

# ΜΕΤΑΦΡΑΣΤΕΣ

---

## ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΑΣΚΗΣΗ

---



**CutePy**

### ΟΜΑΔΑ:

ΣΠΥΡΙΔΩΝ ΜΟΤΣΕΝΙΓΟΣ ΑΜ: 4426

ΒΑΣΙΛΕΙΟΣ ΜΑΡΟΥΛΗΣ ΑΜ: 4573



# ΠΕΡΙΕΧΟΜΕΝΑ

---

## ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ:

- [Το αυτόματο του λεκτικού αναλυτή μας.](#)
- [Διαχείριση σφαλμάτων.](#)
- [Οι συναρτήσεις του λεκτικού αναλυτή.](#)
- [Η σημαντικότερη συν/ση : `def lex\(\)`.](#)
- [Αναφορά ελέγχου λεκτικού αναλυτή](#)

## ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ:

- [Εισαγωγή στον συντακτικό αναλυτή.](#)
- [Συναρτήσεις συντακτικού αναλυτή.](#)
- [Αναφορά ελέγχου συντακτικού αναλυτή](#)

## ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ:

- [Εισαγωγή στον ενδιάμεσο κώδικα.](#)
- [Αλλαγές στις συν/σεις του συντακτικού που αφορούν αριθμητικές παραστάσεις.](#)
- [Αλλαγές στις συν/σεις του συντακτικού που αφορούν λογικές παραστάσεις.](#)
- [Αλλαγές στις συν/σεις του συντακτικού που αφορούν τις δομές `if` , `while`.](#)
- [Αναφορά ελέγχου ενδιάμεσου κώδικα.](#)

## ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ:

- [Εισαγωγή στον πίνακα συμβόλων.](#)
- [Βοηθητικές συναρτήσεις.](#)
- [Αλλαγές στις συν/σεις του συντακτικού που αφορούν των πίνακα συμβόλων.](#)
- [Αναφορά ελέγχου πίνακα συμβόλων.](#)

## ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ:

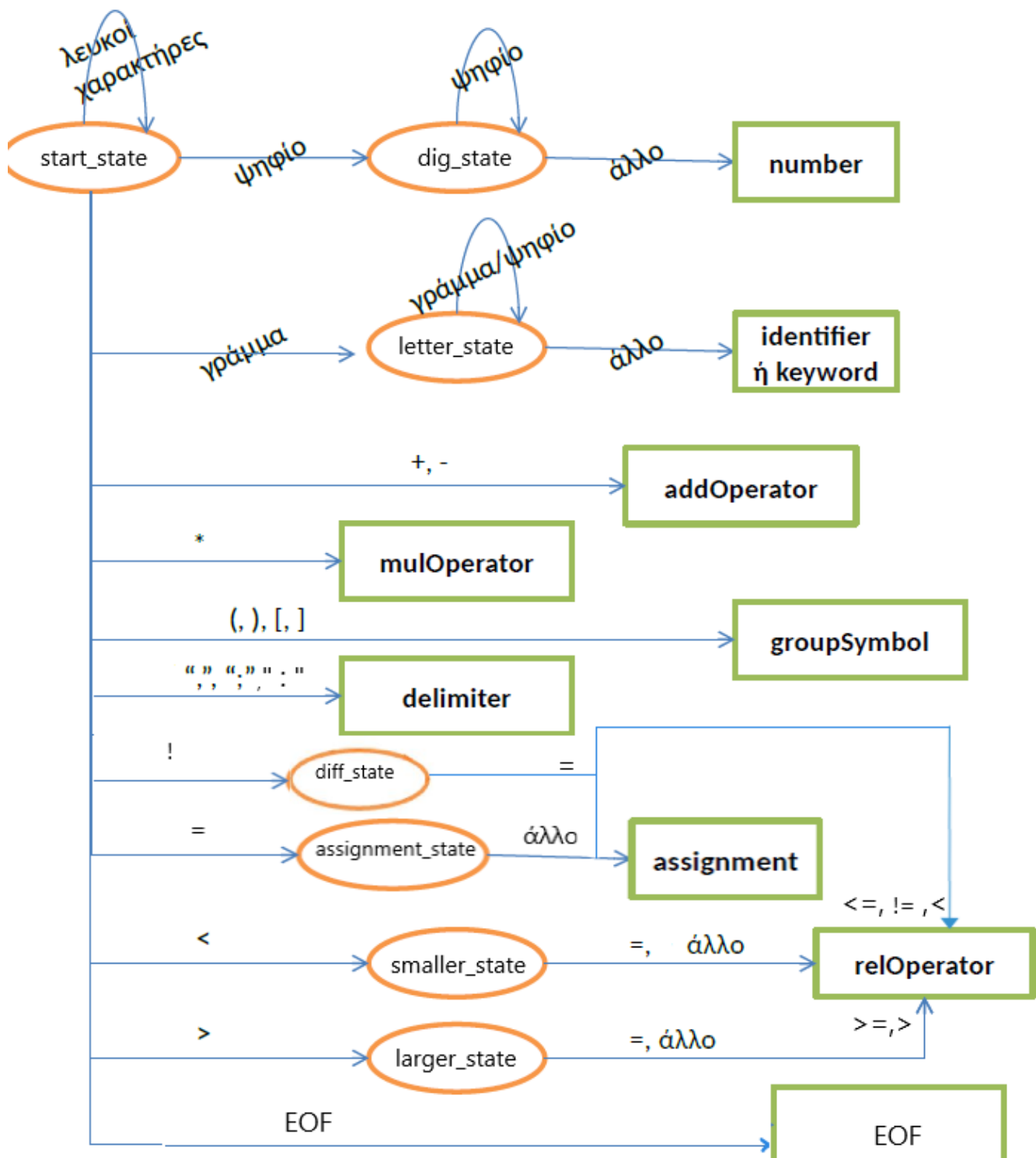
- [Εισαγωγή στον τελικό κώδικα.](#)
- [Βοηθητικές συναρτήσεις.](#)
- [Η κύρια συνάρτηση του τελικού κώδικα](#)
- [Αναφορά ελέγχου τελικού κώδικα.](#)

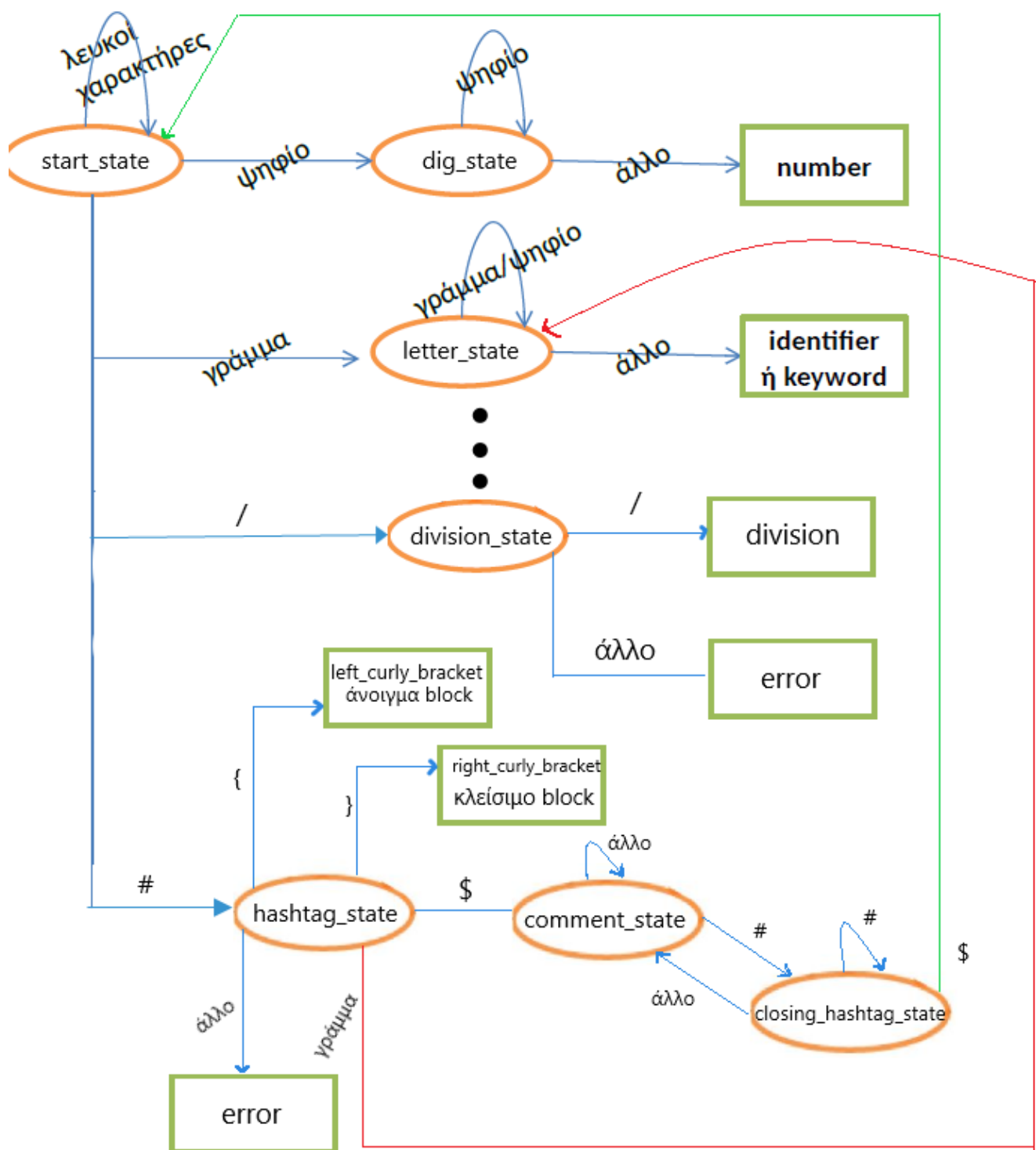
## ΣΥΝΟΨΗ:

- [Σύνοψη.](#)

# ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

Το αυτόματο του λεκτικού αναλυτή μας.

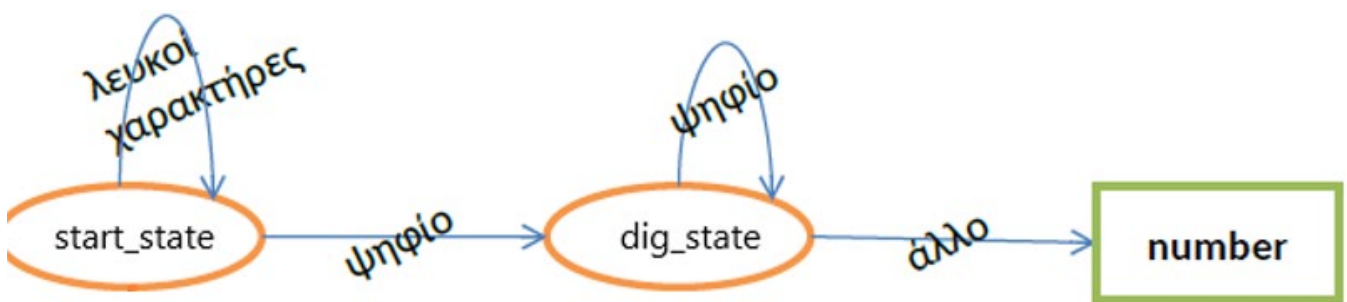




Το αυτόματο αποτελείται από την αρχική κατάσταση, κάποιες ενδιάμεσες και κάποιες τελικές.

