token):                      #we find the word in
1183                  lexres = lex()
1184                  line = lexres[2]
1185
1186                  thisExpression = expression()
1187                  generateQuad('par', thisExpression, 'CV', '_')          #sends thisExpression as a price (CV)
1188
1189          elif(lexres[0] == inout_Token):                 #we find the word inout
1190                  lexres = lex()
1191                  line = lexres[2]
1192
1193                  if(lexres[0] == identifier_Token):      #then we set the expected identifier
1194                          name = lexres[1]
1195
1196                          lexres = lex()
1197                          line = lexres[2]
1198
1199                          generateQuad('par', name, 'REF', '_')           #sends name as a reference (REF)
1200
1201                  else:
1202                          print("FOUND ERROR: We await the variable name after the 'inout' ", line)
1203                          exit(-1)
1204
1205          return
```

Η **actualparlist**() στέλνει το όνομα και την τιμή του ταυτοποιητή, μέσω της **generateQuad('par',thisExpression,'CV','_')** και της **generateQuad('par',name,'REF','_')**

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1207  def condition():                #checks the validity of conditions
1208  |
1209  |       global lexres
1210  |       global line
1211  |
1212  |       Ctrue = []
1213  |       Cfalse = []
1214  |       BT1 = boolterm()
1215  |       Ctrue = BT1[0]
1216  |       Cfalse = BT1[1]
1217  |
1218  |       while(lexres[0]==or_Token):                    #we find the word or
1219  |       |       lexres=lex()
1220  |       |       line = lexres[2]
1221  |       |
1222  |       |       backPatch(Cfalse, nextQuad())
1223  |       |
1224  |       |       BT2 = boolterm()
1225  |       |
1226  |       |       Ctrue = merge(Ctrue, BT2[0])
1227  |       |       Cfalse = BT2[1]
1228  |
1229  |       return Ctrue, Cfalse
```

Η **condition()** ελέγχει αν ισχύουν οι συνθήκες που περιέχει η συνθήκη **or**, και τις επιστρέφει το αποτέλεσμα, με τις λίστες **Ctrue**, **Cfalse**.

```
1231  def boolterm():                #for multiple validity checks, we repeat boolfactor() call
1232  |
1233  |       global lexres
1234  |       global line
1235  |
1236  |       BTtrue = []
1237  |       BTfalse = []
1238  |
1239  |       BF1 = boolfactor()
1240  |       BTtrue = BF1[0]
1241  |       BTfalse = BF1[1]
1242  |
1243  |       while(lexres[0]==and_Token):                    #we find the word and
1244  |       |       lexres=lex()
1245  |       |       line = lexres[2]
1246  |       |
1247  |       |       backPatch(BTtrue, nextQuad())
1248  |       |
1249  |       |       BF2 = boolfactor()
1250  |       |
1251  |       |       BTfalse = merge(BTfalse, BF2[1])
1252  |       |       BTtrue = BF2[0]
1253  |       return BTtrue, BTfalse
```

Η **boolterm()** ελέγχει αν ισχύουν οι συνθήκες που περιέχει η συνθήκη **and**, και τις επιστρέφει το αποτέλεσμα, με τις λίστες **Btrue**, **Bfalse**.

```
1255  def boolfactor():
1256
1257          global lexres
1258          global line
1259          BFtrue = []
1260          BFfalse = []
1261          if(lexres[0]==not_Token):                           #we find the word not
1262                  lexres=lex()
1263                  line = lexres[2]
1264                  if(lexres[0]==leftBracket_Token):           #then we find the expected [ symbol
1265                          lexres = lex()
1266                          line = lexres[2]
1267                          C = condition()
1268                          if(lexres[0]==rightBracket_Token):  #then we find the expected ] symbol
1269                                  lexres = lex()
1270                                  line = lexres[2]
1271                                  BFtrue = C[1]
1272                                  BFfalse = C[0]
1273                          else:
1274                                  print("FOUND ERROR: The bracket does not properly close on the boolfactor case ",line)
1275                                  exit(-1)
1276                  else:
1277                          print("FOUND ERROR: We await a bracket opening after the not on boolfactor", line)
1278                          exit(-1)
1279          elif(lexres[0]==leftBracket_Token):                 #we find the [ symbol
1280                  lexres = lex()
1281                  line = lexres[2]
1282                  C = condition()
1283                  if(lexres[0]==rightBracket_Token):          #then we find the expected ] symbol
1284                          lexres = lex()
1285                          line = lexres[2]
1286                          BFtrue = C[0]
1287                          BFfalse = C[1]
1288                  else:
1289                          print("FOUND ERROR: The bracket does not properly close on the boolfactor case", line)
1290                          exit(-1)
1291          else:
1292                  Eplace1 = expression()
1293                  relop = relationalOperation()           #moves to new space if conditions apply
1294                  Eplace2 = expression()
1295                  BFtrue=makeList(nextQuad())
1296                  generateQuad(relop, Eplace1, Eplace2, '_')
1297                  BFfalse=makeList(nextQuad())
1298                  generateQuad('jump', '_', '_', '_')
1299
1300          return BFtrue, BFfalse
```