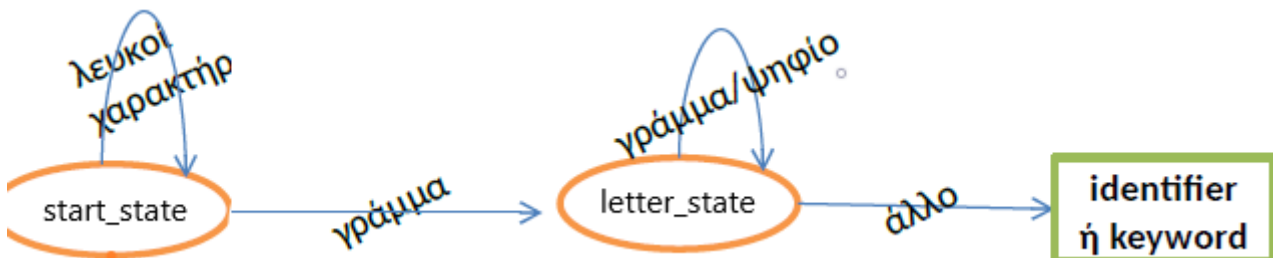
Στο αυτόματο του σχήματος η αρχική κατάσταση είναι η κατάσταση **start\_state**, οι μη τελικές καταστάσεις συμβολίζονται με έλλειψη, ενώ **οι τελικές με παραλληλόγραμμο**.

Με την κλήση του λεκτικού αναλυτή το αυτόματο αρχικοποιείται στην κατάσταση **start\_state**. Εάν στην είσοδο εμφανιστεί κάποιο ψηφίο, τότε το αυτόματο μεταβαίνει στην κατάσταση **dig\_state**, η οποία είναι μη τελική. Όσο στην είσοδο εμφανίζονται ψηφία, τότε για κάθε ψηφίο που διαβάζεται, γίνεται μία μετάβαση, η οποία όμως δεν αλλάζει την κατάσταση του αυτομάτου, αφού είναι μετάβαση από την κατάσταση **dig\_state** στην κατάσταση **dig\_state**. Από την **dig\_state** θα φύγει μόλις έρθει κάτι διαφορετικό, κάτι άλλο που δεν είναι ψηφίο. Όταν συμβεί αυτό το αυτόματο θα μεταβεί στην τελική κατάσταση **number** και θα έχει αναγνωρίσει έναν ακέραιο αριθμό. Όπως φαίνεται στην **εικόνα 1a**.



Εικόνα: 1a.

Εάν στην είσοδο έρθει γράμμα, τότε σύμφωνα με το αυτόματο θα μεταβούμε στην κατάσταση **letter\_state**. Όμοια με την προηγούμενη περίπτωση, το αυτόματο θα παραμείνει στην κατάσταση **letter\_state**, όσο στην είσοδο εμφανίζεται γράμμα ή ψηφίο. Μόλις εμφανιστεί κάτι διαφορετικό, κάτι που δεν είναι ούτε γράμμα ούτε ψηφίο, το αυτόματο μεταβαίνει στην τελική κατάσταση **identifier/keyword**. Όπως φαίνεται στην **εικόνα 1b**.

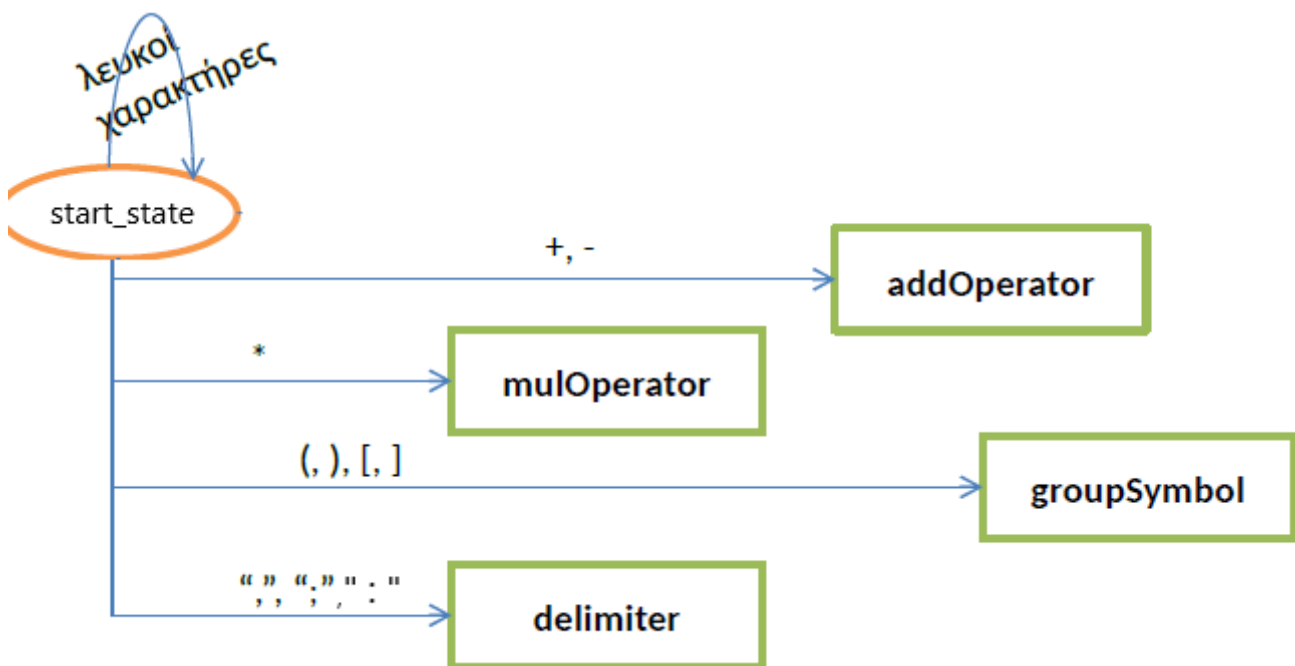


Εικόνα: 1b.

Στο υπόλοιπο αυτόματο τα πράγματα είναι πιο ομαλά.

- ✓ Αν από την αρχική κατάσταση έρθει κάποιος από τους χαρακτήρες +, - τότε θα μεταβούμε αμέσως στην τελική κατάσταση **addOperator**.
- ✓ Αν έρθει το \* τότε πηγαίνουμε στην κατάσταση **mulOperator**.
- ✓ Αν έρθει κάποιο από τα (, ), [, ], τότε θα μεταβούμε στην τελική κατάσταση **groupSymbol**
- ✓ Αν έρθει ένα από τα “, ; : “ τότε καταλήγουμε στην **delimiter**.

Όπως φαίνεται στην **εικόνα 1c**.



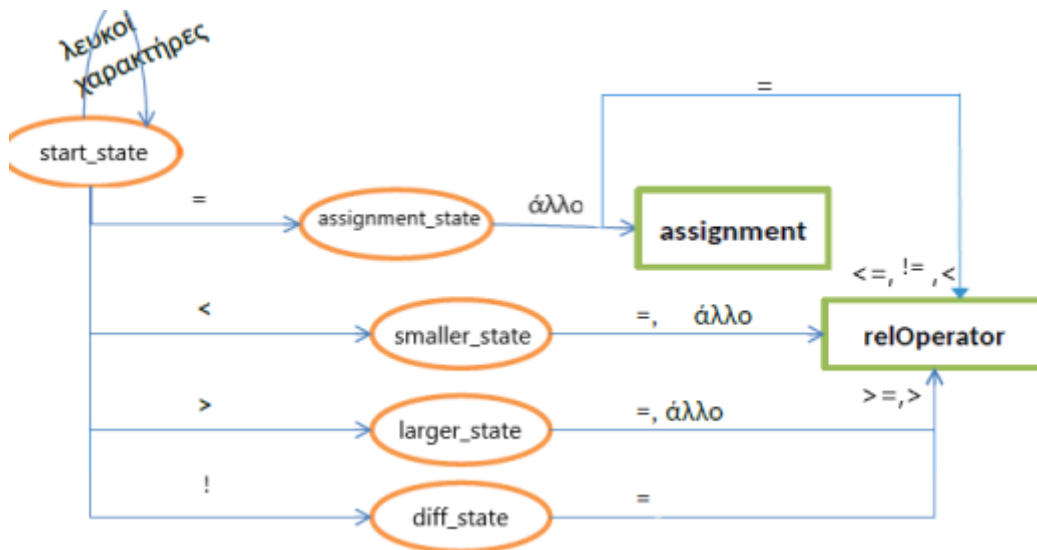
**Εικόνα: 1c.**

Ξεχωριστή κατηγορία αποτελεί το σύμβολο εκχώρησης, το οποίο απαιτεί μία μη τελική κατάσταση την **assignment\_state** πριν φτάσει στην τελική **assignment**, αφού αρχικά έρχεται στην είσοδο το '=' εάν ακολουθεί οτιδήποτε άλλο τότε μεταβαίνουμε στην τελική κατάσταση assignment, αλλιώς αν ακολουθεί δεύτερο '=' τότε μεταβαίνουμε στην τελική κατάσταση relOperator αφού το '==' είναι ο λογικός τελεστής της ισότητας στην γλώσσα CutePy.

Όπως φαίνεται στην **εικόνα 1d**.

**Με τους λογικούς τελεστές, πρέπει να είμαστε προσεκτικοί.** Αν αναγνωρίσουμε, για παράδειγμα, το σύμβολο  $<$ , αυτό δεν σημαίνει ότι μπορούμε να πάμε με ασφάλεια σε τελική κατάσταση, διότι είναι πιθανό να ακολουθεί το σύμβολο  $=$  οπότε η λεκτική μονάδα που θα πρέπει να αναγνωριστεί θα είναι  $<=$ . Όπως φαίνεται στην **εικόνα 1d**.

Μάλιστα, στην περίπτωση αυτή θα πρέπει να προσέξουμε και κάτι ακόμα. Ενώ στις περισσότερες περιπτώσεις, όποιο σύμβολο διαβάζουμε το ενσωματώνουμε σε κάποια λεκτική μονάδα, εδώ για να καταλάβουμε αν το σύμβολο που αναγνωρίσαμε ήταν το  $<$  και όχι το  $<=$ , **θα πρέπει να διαβάσουμε έναν χαρακτήρα** ο οποίος ή είναι λευκός χαρακτήρας ή ανήκει στην επόμενη λεκτική μονάδα. Αν είναι λευκός χαρακτήρας δεν δημιουργείται πρόβλημα. **Αν όμως ανήκει στην επόμενη λεκτική μονάδα πρέπει να είμαστε προσεκτικοί ώστε να μην τον χάσουμε. Θα πρέπει να το επιστρέψουμε.**



**Εικόνα: 1d.**

Εκτός από τους λογικούς τελεστές, το ίδιο συμβαίνει στην αναγνώριση των κατηγοριών identifier, keyword και number, αφού και εκεί, για να αναγνωρίσουμε μία λεκτική μονάδα, είμαστε υποχρεωμένοι να δούμε τον επόμενο σε χαρακτήρα που δεν ανήκει στη λεκτική μονάδα, έτσι ώστε να γνωρίζουμε ότι η υπό αναγνώριση λεκτική μονάδα ολοκληρώθηκε.

Όπως φαίνεται στην **εικόνα 1e**.

```

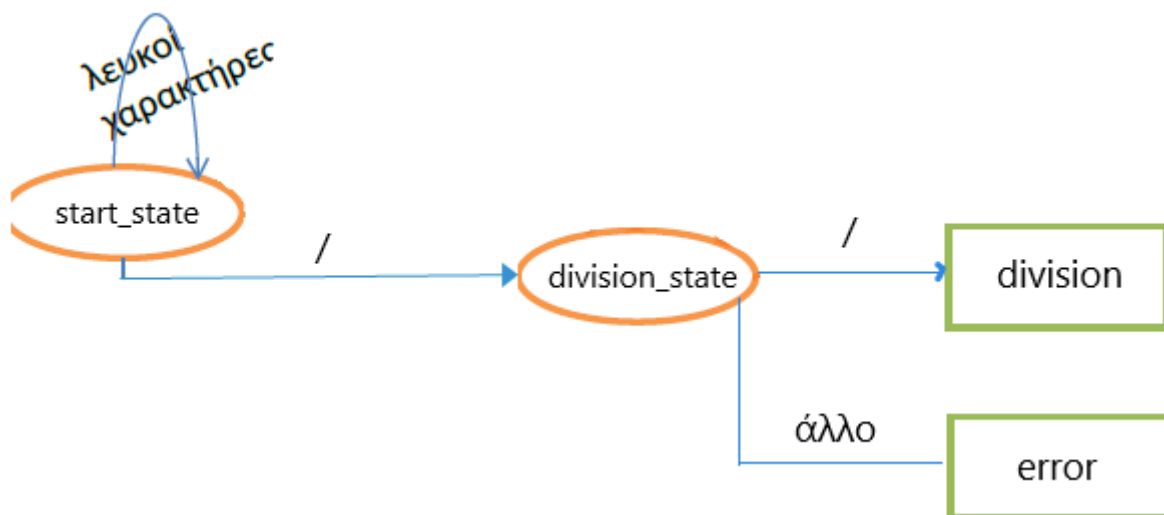
233
234     if( current_state == id or
235        current_state == number or
236        current_state == smaller or
237        current_state == larger or
238        current_state == assignment):
239
240         if (input_char == '\n'):
241             linenummer -= 1
242         input_char=file.seek(file.tell()-1,0)
243
244     recognized_string = recognized_string[:-1]

```

Εικόνα: 1e.

Στην συνέχεια , αν στην είσοδο έρθει τι σύμβολο ‘ / ’ μεταβαίνουμε σε μια μη τελική κατάσταση που ονομάζεται **division\_state**. Εάν το επόμενο σύμβολο είναι και πάλι ‘ / ’ τότε μεταβαίνουμε στην τελική κατάσταση **division**.

Όπως φαίνεται στην **εικόνα 1f** .



Εικόνα: 1f.

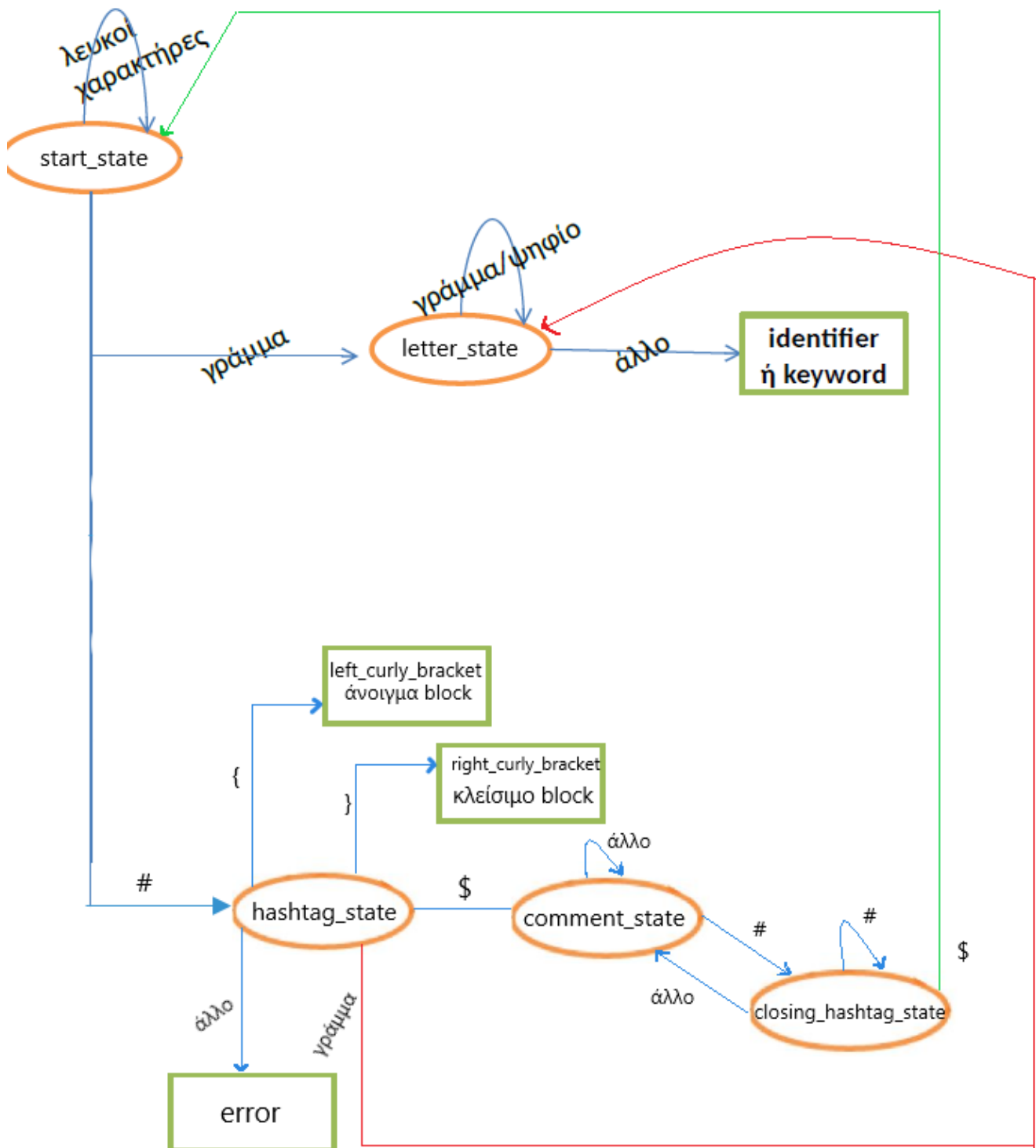
\*Για τα **errors** θα αναφερθούμε παρακάτω. ----> **Διαχείριση σφαλμάτων.**



Εάν στην είσοδο δούμε το hashtag ' # ' τότε μεταβαίνουμε στην κατάσταση **hashtag\_state** και υστέρα διαβάζουμε τον επόμενο χαρακτήρα που μπορεί να είναι:

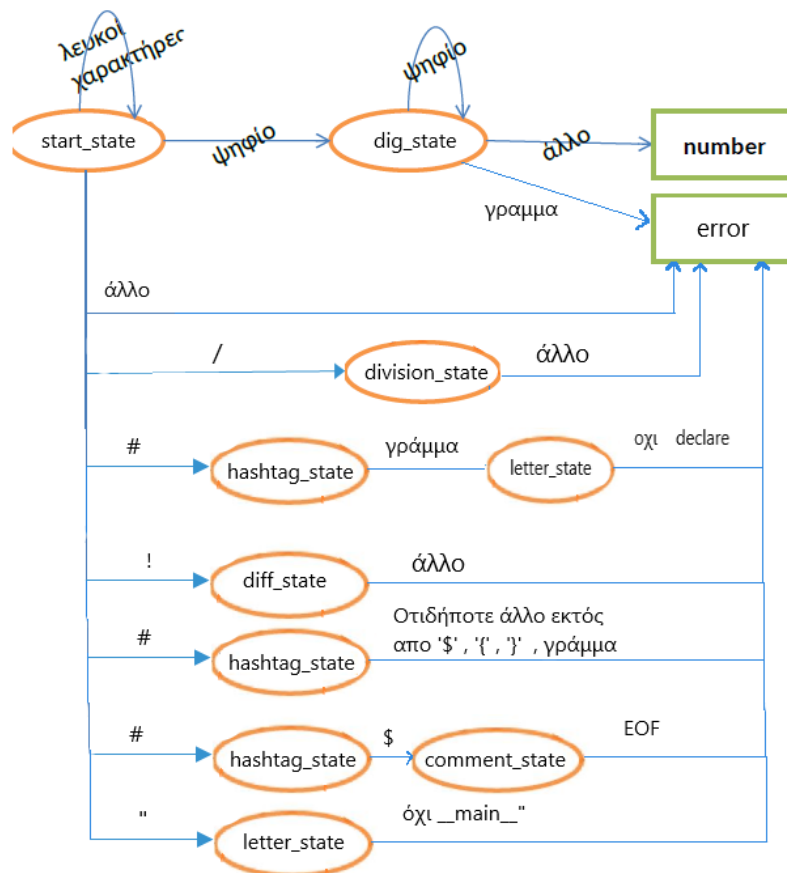
- ✓ ' { ' και έτσι θα μετάβουμε στην τελική κατάσταση **left\_curly\_bracket** που ισοδυναμεί με άνοιγμα ενός block.
- ✓ ' } ' και έτσι θα μετάβουμε στην τελική κατάσταση **right\_curly\_bracket** που ισοδυναμεί με κλείσιμο ενός block.
- ✓ ' γράμμα ' και έτσι θα μεταβούμε στην μη τελική κατάσταση **letter\_state** αυτό γιατί το declare ξεκινά με # .
- ✓ ' \$ ' και έτσι θα μεταβούμε στην **μη** τελική κατάσταση **comment\_state**.
  - Όσο είμαι στην **comment\_state** μπορώ να δω 'οτιδήποτε άλλο' και παραμένω στην ίδια κατάσταση ή να δω
  - ' # ' και να **μεταβώ** στην μη τελική **κατάσταση closing\_hashtag\_state** όπου εκεί θα περιμένω το ' \$ ' για να μεταβώ στην κατάσταση **start\_state**. Αν όμως δω κάτι άλλο τότε πρέπει να επιστρέψω πίσω στην **comment\_state**.

Όπως φαίνεται στην **εικόνα 1g**.



Εικόνα: 1g.