Η **boolfactor()** ελέγχει αν ισχύουν οι συνθήκες που περιέχει η συνθήκη **not**.

Αν οι συνθήκες ισχύουν, μετακινείται στην επόμενη κατάσταση.

Οι **Eplace1, Eplace2** χρησιμοποιούνται για να διατηρούν την τιμή που επιστρέφει η **expression()**.

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1302  def expression():              #in case of plus or minus, repeat addOperation() call
1303  |
1304  |       global lexres
1305  |       global line
1306  |
1307  |       optionalSign()
1308  |
1309  |       T1place = term()
1310  |
1311  |       while(lexres[0]==plus_Token or lexres[0]==minus_Token):              #we find the + or - symbol
1312  |       |       plusOrMinus = addOperation()
1313  |       |
1314  |       |       T2place = term()
1315  |       |
1316  |       |       w = newTemp()
1317  |       |       generateQuad(plusOrMinus, T1place, T2place, w)
1318  |       |       T1place = w
1319  |
1320  |       Eplace = T1place
1321  |       return Eplace
```

Η **expression()** αρχικά καλεί την optional_sign, ώστε να διαπιστωθεί αν γίνεται πρόσθεση ή αφαίρεση. Με βάση το αποτέλεσμα που λαμβάνει από την **addOperation(),** καλεί την **term()**, και στη συνέχεια καλεί την **generateQuad(plusOrMinus,T1place,T2place,w)** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση πρόσθεσης ή αφαίρεσης.

```
1323  def term():                   #in case of multiply or divide, repeat calladdOperation() call
1324  |
1325  |       global lexres
1326  |       global line
1327  |
1328  |       F1place = factor()
1329  |
1330  |       while(lexres[0]==multiply_Token or lexres[0]==divide_Token):           #we find the * or / symbol
1331  |       |       mulOrDiv = multiplyOperation()
1332  |       |
1333  |       |       F2place = factor()
1334  |       |
1335  |       |       w=newTemp()
1336  |       |       generateQuad(mulOrDiv, F1place, F2place, w)
1337  |       |       F1place = w
1338  |       Tplace =F1place
1339  |       return Tplace
```

Η **term()** με βάση το αποτέλεσμα που λαμβάνει από την **multiplyOperation(),** καλεί την **factor()**, και στη συνέχεια καλεί την **generateQuad(mulOrDiv,F1place,F2place,w)** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση πολλαπλασιασμού ή διαίρεσης.

Χρησιμοποιούμε τις **F1place** και **F2place** ώστε να διατηρούνται οι τιμές που επιστρέφει η **factor()**.