# Διαχείριση σφαλμάτων.



Έχουμε κατηγοριοποιήσει τις περιπτώσεις των λαθών που μπορεί να συμβούν για να ενημερώσουμε τον χρήστη με τα αντίστοιχα μηνύματα όπως φαίνεται στην διπλανή εικόνα του κώδικα μας.

```
81 #errors
82 error_underscore = -400
83 error_digitFollowedByLetter = -401
84 error_integerOutOfBounds = -402
85 error_tooBigWord = -403
86 error_unacceptableSymbol = -404
87 error_soloHashtag = -405
88 error_CommentMissingHashtag = -406
89 error_CommentEOF = -407
90 error_Division = -408
91 error_leftCurly = -409
92 error_rightCurly = -410
93 error_exclMark = -411
94 error_hashdec = -412
95 error_quotes = -413
```

- **error\_underscore** : Όταν από την κατάσταση start\_state είσοδο έρθει η κάτω παύλα.

- **error\_digitFolowedByLetter** : Όταν βρίσκομαι στην κατάσταση dig\_state έχω διαβάσει δηλαδή έναν αριθμό και διαβάζω γράμμα.
- **error\_integerOutOfBounds** : Ο αριθμός είναι εκτός διαστήματος  $[-(2^{32}-1), 2^{32}-1]$ .
- **error\_tooBigWord** : Η λέξη έχει μήκος μεγαλύτερο από 30 χαρακτήρες.
- **error\_unacceptableSymbol** : Όταν είμαι στην αρχική κατάσταση και δω οποιοδήποτε άλλο σύμβολο που δεν ανήκει στην γλώσσα.
- **error\_soloHashtag** : Όταν είμαι στην αρχική κατάσταση έρχεται hashtag και δεν ακολουθείται , από άγκιστρα , από δολάρια.
- **error\_CommentMissingHashtag** : Όταν από την αρχική κατάσταση δούμε κατευθείαν δολάριο για άνοιγμα σχολίων χωρίς όμως να έχω hashtag.
- **error\_CommentEOF** : Φτάνουμε στο τέλος του αρχείου χωρίς να έχουν κλείσει τα σχόλια.
- **error\_Division** : Όταν έχουμε δει το ' / ' και έρχεται κάτι άλλο.
- **error\_leftCurly** : Βλέπουμε το ' { ' χωρίς να έχουμε δει hashtag.
- **error\_rightCurly** : Βλέπουμε το ' } ' χωρίς να έχουμε δει hashtag.
- **error\_exclMark** : Όταν βλέπου '!' και δεν ακολουθείται από '='.
- **error\_hashdec**: Κάποια λέξη ξεκινά με '#' (εκτος του declare)
- **error\_quotes**: Κάποια λέξη ξεκινά με " εκτος του ("\_\_main\_\_")

# Οι συναρτήσεις του λεκτικού αναλυτή.

---

def lex()	Βρίσκει τα token δηλαδή τις λεκτικές μονάδες από τις οποίες αποτελείται το πρόγραμμα εισόδου. <b>Ελέγχει χαρακτήρα - χαρακτήρα</b> και ανάλογα μεταβαίνει στις κατάλληλες καταστάσεις. Τέλος επιστρέφει το token της λεκτικής μονάδας που αναγνώρισε.
def printErrors(current_state)	Βρίσκει το error και τυπώνει το κατάλληλο μήνυμα στην οθόνη.
def makeFamily(current_state)	Βρίσκει την οικογένεια της λεκτικής ομάδας που έχουμε αναγνωρίσει. Και το επιστρέφει για να τυπωθεί.
def recognizeKeywords(current_state)	Βρίσκει ποια λέξη κλειδί είναι η λεκτική μονάδα που αναγνωρίσαμε.
def checkingOutOfBounds(current_state)	Ελέγχει αν ο αριθμός που αναγνωρίσαμε είναι εκτός του διαστήματος $-2^{32}, +2^{32}$ .

# Η σημαντικότερη συν/ση : def lex().

Η συνάρτηση def lex() όταν την καλούμε , ελέγχει χαρακτήρα – χαρακτήρα το αρχείο που δόθηκε σαν είσοδο και επιστρέφει ένα token.

Έτσι επαναληπτικά εάν δεν έχουμε βρει token ελέγχουμε τον χαρακτήρα , μεταβαίνουμε στην κατάλληλη κατάσταση σύμφωνα με αυτό που διαβάσαμε και επαναληπτικά διαβάζουμε και ελέγχουμε τον επόμενο χαρακτήρα μέχρι να βρεθούμε σε μια τελική κατάσταση (token) .

```
163 def lex():
164     global line
165     global recognized_string
166     global current_state
167     recognized_string = ''
168     current_state = start_state
169     linenummer = line
170     token = []
171
172     #Trexw oso den briskw token
173     while(current_state >= 0 and current_state <= 10):
174         input_char = file.read(1)
```

Αφού ελέγχουν οι χαρακτήρες και γίνουν οι κατάλληλες μεταβάσεις στο αυτόματο , η συνάρτηση μου επιστρέφει το

Token που αποτελείται από:

- Την **οικογένεια** στην οποία ανήκει η λέξη που βρήκαμε.
- Την **λέξη**.
- Την **γραμμή**.
- Τον **κωδικό** της λέξης.

```
246     makeFamily(current_state)
247     token.append(family)
248     token.append(recognized_string)
249     token.append(linenummer)
250     token.append(current_state)
251     printErrors(current_state)
252     line=linenummer
253
254     return token
```

Εάν δεν βρεθεί κάποιο Token τότε μπορεί να υπάρχει λεκτικό λάθος , και έτσι η συνάρτηση μας θα τυπώσει το κατάλληλο μήνυμα λάθους.

Επιπρόσθετα πραγματοποιούνται κάποιοι έλεγχοι όπως το διάστημα των ακέραιων όπως περιγράψαμε στον προηγούμενο πίνακα.

# Αναφορά ελέγχου λεκτικού αναλυτή

```
def main_factorial():
#{
    #$ declarations #$
    #declare x
    #declare i,fact

    #$ body of main_factorial #$
    x = int(input());
    fact = 1;
    i = 1;
    while (i<=x):
    #{
        fact = fact * i;
        i = i + 1;
    #}
    print(fact);
#}
```

Επιβεβαιώνουμε την σωστή λειτουργία του λεκτικού αναλυτή μας , δίνοντας ως είσοδο το πρόγραμμα **factorial.cpy** που μας δόθηκε μαζί με την εκφώνηση της παρούσας εργαστηριακής άσκησης.

Όπως φαίνεται στην **εικόνα Α**.

```
1 def main_factorialERROR():
2   ##
3
4   %
5   helloworldhelloworldhelloworld
6   #hello
7   #declare i,x
8   _world
9   x ! 5
10  $
11  fact = 1
12  i = 1;
13  wh$le (i!=x):
14  #{
15  i = i / 5
16  #}
17 #}
18
19 #$ Hello world
20
```

Επίσης διαφοροποιήσαμε το αρχείο factorial , δημιουργώντας errors για να ελέγξουμε και την εκτύπωση των λαθών .

Έτσι όπως φαίνεται στην **εικόνα Β** παίρνουμε σωστά , τα κατάλληλα μηνύματα .

```

C:\Users\USER\Desktop\testing>python lectical.py factorial.cpy
['Family: keyword', 'def', 511, 1]
['Family: id', 'main_factorial', 600, 1]
['Family: groupSymbol', '(', 300, 1]
['Family: groupSymbol', ')', 301, 1]
['Family: delimiter', ':', 400, 1]
['Family: groupSymbol', '#{', 304, 2]
['Family: keyword', '#declare', 512, 4]
['Family: id', 'x', 600, 4]
['Family: keyword', '#declare', 512, 5]
['Family: id', 'i', 600, 5]
['Family: delimiter', ',', 401, 5]
['Family: id', 'fact', 600, 5]
['Family: id', 'x', 600, 8]
['Family: assignment', '=', 602, 8]
['Family: keyword', 'int', 502, 8]
['Family: groupSymbol', '(', 300, 8]
['Family: keyword', 'input', 501, 8]
['Family: groupSymbol', '(', 300, 8]
['Family: groupSymbol', ')', 301, 8]
['Family: groupSymbol', ')', 301, 8]
['Family: delimiter', ';', 402, 8]
['Family: id', 'fact', 600, 9]
['Family: assignment', '=', 602, 9]
['Family: number', '1', 601, 9]
['Family: delimiter', ';', 402, 9]
['Family: id', 'i', 600, 10]
['Family: assignment', '=', 602, 10]
['Family: number', '1', 601, 10]
['Family: delimiter', ';', 402, 10]
['Family: keyword', 'while', 507, 11]
['Family: groupSymbol', '(', 300, 11]
['Family: id', 'i', 600, 11]
['Family: relOperator', '<=', 101, 11]
['Family: id', 'x', 600, 11]
['Family: groupSymbol', ')', 301, 11]
['Family: delimiter', ':', 400, 11]
['Family: groupSymbol', '#{', 304, 12]
['Family: id', 'fact', 600, 13]
['Family: assignment', '=', 602, 13]
['Family: id', 'fact', 600, 13]
['Family: mulOperator', '*', 202, 13]
['Family: id', 'i', 600, 13]
['Family: delimiter', ';', 402, 13]
['Family: id', 'i', 600, 14]
['Family: assignment', '=', 602, 14]
['Family: id', 'i', 600, 14]
['Family: addOperator', '+', 200, 14]
['Family: number', '1', 601, 14]
['Family: delimiter', ';', 402, 14]
['Family: groupSymbol', '#}', 305, 15]
['Family: keyword', 'print', 513, 16]
['Family: groupSymbol', '(', 300, 16]
['Family: id', 'fact', 600, 16]
['Family: groupSymbol', ')', 301, 16]
['Family: delimiter', ';', 402, 16]
['Family: groupSymbol', '#}', 305, 17]
['Family: keyword', 'if', 503, 20]
['Family: keyword', '__name__', 506, 20]
['Family: relOperator', '==', 104, 20]
['Family: groupSymbol', '"', 403, 20]
['Family: keyword', '__main__', 500, 20]
['Family: groupSymbol', '"', 403, 20]
['Family: delimiter', ':', 400, 20]
['Family: id', 'main_factorial', 600, 22]
['Family: groupSymbol', '(', 300, 22]
['Family: groupSymbol', ')', 301, 22]
['Family: delimiter', ';', 402, 22]
[604, '', 604, 22]

```

Εικόνα Α.



Όπως φαίνεται και παραπάνω ο λεκτικός αναλυτής δουλεύει σωστά και αυτό διότι όπως φαίνεται βγάζει τα σωστά token.

Αρχικά διαβάζει τον πρώτο χαρακτήρα που βρίσκει στο αρχείο που είναι ένα γράμμα και έτσι μεταβαίνει στην κατάσταση `letter_state` παραμένει εκεί όσο βλέπει γράμματα και αφού ολοκληρωθεί και διαβάσει ένα κενό καταλαβαίνει ότι πρόκειται για τη λέξη `def` που είναι ένα keyword. Επαναληπτικά διαβάζει τους επόμενους χαρακτήρες και έτσι βρίσκει το όνομα της συνάρτησης `main_factorial` και το επιστρέφει.

Στη συνέχεια διαβάζει μια αριστερή παρένθεση και μεταβαίνει απευθείας σε τελική κατάσταση οπότε μου επιστρέφει αυτή την παρένθεση το ίδιο συμβαίνει και με το διάβασμα της δεξιάς παρένθεσης. Έπειτα διαβάζει μια άνω κάτω και την επιστρέφει.