```
1341  def factor():
1342  
1343          global lexres
1344          global line
1345  
1346          if(lexres[0]==number_Token):                        #we set the number
1347                  fact = lexres[1]
1348                  lexres = lex()
1349                  line = lexres[2]
1350  
1351          elif(lexres[0]==leftParenthesis_Token):             #we find the [ symbol
1352                  lexres = lex()
1353                  line = lexres[2]
1354  
1355                  Eplace = expression()
1356                  fact = Eplace
1357  
1358                  if(lexres[0]==rightParenthesis_Token):      #then we find the expected ] symbol
1359                          lexres = lex()
1360                          line = lexres[2]
1361  
1362                  else:
1363                          print("FOUND ERROR: We await the right parenthesis ')' after the expression on FACTOR ",line)
1364                          exit(-1)
1365  
1366          elif(lexres[0]==identifier_Token):                  #we set the identifier
1367                  fact_temp = lexres[1]
1368                  lexres = lex()
1369                  line = lexres[2]
1370                  fact = idtail()
1371  
1372          else:
1373                  print("FOUND ERROR: We await constant or expression or variable on factor",line)
1374                  exit(-1)
1375  
1376          return fact
```

Η **factor()** καλείται στην περίπτωση που αναγνωρίστηκε σύμβολο πολλαπλασιασμού ή διαίρεσης, εντοπίζει τους αριθμούς, για τους οποίους θα γίνουν οι πράξεις.

Προτού διαβάσουμε την επόμενη λεκτική μονάδα αναθέτουμε την τιμή του **lexres[1]** στο **fact**.

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1378  def idtail():
1379  
1380          global lexres
1381          global line
1382          name = lexres[1]
1383  
1384          if(lexres[0] == leftParenthesis_Token ):                #we find the [ symbol
1385                  lexres = lex()
1386                  name = lexres[1]
1387                  line = lexres[2]
1388  
1389                  actualparlist()
1390                  w=newTemp()
1391                  generateQuad('par', w, 'RET', '_')          #sends w as a function's price (RET)
1392                  generateQuad('call', name, '_', '_')
1393  
1394                  if(lexres[0]==rightParenthesis_Token):          #then we find the expected ] symbol
1395                          lexres = lex()
1396                          line = lexres[2]
1397  
1398                          return w
1399                  else:
1400                          print("FOUND ERROR: We want the right parenthesis ')' after the actualparlist on IDTAIL ",line)
1401                          exit(-1)
1402  
1403          else:
1404                  return name
```

Η **idtail()** καλεί την **factor()**, και στη συνέχεια καλεί την **generateQuad('par',w,'RET','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση αποθήκευσης της πράξης, και έπειτα καλεί την **generateQuad('call',name,'_','_')** για να ξεκινήσει την παραγωγή της 4αδας που αντιστοιχεί στην κατάσταση αποθήκευσης του ονόματος της ταυτοποίησης της κατάστασης.

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1406  def optionalSign():
1407  
1408          global lexres
1409          global line
1410  
1411          if(lexres[0] == plus_Token or lexres[0] == minus_Token):        #we find the + or - symbol
1412  
1413                  addOperation()
1414  
1415          return
```

Η **optionalSign()** καλεί την **addOperation(),** αν εντοπίσει + ή -.

```
1417  def relationalOperation():              #identifies the operation
1418
1419          global lexres
1420          global line
1421
1422          if(lexres[0]==equal_Token):                     #we find the = symbol
1423                  relop = lexres[1]
1424                  lexres = lex()
1425                  line = lexres[2]
1426
1427          elif(lexres[0]==lessThan_Token):                #we find the < symbol
1428                  relop = lexres[1]
1429                  lexres = lex()
1430                  line = lexres[2]
1431
1432          elif(lexres[0]==lessOREqual_Token):             #we find the <= symbol
1433                  relop = lexres[1]
1434                  lexres = lex()
1435                  line = lexres[2]
1436
1437          elif(lexres[0]==notEqual_Token):                #we find the <> symbol
1438                  relop = lexres[1]
1439                  lexres = lex()
1440                  line = lexres[2]
1441
1442          elif(lexres[0]== greaterThan_Token):            #we find the > symbol
1443                  relop = lexres[1]
1444                  lexres = lex()
1445                  line = lexres[2]
1446
1447          elif(lexres[0]==greaterOREqual_Token):          #we find the >= symbol
1448                  relop = lexres[1]
1449                  lexres = lex()
1450                  line = lexres[2]
1451
1452          else:
1453                  print("FOUND ERROR:  =  h < h < =  h <> h > =  h >    is missing ",line)
1454                  exit(-1)
1455          return relop
```

Η **relationalOperation()** αναγνωρίζει το σύμβολο σύγκρισης, και επιστρέφει τον κωδικό ταυτοποίησης.

Αν βρει κάποιο λάθος, ενημερώνει τον χρήστη με το κατάλληλο μήνυμα λάθους.

```
1457  def addOperation():                                  # +
1458
1459          global lexres
1460          global line
1461
1462          if(lexres[0]==plus_Token):                   #we find the + symbol
1463                  addOp = lexres[1]
1464                  lexres = lex()
1465                  line = lexres[2]
1466
1467          elif(lexres[0]==minus_Token):                #we find the - symbol
1468                  addOp = lexres[1]
1469                  lexres = lex()
1470                  line = lexres[2]
1471
1472          return addOp
```

Η **addOperation()** επιστρέφει τον κωδικό ταυτοποίησης για το + ή για το -, μετά από την ταυτοποίηση του συμβόλου.

```
1474  def multiplyOperation():                             # *
1475
1476          global lexres
1477          global line
1478
1479          if (lexres[0] == multiply_Token):             #we find the * symbol
1480                  oper = lexres[1]
1481                  lexres = lex()
1482                  line = lexres[2]
1483
1484          elif (lexres[0] == divide_Token):             #we find the / symbol
1485                  oper = lexres[1]
1486                  lexres = lex()
1487                  line = lexres[2]
1488
1489          return oper
```

Η **multiplyOperation()** επιστρέφει τον κωδικό ταυτοποίησης για το * ή για το /, μετά πό την ταυτοποίηση του συμβόλου.

```
1493  def intCode(intF):
1494
1495          for i in range(len(listOfTotalQuads)):
1496                  quad = listOfTotalQuads[i]
1497                  intF.write(str(quad[0]))
1498                  intF.write(":   ")
1499                  intF.write(str(quad[1]))
1500                  intF.write("   ")
1501                  intF.write(str(quad[2]))
1502                  intF.write("   ")
1503                  intF.write(str(quad[3]))
1504                  intF.write("   ")
1505                  intF.write(str(quad[4]))
1506                  intF.write("\n")
```

Η **intCode(intF)** γράφει στο αρχείο **intF**, τα περιεχόμενα της **ListOfTotalQuads.**