Έπειτα διαβάζει το `hashtag` και μεταβαίνει στην κατάσταση `hashtag_state` όπου διαβάζει τον επόμενο χαρακτήρα και είναι μια αριστερή αγκύλη έτσι μου επιστρέφει το token `'#{'`.

Γενικότερα αυτή είναι η φιλοσοφία γύρω από τη λειτουργία του λεκτικού αναλυτή διαβάζει κάθε φορά τον επόμενο χαρακτήρα και είτε μεταβαίνει σε προσωρινές καταστάσεις είτε μεταβαίνει σε τελικές καταστάσεις όπου επιστρέφει τα κατάλληλα token.

Ιδιαιτερότητα υπάρχει όταν πρέπει να διαβαστεί και ο επόμενος χαρακτήρας ώστε να αποφασίσει για το σωστό token, (καταστάσεις που αφορούν αριθμητικούς τελεστές κ.α.) αυτό φαίνεται στη γραμμή 11 του προγράμματος που δόθηκε σαν είσοδο στο λεκτικό μας αναλυτή όπου πρώτα βρίσκει το σύμβολο του μικρότερου και τότε ελέγχει τον επόμενο χαρακτήρα ώστε να επιστρέψει είτε μικρότερο είτε μικρότερο ή ίσο στην περίπτωση μας θα ακολουθείται ίσο οπότε μας επιστρέφει σωστά το token `'<='`. Και στη συνέχεια κάνει **οπισθοδρόμηση**.

Φυσικά και τα σχόλια δεν θα πρέπει να τυπώνονται σαν λεκτικές μονάδες όπως δεν πρέπει να τυπώνονται τα κενά και οι αλλαγές γραμμών όπως πολύ σωστά δεν τυπώνει ο λεκτικός αναλυτής μας.

Φυσικά δεν τυπώνει κάποιο μήνυμα λάθους αφού έχουμε φροντίσει εμείς για αυτό, αφού δεν περιέχει λάθη το πρόγραμμα. Παρακάτω στην **εικόνα Β** έχουμε τοποθετήσει λάθη ώστε να ελέγξουμε και τη σωστή ανίχνευση λαθών από το λεκτικό αναλυτή μας.

```

C:\Users\USER\Desktop\testing>python lec_syn.py factorialerror.cpy
['Family: keyword', 'def', 511, 1]
['Family: id', 'main_factorialERROR', 600, 1]
['Family: groupSymbol', '(', 300, 1]
['Family: groupSymbol', ')', 301, 1]
['Family: delimiter', ':', 400, 1]
LEKTIKO error: monaxiko hashtag (#), line : 2
['Family: keyword', '##', -405, 2]
LEKTIKO error: monaxiko hashtag (#), line : 3
['Family: keyword', '#\n', -405, 3]
LEKTIKO error: To simbolo den iparxei sto alfabito, line : 4
['Family: keyword', '%', -404, 4]
LEKTIKO error: H leksi upervainei tous 30 xaraktires, line : 6
['Family: keyword', 'helloworldhelloworldhelloworld', -403, 6]
LEKTIKO error: arxizei lexi me '#' line : 6
['Family: keyword', '#hello', -412, 6]
['Family: keyword', '#declare', 512, 7]
['Family: id', 'i', 600, 7]
['Family: delimiter', ',', 401, 7]
['Family: id', 'x', 600, 7]
LEKTIKO error: H leksi arxizei me '_', line : 8
['Family: keyword', '_world', -400, 8]
['Family: id', 'x', 600, 9]
LEKTIKO error: thaumastiko '!' mono tou , line : 9
['Family: keyword', '! ', -411, 9]
['Family: number', '5', 601, 9]
LEKTIKO error: monaxiko dollarario ($), line : 10
['Family: keyword', '$', -406, 10]
['Family: id', 'fact', 600, 11]
['Family: assignment', '=', 602, 11]
['Family: number', '1', 601, 11]
['Family: id', 'i', 600, 12]
['Family: assignment', '=', 602, 12]
['Family: number', '1', 601, 12]
['Family: delimiter', ';', 402, 12]
['Family: id', 'wh', 600, 13]
LEKTIKO error: monaxiko dollarario ($), line : 13
['Family: keyword', '$', -406, 13]
['Family: id', 'le', 600, 13]
['Family: groupSymbol', '(', 300, 13]
['Family: id', 'i', 600, 13]
['Family: relOperator', '!=', 105, 13]
['Family: id', 'x', 600, 13]
['Family: groupSymbol', ')', 301, 13]
['Family: delimiter', ':', 400, 13]
['Family: groupSymbol', '#{', 304, 14]
['Family: id', 'i', 600, 15]
['Family: assignment', '=', 602, 15]
['Family: id', 'i', 600, 15]
LEKTIKO error: Den exw brei to deutero slash (/) gia na einai diairesi, line : 15
['Family: keyword', '/ ', -408, 15]
['Family: number', '5', 601, 15]
['Family: groupSymbol', '#}', 305, 16]
['Family: groupSymbol', '#}', 305, 17]
LEKTIKO error: To arxio teleiwse kai den ekleisan ta sxolia, line : 20
['Family: keyword', '', -407, 20]
[604, '', 604, 20]

```

## Εικόνα Β

Σε αυτή την περίπτωση έχουμε δημιουργήσει αρχείο όπως περιγράφεται στην **σελίδα 15** που περιέχει όλα τα λάθη που μπορεί να ανιχνεύσει ο λεκτικός αναλυτής μας και να τα τυπώσει. Αρχικά βρίσκει τα # που δεν ακολουθούνται από { ή } και τυπώνει το κατάλληλο μήνυμα λάθους έπειτα ξεκινώντας από την αρχική κατάσταση αναγνωρίζει το σύμβολο % όπου δεν περιέχεται στο αλφάβητο της γλώσσας και έτσι τυπώνει το κατάλληλο μήνυμα.

Γενικά όπως είδαμε και παραπάνω στο αυτόματο γίνεται διαχείριση λαθών και έτσι όταν ο λεκτικός αναλυτής περιμένει να δει κάτι άλλο από αυτό που αναγνωρίζει τότε πηγαίνει σε κατάσταση error και τυπώνει το κατάλληλο μήνυμα. Αυτό γίνεται επαναληπτικά μέχρι το τέλος του αρχείου υπάρχουν πολλά μηνύματα λάθους ώστε να είναι κατατοπιστικά για το χρήστη.

# Εισαγωγή στον συντακτικό αναλυτή.

Τη φάση της λεκτικής ανάλυσης ακολουθεί η φάση της συντακτικής ανάλυσης. Κατά τη συντακτική ανάλυση ελέγχεται εάν η **ακολουθία** των λεκτικών μονάδων που σχηματίζεται από τον λεκτικό αναλυτή, **αποτελεί μία νόμιμη ακολουθία με βάση τη γραμματική της γλώσσας**. Όποια ακολουθία δεν αναγνωρίζεται από τη γραμματική, αποτελεί μη νόμιμο κώδικα και οδηγεί στον εντοπισμό συντακτικού σφάλματος.

Η γραμματική η οποία χρησιμοποιείται είναι μία γραμματική χωρίς συμφραζόμενα.

Έτσι οι συναρτήσεις που ακολουθούν, δημιουργήθηκαν με βάση τους κανόνες της γραμματικής που μας δόθηκε.

## Συναρτήσεις συντακτικού αναλυτή.

Η τεχνική σύμφωνα με την οποία θα μεταβούμε από την περιγραφή με τη μορφή γραμματικής στον κώδικα λέγεται τεχνική της υλοποίησης με τη **μέθοδο της αναδρομικής κατάβασης**.

Η μετατροπή γίνεται σύμφωνα με τον παρακάτω αλγόριθμο, ο οποίος βρίσκει εφαρμογή σε γραμματικές χωρίς συμφραζόμενα, κατάλληλες για υλοποίηση με αναδρομική κατάβαση:

- Για κάθε κανόνα της γραμματικής υλοποιούμε μία συνάρτηση. Δίνουμε στη συνάρτηση το ίδιο όνομα με τον κανόνα η συνάρτηση αυτή δεν χρειάζεται να παίρνει κάτι σαν όρισμα ή να επιστέφει κάποιο αποτέλεσμα στη συνάρτηση που την κάλεσε.
- Γράφουμε τον κώδικα που αντιστοιχεί στο δεξί μέλος του κανόνα.
  - ✓ Για κάθε **μη τερματικό σύμβολο**, καλούμε την **αντίστοιχη συνάρτηση** που έχουμε υλοποιήσει.
  - ✓ Για **κάθε τερματικό σύμβολο** ελέγχουμε ότι πράγματι το τερματικό σύμβολο αυτό συμπίπτει με την επόμενη λεκτική μονάδα στην είσοδο.
  - ❖ Αν πράγματι το τερματικό σύμβολο που συναντάμε στη γραμματική συμπίπτει με την επόμενη λεκτική μονάδα στην είσοδο, τότε **καταναλώνουμε** τη λεκτική μονάδα, θεωρούμε την αναγνώριση μέχρι το σημείο αυτό επιτυχής και προχωρούμε στην αναγνώριση της επόμενης λεκτικής μονάδας.

❖ Αν το τερματικό σύμβολο που συναντάμε στη γραμματική δεν **συμπίπτει** με την επόμενη λεκτική μονάδα στην είσοδο τότε:

- Αν μία από τις εναλλακτικές που δίνει η γραμματική είναι το κενό δεν εμφανίζουμε μήνυμα σφάλματος και ακολουθούμε αυτήν την επιλογή, ελπίζοντας το σύμβολο που ζητούμε να αναγνωριστεί από τον κανόνα που θα ακολουθήσει.
- Αν η γραμματική δεν δίνει σαν επιλογή το κενό, τότε έχουμε φτάσει σε κατάσταση σφάλματος, εμφανίζεται το κατάλληλο μήνυμα λάθους και τερματίζεται η μετάφραση.

**Οι συναρτήσεις αυτές υλοποιήθηκαν σύμφωνα με τη γραμματική που μας δόθηκε:**

def syntax_analyzer()	Ο συντακτικός αναλυτής μας ξεκινά πάντοτε με αυτή την συνάρτηση. Διαβάζει το 1ο token, και καλεί την startRule() αν ολοκληρωθεί σωστά εμφανίζει μήνυμα επιτυχούς συντακτικής ανάλυσης (εφόσον δεν υπάρχει κάποιο σφάλμα).
startRule	Καλεί την def_main_part και την call_main_part.
def_main_part	Η συνάρτηση αυτή καλεί την def_main_function . Όσο το token είναι το καλείται η def_main_function. Καλείται τόσες φορές όσες είναι και οι κυρίες συναρτήσεις.
def_main_function	Η συνάρτηση αυτή είναι υπεύθυνη να ελέγχει την συντακτική ορθότητα των κύριων συναρτήσεων. Αν γίνει άνοιγμα bracket τότε η συνάρτηση καλή την declarations αφού συμβεί αυτό και το επόμενο token είναι το “def” τότε καλή επαναληπτικά την def_function ενώ όταν σταματήσει το token να είναι το “def” καλή την statements.
def_function	Η συνάρτηση αυτή είναι υπεύθυνη να ελέγχει τη συντακτική ορθότητα των

	συναρτήσεων που δεν είναι κύριες. Η γραμματική της είναι η ίδια με την συνάρτηση <code>def_main_function</code>
<code>declarations</code>	Η συνάρτηση αυτή, όσο το <code>token</code> είναι το « <code>#declare</code> » καλεί επαναληπτικά την <code>declaration_line</code> . Είναι υπεύθυνη να ελέγχει τους ορισμούς των μεταβλητών που μπορεί να χρησιμοποιηθούν.
<code>declaration_line</code>	Η συνάρτηση είναι υπεύθυνη να ελέγχει τους ορισμούς των μεταβλητών καταναλώνει το <code>token</code> και καλεί την <code>id_list</code> .
<code>statement</code>	Η συνάρτηση αυτή ελέγχει αν πρόκειται για απλό <code>statement</code> δηλαδή ένα <code>statement</code> που υλοποιείται σε μία μόνο γραμμή και έτσι καλεί την <code>simple_statement</code> ή ένα σύνθετο <code>statement</code> που υλοποιείται σε περισσότερες γραμμές και συνεπώς καλεί την <code>structured_statement</code> .
<code>statements</code>	Η συνάρτηση καλείται από τις συναρτήσεις που είναι υπεύθυνες να ελέγχουν τη συντακτική ορθότητα των συναρτήσεων και έτσι καλεί την <code>statement</code> εάν το επόμενο <code>token</code> αφορά κάποιο <code>statement</code> τότε ξανακαλεί την <code>statement</code> .
<code>simple_statement</code>	Ελέγχει το είδος του απλού <code>statement</code> μπορεί να είναι είτε η <code>print()</code> και έτσι θα καλέσει την <code>print_stat</code> είτε η <code>return</code> και έτσι θα καλέσει την <code>return_stat</code> είτε <code>id</code> και θα αφορά κάποια ανάθεση έτσι θα καλέσει την <code>assignment_stat</code> .
<code>structured_statement</code>	Ελέγχει το είδος του σύνθετου <code>statement</code> ελέγχοντας το <code>token</code> , αν πρόκειται για μια συνθήκη <code>if</code> καλεί την <code>if_stat</code> ενώ αν πρόκειται για μια συνθήκη <code>while</code> καλεί την <code>while_stat</code> .
<code>assignment_stat</code>	Η συνάρτηση αυτή είναι υπεύθυνη να ελέγχει τη συντακτική ορθότητα μιας ανάθεσης αν δηλαδή αποτελείται από τα κατάλληλα <code>token</code> . Σημαντικό είναι ότι διαχωρίζει και εκτελεί διαφορετικούς ελέγχους αν πρόκειται για μια απλή ανάθεση ή για μια σύνθετη

	ανάθεση που περιέχει κάποια έκφραση αν συμβαίνει το δεύτερο καλεί την expression.
print_stat	Η συνάρτηση αυτή είναι υπεύθυνη να ελέγχει τη συντακτική ορθότητα ενός print statement. Αν δει αριστερή παρένθεση καλεί την expression για να ελεγχθεί και η έκφραση που περιέχεται σε αυτή.
return_stat	Παρόμοια με την print_stat εκτελεί το συντακτικό έλεγχο ενός return statement . και εδώ αν δει αριστερή παρένθεση καλεί την expression για να ελεγχθεί και η έκφραση που περιέχεται σε αυτή.
if_stat	Είναι υπεύθυνη να ελέγχει τη συντακτική ορθότητα μιας σύνθετης δομής if. Αρχικά καταναλώνει το token και εάν δει αριστερή παρένθεση καλεί την condition για να ελεγχθεί η συνθήκη που περιέχεται. Εάν διαπιστωθεί token αριστερού bracket τότε καλείται η statements γιατί τότε θα έχουμε πάνω από μία γραμμή κώδικα που θα εκτελεστεί αν ισχύει η συνθήκη if. Ενώ αν δεν διαπιστωθεί token αριστερού bracket τότε θα εκτελεστεί μία μόνο γραμμή κώδικα και έτσι καλούμε την statement.
while_stat	Όπως και με την παραπάνω συνάρτηση εκτελούμε συντακτικό έλεγχο ενός while statement αρχικά αφού δούμε την αριστερή παρένθεση καλούμε την condition για να ελέγξουμε τη συνθήκη που περιέχεται. Εάν διαπιστωθεί token αριστερού bracket τότε καλείται η statements γιατί τότε θα έχουμε πάνω από μία γραμμή κώδικα που θα εκτελεστεί αν ισχύει η συνθήκη while. Ενώ αν δεν διαπιστωθεί token αριστερού bracket τότε θα εκτελεστεί μία μόνο γραμμή κώδικα και έτσι καλούμε την statement.
id_list	Η συνάρτηση αυτή καλείται από την def_function ώστε να ελέγξει την συντακτική ορθότητα των ορισμάτων της



	συνάρτησης είτε καλείται από την <code>declaration_line</code> ώστε να ελέγξει τη συντακτική ορθότητα κατά την διαδικασία ορισμού μεταβλητών
<code>expression</code>	Η συνάρτηση αυτή αρχικά καλεί την <code>optional_sign</code> ώστε να αποφασίσει για το πρόσημο έπειτα καλεί την <code>term</code> ώστε είτε να βρει τον παράγοντα δηλαδή κάποιον αριθμό είτε κάποια μεταβλητή είτε κάποια έκφραση. Στη συνέχεια όσο το token που διαβάζετε είναι το σύμβολο της της πρόσθεσης είτε τις αφαιρέσεις καλεί την κατάλληλη συνάρτηση που μου επιστρέφει αυτά τα σύμβολα στην έκφρασή.
<code>term</code>	Η συνάρτηση αυτή θα μου βρει τον παράγοντα μέσω της συνάρτησης <code>factor</code> και όσο το token που διαβάζω είναι το σύμβολο της διαίρεσης είτε το σύμβολο του πολλαπλασιασμού καλώ τις συναρτήσεις <code>MUL_OP</code> και <code>factor</code> .
<code>factor</code>	Η συνάρτηση αυτή είτε μου επιστρέφει τον ακέραιο αριθμό είτε ελέγχει εάν έχουν ανοίξει και έχουν κλείσει σωστά οι παρενθέσεις σε μια έκφραση είτε εάν το token που διαβάσει είναι <code>id</code> καλεί την <code>idtail</code> .
<code>idtail</code>	Η συνάρτηση αυτή ελέγχει ανοίγει αριστερή παρένθεση, αν συμβαίνει αυτό καταναλώνει το token και καλεί την <code>actual_par_list</code> . Αλλιώς δεν κάνει τίποτα.
<code>actual_par_list</code>	Η συνάρτηση αυτή δημιουργεί μια λίστα με όλους τους παραμέτρους ή τους παράγοντες.
<code>optional_sign</code>	Η συνάντηση αυτή αποφασίζει για το πρόσημο.
<code>condition</code>	Η συνάρτηση αυτή ελέγχει τις συνθήκες που επιστρέφουν αληθές ή ψευδές .
<code>bool_term</code>	Η συνάρτηση αυτή αφορά τις εκφράσεις που περιέχουν λογικούς τελεστές και κυρίως το λογικό τελεστή “και”.
<code>bool_factor</code>	Η συνάρτηση αυτή ελέγχει αν μια συνθήκη που αποτιμά τιμές τρώει ή false.



	Αρχικά ελέγχει αν στην αρχή της έκφρασης υπάρχει το not. Αν υπάρχει ψάχνει να βρει αριστερό bracket και δεξιά bracket αλλιώς τυπώνει μήνυμα λάθους. Εάν δεν υπάρχει not στην αρχή της έκφρασης ελέγχει αν υπάρχει αριστερό bracket και αν όντως υπάρχει ελέγχει μέσω της condition την έκφραση που περιέχεται μέσα σε αυτό. Εάν δεν έχει τίποτα από τα παραπάνω τότε καλεί την expression.
call_main_part	Η συνάρτηση αυτή είναι υπεύθυνη να ελέγχει τη σωστή συντακτικά κλήση της main συναρτήσεων ενός προγράμματος.
main_function_call	Η συνάρτηση αυτή είναι υπεύθυνη να ελέγχει την σωστή κλήση των κύριων συναρτήσεων.
ADD_OP	Διαβάζει τα σύμβολα “+” και “-” αν τα διαβάσει καταναλώνει το token.
MUL_OP	Διαβάζει τα σύμβολα “*” και “/” αν τα διαβάσει καταναλώνει το token.
REL_OP	Διαβάζει τους χαρακτήρες σύγκρισης αν τα διαβάσει καταναλώνει το token.

Οι παραπάνω συναρτήσεις αν διαπιστώσουν λάθος στην συντακτική ορθότητα του προγράμματος τυπώνουν κατάλληλο μήνυμα που περιέχει μια σύντομη περιγραφή του λάθους και τον αριθμό γραμμής που υπάρχει το παραπάνω λάθος.

Επίσης θα σταματήσει η εκτέλεση του συντακτικού ελέγχου στο πρώτο λάθος.

Εάν δεν διαπιστωθεί κάποιο συντακτικό λάθος τότε ο συντακτικός έλεγχος θα έχει τελειώσει και θα τυπωθεί μήνυμα επιτυχούς ελέγχου στην οθόνη του χρήστη.

# Αναφορά ελέγχου συντακτικού αναλυτή

```
def main_factorial():
#{
    #$ declarations #$
    #declare x
    #declare i,fact

    #$ body of main_factorial #$
    x = int(input());
    fact = 1;
    i = 1;
    while (i<=x):
    #{
        fact = fact * i;
        i = i + 1;
    #}
    print(fact);
#}
```

Επιβεβαιώνουμε την σωστή λειτουργία του συντακτικού αναλυτή μας , δίνοντας ως είσοδο το πρόγραμμα factorial.cpy που μας δόθηκε μαζί με την εκφώνηση της παρούσας εργαστηριακής άσκησης.

Το πρόγραμμα στη διπλανή φωτογραφία.

Στην αρχή του συντακτικού ελέγχου θα καλεστεί η συνάρτηση startRule(). Αυτή με τη σειρά της θα καλέσει την συνάρτηση

def\_main\_part όπου θα καλέσει την def\_main\_function πρώτα θα αναγνωριστεί η λέξη κλειδί “def” ύστερα θα αναγνωριστεί το όνομα της κυρίας συνάρτησης, ή αριστερή και η δεξιά παρένθεση η άνω κάτω. Και το άνοιγμα των bracket. Έπειτα θα καλεστεί η συνάρτηση declarations αφού θέλουμε να αναγνωρίσουμε τις δηλώσεις των τοπικών μεταβλητών της συνάρτησης αυτής που βρισκόμαστε . Αφού γίνουν οι δηλώσεις ακολουθούν οι αρχικοποιήσεις - ορισμοί των μεταβλητών αυτών με την συνάρτηση assignment\_stat ύστερα θα βρεθεί η δομή του while και θα καλεστεί η συνάρτηση while\_stat και μέσα από αυτή θα καλεστεί η συνάρτηση condition για να γίνει συντακτικός έλεγχος της συνθήκης που περιέχεται στην επαναλαμβανόμενη δομή τέλος θα καλεστεί η print\_stat όπου θα είναι και η τελευταία συνάρτηση που θα καλέσει ο συντακτικός αναλυτής μας για το συγκεκριμένο πρόγραμμα.