```
1509  def cCode():                              #the text for the c file
1510
1511          global listOfTemporaryVariables
1512
1513          if(len(listOfTemporaryVariables)!=0):
1514                  cFile.write("int ")
1515
1516          for i in range(len(listOfTemporaryVariables)):          #Temp_i variables.
1517                  cFile.write(listOfTemporaryVariables[i])
1518
1519                  if(len(listOfTemporaryVariables) == i+1):
1520                          cFile.write(";\n\n\t")
1521                  else:
1522                          cFile.write(",")
1523
1524          for j in range(len(listOfTotalQuads)):
1525                  if(listOfTotalQuads[j][1] == 'begin_block'):
1526                          cFile.write("L_"+str(j+1)+":\n\t")
1527                  elif(listOfTotalQuads[j][1] == ":="):
1528                          cFile.write("L_"+str(j+1)+": "+ listOfTotalQuads[j][4]+"="+listOfTotalQuads[j][2]+";\n\t")
1529                  elif(listOfTotalQuads[j][1] == "+"):
1530                          cFile.write("L_"+str(j+1)+": "+ listOfTotalQuads[j][4]+"="+listOfTotalQuads[j][2]+"+"+listOfTotalQuads[j][3]+";\n\t")
1531                  elif(listOfTotalQuads[j][1] == "-"):
1532                          cFile.write("L_"+str(j+1)+": "+ listOfTotalQuads[j][4]+"="+listOfTotalQuads[j][2]+"-"+listOfTotalQuads[j][3]+";\n\t")
1533                  elif(listOfTotalQuads[j][1] == "*"):
1534                          cFile.write("L_"+str(j+1)+": "+ listOfTotalQuads[j][4]+"="+listOfTotalQuads[j][2]+"*"+listOfTotalQuads[j][3]+";\n\t")
1535                  elif(listOfTotalQuads[j][1] == "/"):
1536                          cFile.write("L_"+str(j+1)+": "+ listOfTotalQuads[j][4]+"="+listOfTotalQuads[j][2]+"/"+listOfTotalQuads[j][3]+";\n\t")
1537                  elif(listOfTotalQuads[j][1] == "jump"):
1538                          cFile.write("L_"+str(j+1)+": "+"goto L_"+str(listOfTotalQuads[j][4])+ ";\n\t")
1539                  elif(listOfTotalQuads[j][1] == "<"):
1540                          cFile.write("L_"+str(j+1)+": "+"if ("+listOfTotalQuads[j][2]+"<"+listOfTotalQuads[j][3]+") goto L_"+str(listOfTotalQuads[j][4])+";\n\t")
1541                  elif(listOfTotalQuads[j][1] == ">"):
1542                          cFile.write("L_"+str(j+1)+": "+"if ("+listOfTotalQuads[j][2]+">"+listOfTotalQuads[j][3]+") goto L_"+str(listOfTotalQuads[j][4])+";\n\t")
1543                  elif(listOfTotalQuads[j][1] == ">="):
1544                          cFile.write("L_"+str(j+1)+": "+"if ("+listOfTotalQuads[j][2]+">="+listOfTotalQuads[j][3]+") goto L_"+str(listOfTotalQuads[j][4])+";\n\t")
1545                  elif(listOfTotalQuads[j][1] == "<="):
1546                          cFile.write("L_"+str(j+1)+": "+"if ("+listOfTotalQuads[j][2]+"<="+listOfTotalQuads[j][3]+") goto L_"+str(listOfTotalQuads[j][4])+";\n\t")
1547                  elif(listOfTotalQuads[j][1] == "<>"):
1548                          cFile.write("L_"+str(j+1)+": "+"if ("+str(listOfTotalQuads[j][2])+"!="+str(listOfTotalQuads[j][3])+") goto L_"+str(listOfTotalQuads[j][4])+";\n\t
1549                  elif(listOfTotalQuads[j][1] == "="):
1550                          cFile.write("L_"+str(j+1)+": "+"if ("+listOfTotalQuads[j][2]+"=="+listOfTotalQuads[j][3]+") goto L_"+str(listOfTotalQuads[j][4])+";\n\t")
1551                  elif(listOfTotalQuads[j][1] == "out"): #print to apotelesma tou expression.
1552                          cFile.write("L_"+str(j+1)+": "+"printf(\""+listOfTotalQuads[j][2]+"= %d\", "+listOfTotalQuads[j][2]+");\n\t")
1553                  elif(listOfTotalQuads[j][1] == 'halt'):
1554                          cFile.write("L_"+str(j+1)+": {}\n\t")
```

Η **cCode()** είναι υπεύθυνη για την εγγραφή κειμένου, μέσα στο αρχείο c.

```
1556  def files():
1557  
1558          global cFile            #intFile.int & cFile.c
1559  
1560          intFile = open('intFile.int', 'w+')             #Open files to write
1561          cFile = open('cFile.c', 'w+')
1562  
1563          cFile.write("int main(){\n\t")
1564  
1565          syntax_an()                             #we activate the syntax method
1566          program()
1567          print("Telos syntax")
1568          intCode(intFile)
1569          cCode()
1570  
1571          cFile.write("\n}")
1572  
1573          cFile.close()                           #Close open file
1574          intFile.close()
1575  files()
```

Η **files()** είναι υπεύθυνη για το άνοιγμα και το κλείσιμο του αρχείου, καθώς και για την ενεργοποίηση του συντακτικού κώδικα.

```
1576
1577  def printListOfTotalQuads():                  #Prints listOfTotalQuads
1578  
1579          for i in range(len(listOfTotalQuads)):
1580                  print (str(listOfTotalQuads[i][0])+" "+str(listOfTotalQuads[i][1])+" "+str(listOfTotalQuads[i][2])+" "+str(listOfTotalQuads[i][3])+" "+str(listOfTotalQuads[i][4]))
1581  printListOfTotalQuads()
1582  
1583  print("No problems appeared")
```

Η **printListOfTotalQuads()** καταγράφει τις 4αδες που περιέχει η λίστα **ListOfTotalQuads**, και αφού ολοκληρωθεί η διαδικασία, ενημερώνει τον χρήστη πως δεν εντοπίστηκαν προβλήματα.