Φαίνεται ότι το πρόγραμμα αυτό είναι τόσο λεκτικά όσο και συντακτικά ορθό αφού τυπώνεται στην γραμμή εντολών το κατάλληλο μήνυμα.

Εάν τώρα δεν ήταν συντακτικά ορθό έλειπε παραδείγματος χάρη μια παρένθεση είτε αριστερή είτε δεξιά είτε κάποιο bracket δεν άνοιγε ή δεν έκλεινε τότε θα τυπωνόταν το κατάλληλο μήνυμα λάθους εμπεριέχοντας τη γραμμή που συνέβη αυτό.

Στην ουσία ο συντακτικός έλεγχος αποτελείται από τις συναρτήσεις που αναφέρθηκαν παραπάνω και καλούνται αναμεταξύ τους μέχρι να τελειώσει το πρόγραμμα που θέλουμε να μεταγλωττίσουμε.

# Εισαγωγή στον ενδιάμεσο κώδικα.

Ένα πρόγραμμα συμβολισμένο στην ενδιάμεση γλώσσα αποτελείται από μία σειρά από τετράδες, οι οποίες είναι αριθμημένες έτσι ώστε σε κάθε τετράδα να μπορούμε να αναφερθούμε χρησιμοποιώντας τον αριθμό της ως ετικέτα. Κάθε τετράδα αποτελείται από έναν τελεστή και τρία τελούμενα. Αν μετρήσουμε και την ετικέτα, πρόκειται για μία τετράδα που αποτελείται από ... πέντε πράγματα. Από τις ονομασίες *τελεστής* και *τελούμενα* μπορεί κανείς να συμπεράνει ότι ο **τελεστής καθορίζει την ενέργεια** που πρόκειται να γίνει και τα **τελούμενα είναι εκείνα πάνω στα οποία θα εφαρμοστεί η ενέργεια**.

Για την δημιουργία αυτών των τετράδων ενδιάμεσου κώδικα χρειαζόμαστε κάποιες βοηθητικές συναρτήσεις αυτές τις συναρτήσεις περιγράφουμε στον παρακάτω πίνακα.

## Βοηθητικές συναρτήσεις:

<code>def nextQuad():</code>	Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.
<code>def genQuad(op, x, y, z):</code>	Δημιουργεί την επόμενη τετράδα (op, x, y, z).
<code>def newTemp():</code>	Δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή, οι προσωρινές μεταβλητές είναι της μορφής <code>%_1</code> , <code>%_2</code> , <code>%_3</code> .
<code>def emptyList():</code>	Δημιουργεί μία κενή λίστα ετικετών τετράδων.
<code>def makeList(x):</code>	Δημιουργεί μία λίστα ετικετών τετράδων που περιέχει μόνο το x.
<code>def merge(list1, list2):</code>	Δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών list1, list2.
<code>def backPatch(list, z):</code>	Η list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

# Αλλαγές στις συν/σεις του συντακτικού που αφορούν αριθμητικές παραστάσεις.

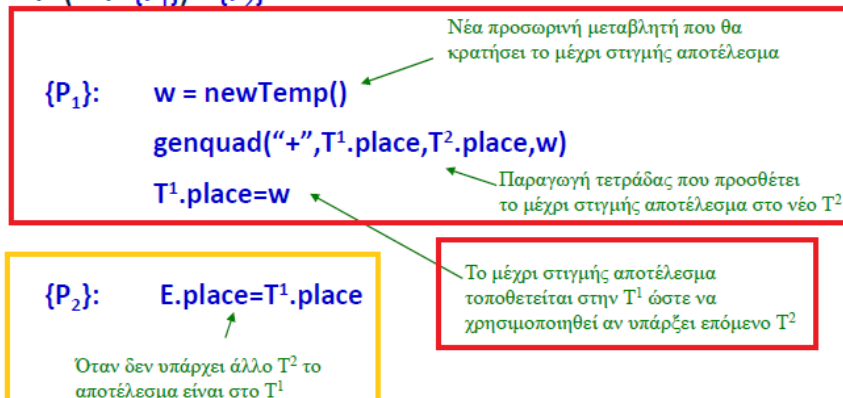
Στις συναρτήσεις που αφορούν αριθμητικές παραστάσεις προσθέσαμε το σχέδιο του ενδιαμέσου κώδικα σύμφωνα με όσα διδαχθήκαμε από τις σημειώσεις.

Έτσι δημιουργούνται οι κατάλληλες προσωρινές μεταβλητές και γενικότερα γίνεται η κατάλληλη σχεδίαση του ενδιαμέσου κώδικα δημιουργώντας κατάλληλες τετράδες με τέτοιον τρόπο ώστε να έχουμε το αποτέλεσμα που χρειαζόμαστε για κάθε πράξη είτε αυτή είναι πρόσθεση που γίνεται με τη συνάρτηση `expression()` είτε πολλαπλασιασμός που γίνεται με τη συνάρτηση `term()`.

Στις παρακάτω εικόνες μπορεί να δει κανείς το σχέδιο ενδιαμέσου κώδικα που γράφτηκε μέσα στις συναρτήσεις που αφορούν τις αριθμητικές παραστάσεις.

## `expression()`

$E \rightarrow T^1 ( + T^2 \{P_1\} ) * \{P_2\}$



```
#Aritmitikes parastaseis.
#E -> T1 ( + T2 {P1} ) * {P2} sel 14
def expression():
    global results_lex
    global line

    optional_sign()

    T1place = term() #T1

    while(results_lex[2]==plus or results_lex[2]==minus):
        addOperators = ADD_OP()
        T2place = term() #T2

        #{P1}:
        w = newTemp()
        genQuad(addOperators, T1place, T2place, w)
        T1place = w

    #{P2}:
    Eplace = T1place
    return Eplace
```

## term()

$T \rightarrow F^1 (\times F^2 \{P_1\})^* \{P_2\}$

$\{P_1\}$ :     `w = newTemp()`  
              `genquad("×", F1.place, F2.place, w)`  
              `F1.place = w`

$\{P_2\}$ :     `T.place = F1.place`

```
#Aritmitikes parastaseis pollaplasiasmos.  
#T -> F1 (× F2 {P1})* {P2} sel 15  
def term():  
    global results_lex  
    global line  
  
    F1place = factor() #F1  
  
    while(results_lex[2]==multiplication or results_lex[2]==division):  
        mulOperators= MUL_OP()  
  
        F2place = factor() #F2  
         $\{P_1\}$ :  
        w=newTemp()  
        genQuad(mulOperators, F1place, F2place, w)  
        F1place = w  
  
         $\{P_2\}$ :  
        Tplace =F1place  
    return Tplace
```

## factor()

$F \rightarrow ( E ) \{P_1\}$

Απλή μεταφορά από το E.place στο F.place

$\{P_1\}$ : `F.place=E.place`

$F \rightarrow \text{id } \{P_1\}$

Απλή μεταφορά από το id.place στο F.place

$\{P_1\}$ : `F.place=id.place`

```

#F -> ( E ) {P1} || F -> id {P1} sel16
def factor():
    global results_lex
    global line
    linetemp = line

    if(results_lex[2]==number):
        Fplace = results_lex[1]
        linetemp = line
        results_lex = lex()

    elif(results_lex[2]==left_parenthesis):
        linetemp = line
        results_lex = lex()

        #{P1}: endiamesos sel16
        Eplace = expression()
        Fplace = Eplace

    if(results_lex[2]==right_parenthesis):
        linetemp = line
        results_lex = lex()
    else:
        print("SYNTAX error: Missing ')' on end of the factor line:",linetemp)
        exit(-1)

    elif(results_lex[2]==id):
        factor_t = results_lex[1]
        linetemp = line
        results_lex = lex()

        ##{P1}: endiamesos sel16
        Fplace = idtail(factor_t)

    else:
        print("SYNTAX error: Missing constant OR expression OR variable on factor line:",linetemp)
        exit(-1)

    return Fplace

```

# Αλλαγές στις συν/σεις του συντακτικού που αφορούν λογικές παραστάσεις.

Η γλώσσα **cutePy** δέχεται όχι μόνο αριθμητικές παραστάσεις αλλά και λογικές παραστάσεις εκτελεί πράξεις με αυτές αλλά σημαντικότερα ελέγχει τις συνθήκες μέσα σε δομές απλές είτε επαναλαμβανόμενες.

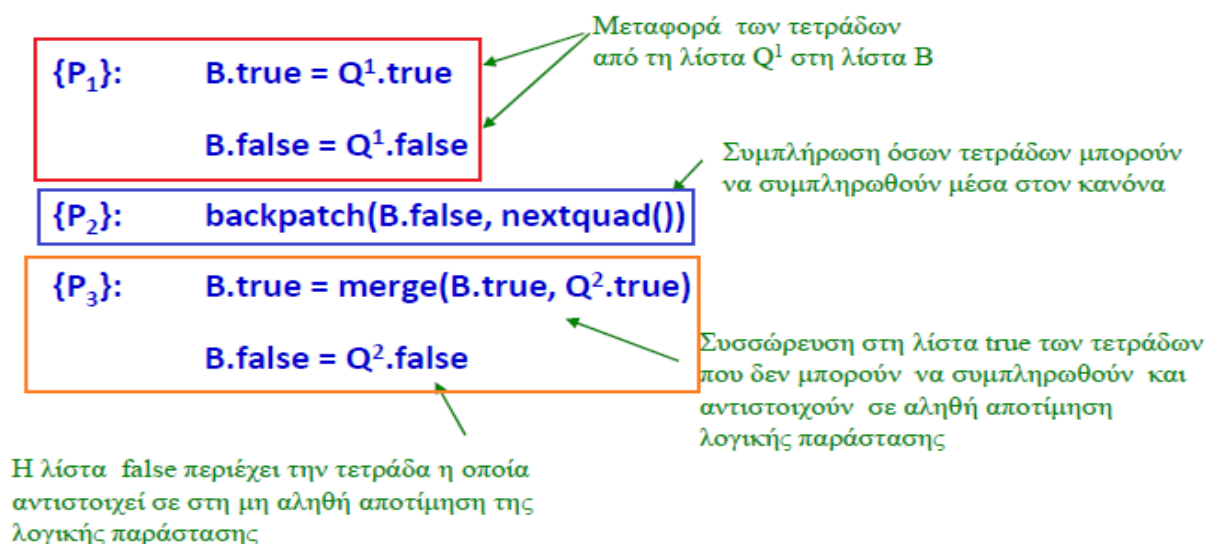
Γενικότερα οι συναρτήσεις που αφορούν τις λογικές παραστάσεις μας παράγουν τετράδες που αποτιμούν το αληθές και τετράδες που αποτιμούν το ψευδές αποτέλεσμα της λογικής παράστασης. Εάν το αποτέλεσμα της συνθήκης μπορεί να αποτιμηθεί άμεσα τότε γίνονται τα κατάλληλα άλματα ώστε να εκτελεσθεί το μέρος του κώδικα σύμφωνα με το αποτέλεσμα που μας έδωσε η παράσταση εάν τώρα το αποτέλεσμα της παράστασης δεν μπορεί να αποτιμηθεί άμεσα πρέπει να ελεγχθεί και η επόμενη συνθήκη έτσι προχωράμε στον έλεγχο αυτής.

Στις παρακάτω εικόνες φαίνονται οι αλλαγές που έγιναν στις συναρτήσεις του συντακτικού ελέγχου σύμφωνα με τα όσα διδαχθήκαμε στις διαλέξεις.

**condition()**

## Λογικές Παραστάσεις - OR

$B \rightarrow Q^1 \{P_1\} ( \text{or} \{P_2\} Q^2 \{P_3\} )^*$



```

#B -> Q1 {P1} ( or {P2} Q2 {P3}) * sel 26. || R -> ( B ) {P1}
def condition():
    global results_lex
    global line

    #endiamesos
    Btrue = []
    Bfalse = []

    Q1 = bool_term()
    #{P1}:

    #B.true = Q1.true      || R.true=B.true
    #B.false = Q1.false    || R.false=B.false

    Btrue = Q1[0]
    Bfalse = Q1[1]

    while(results_lex[2]==or_key):
        results_lex=lex()

    #{P2}: backpatch(B.false, nextquad())
           backPatch(Bfalse, nextQuad())

    Q2 = bool_term()
    #{P3}: B.true = merge(B.true, Q2.true)
           #B.false = Q2.false

    Btrue = merge(Btrue, Q2[0])
    Bfalse = Q2[1]

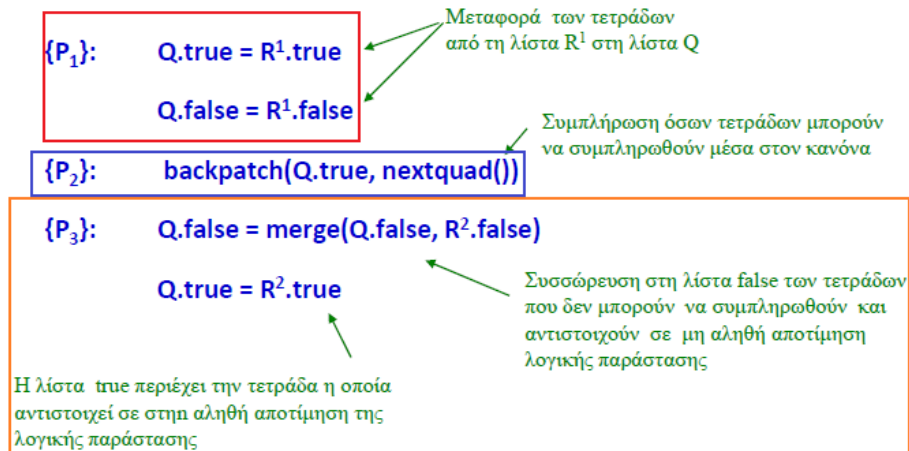
    #endiamesos
    return Btrue, Bfalse

```



## Λογικές Παραστάσεις - AND

$Q \rightarrow R^1 \{P_1\} ( \text{and} \{P_2\} R^2 \{P_3\} )^*$



```
#Q -> R1 {P1} ( and {P2} R2 {P3} ) * sel 27.
def bool_term():
    global results_lex
    global line

    #endiamesos
    Qtrue = []
    Qfalse = []
    R1 = bool_factor()

    #{P1}:
    #Q.true = R1.true
    #Q.false = R1.false

    Qtrue = R1[0]
    Qfalse = R1[1]

    while(results_lex[2]==and_key):
        results_lex=lex()
        linetemp = line

        #{P2}: backpatch(Q.true, nextquad())
               backPatch(Qtrue, nextQuad())

        R2 = bool_factor()

        #{P3}: Q.false = merge(Q.false, R2.false)
               #Q.true = R2.true

        Qfalse = merge(Qfalse, R2[1])
        Qtrue = R2[0]

    #endiamesos
    return Qtrue, Qfalse
```

## Bool\_factor()

R -> not ( B ) {P<sub>1</sub>}

{P<sub>1</sub>}:     R.true=B.false     ← Αντιστροφή και μεταφορά  
             R.false=B.true     ← τετράδων από τη λίστα B στη λίστα R

```
# R -> not ( B ) {P1} ---NOT--- sel 31 || R -> E1 relop E2 {P1} sel 33
def bool_factor():
    global results_lex
    global line
    linetemp = line

    #endiamesos
    Rtrue = []
    Rfalse = []

    if(results_lex[2]==not_key):
        linetemp = line
        results_lex=lex()

    if(results_lex[2]==left_square_bracket):
        linetemp = line
        results_lex = lex()

    #endiamesos
    cond = condition()

    if(results_lex[2]==right_square_bracket):
        linetemp = line
        results_lex = lex()

    #{P1}: ---GIA NOT---
        #R.true=B.false
        #R.false=B.true

        Rtrue = cond[1]
        Rfalse = cond[0]
```

$R \rightarrow E^1 \text{ relop } E^2 \{P_1\}$

```
{P1}:    R.true=makelist(nextquad())
          genQuad(relop, E1.place, E2.place, "_")
          R.false=makelist(nextquad())
          genQuad("jump", "_", "_", "_")
```

Δημιουργία μη συμπληρωμένης  
τετράδας και εισαγωγή στη λίστα  
μη συμπληρωμένων τετράδων για  
την αληθή αποτίμηση της relop

Δημιουργία μη συμπληρωμένης  
τετράδας και εισαγωγή στη λίστα  
μη συμπληρωμένων τετράδων για  
τη μη αληθή αποτίμηση της relop

```
else:

    #endiamesos
    E1place = expression()
    relop = REL_OP()
    E2place = expression()

#{P1}:
    #R.true=makelist(nextquad())
    #genQuad(relop, E1.place, E2.place, "_")
    #R.false=makelist(nextquad())
    #genQuad("jump", "_", "_", "_")

    Rtrue=makeList(nextQuad())
    genQuad(relop, E1place, E2place, '_')
    Rfalse=makeList(nextQuad())
    genQuad('jump', '_', '_', '_')

return Rtrue, Rfalse
```

Επιπλέον έχουν γίνει αλλαγές στις συναρτήσεις print() , assignment() ,return() όπου ακολουθούν και αυτές το σχέδιο ενδιάμεσου κώδικα που διδαχθήκαμε στις διαλέξεις θεωρήσαμε σωστό χάριν απλότητας να μην της δείξουμε με κάποια εικόνα.

# Αλλαγές στις συν/σεις του συντακτικού που αφορούν τις δομές if , while.

Οι δομές αυτές αποτελούν το πιο δύσκολο εγχείρημα του ενδιαμέσου κώδικα αφού καλούν τις συναρτήσεις που αφορούν τόσο τις λογικές παραστάσεις έτσι ώστε να ελεγχθούν οι συνθήκες που περιέχονται σε αυτές όσο και οι αριθμητικές παραστάσεις ώστε να εκτελεστούν οι εντολές που περιέχονται είτε αν προβεί η δομή αυτή σε αληθές είτε σε ψευδές. αποτέλεσμα

Έτσι παραδείγματος χάρη η **δομή If πρέπει να καλέσει την συνάρτηση condition** ώστε να αποτιμηθεί το αποτέλεσμα της συνθήκης αν αυτό είναι αληθές τότε θα εκτελεστεί ο κώδικας που εμπεριέχεται στο if αλλιώς θα ελέγξουμε αν υπάρχει else ώστε να εκτελεστούν αυτές οι εντολές που περιέχονται στο else. Επιπλέον πρέπει να γίνουν τα κατάλληλα άλματα και τα κατάλληλα backpatch και να συμπληρωθούν οι τετράδες που είχαν μείνει ασυμπλήρωτες.

Και πάλι το σχέδιο ενδιαμέσου κώδικα είναι βασισμένος σε όσα διδαχθήκαμε και στις σημειώσεις όπως φαίνεται και στις παρακάτω εικόνες.

## If stat()

### Δομή if

$S \rightarrow \text{if } B \text{ then } \{P1\} S^1 \{P2\} \text{TAIL } \{P3\}$

**{P1}: backpatch(B.true,nextquad())**

**{P2}: ifList=makelist(nextquad())  
genquad("jump","\_","\_","\_")**

**backpatch(B.false,nextquad())**

**{P3}: backpatch(ifList,nextquad())**

$\text{TAIL} \rightarrow \text{else } S^2 \mid \text{TAIL} \rightarrow \epsilon$

Συμπλήρωση των τετράδων που έχουν μείνει ασυμπλήρωτες και γνωρίζουμε τώρα ότι πρέπει να συμπληρωθούν με την επόμενη τετράδα, στο if και else αντίστοιχα

Εξασφαλίζουμε ότι εάν εκτελεστούν οι εντολές του if δε θα εκτελεστούν στη συνέχεια οι εντολές του else

```

#Den kanw if==if_key afou to exw dei stin structured_statement.
#S -> if B then {P1} S1 {P2} TAIL {P3} , TAIL -> else S2 | TAIL -> ε sel S1

def if_stat():
    global results_lex
    global line
    linetemp = line
    results_lex= lex()

    if(results_lex[2] == left_parenthesis):
        linetemp = line
        results_lex = lex()

        #{P1}:backpatch(B.true,nextquad())
        cond = condition()
        backPatch(cond[0], nextQuad())

```

```

if(results_lex[2] == colon):
    linetemp = line
    results_lex = lex()

if(results_lex[2] == left_curly_bracket):
    linetemp = line
    results_lex = lex()
    statements()

#{P2}:
    #ifList=makelist(nextquad())
    #genquad("jump", "_", "_", "_")
    #backpatch(B.false,nextquad())

    ifList = makeList(nextQuad())
    genQuad('jump', '_', '_', '_')
    backPatch(cond[1], nextQuad())

    if(results_lex[2] == right_curly_bracket):
        linetemp = line
        results_lex = lex()

    else:
        print("SYNTAX error: Missing '#' on if statements line:",linetemp )
        exit(-1)

else:
    statement()

#{P2}: (An exw mono mia entoli pou akolouthei)
    #ifList=makelist(nextquad())
    #genquad("jump", "_", "_", "_")
    #backpatch(B.false,nextquad())

    ifList = makeList(nextQuad())
    genQuad('jump', '_', '_', '_')
    backPatch(cond[1], nextQuad())

```

```

if(results_lex[2] == else_key):
    linetemp = line
    results_lex = lex()

if(results_lex[2] == colon):
    linetemp = line
    results_lex = lex()

if(results_lex[2] == left_curly_bracket):
    linetemp = line
    results_lex = lex()
    statements()

    #{P3}: backpatch(ifList,nextquad())
    backPatch(ifList, nextQuad())

    if(results_lex[2] == right_curly_bracket):
        linetemp = line
        results_lex = lex()

    else:
        print("SYNTAX error: Missing '#' on if statements line:",linetemp)
        exit(-1)

    else:
        statement()

    #{P3}: backpatch(ifList,nextquad()) (a exw mono mia entoli )
    backPatch(ifList, nextQuad())

else:
    print("SYNTAX error: Missing ':' on else statement line:",linetemp)
    exit(-1)

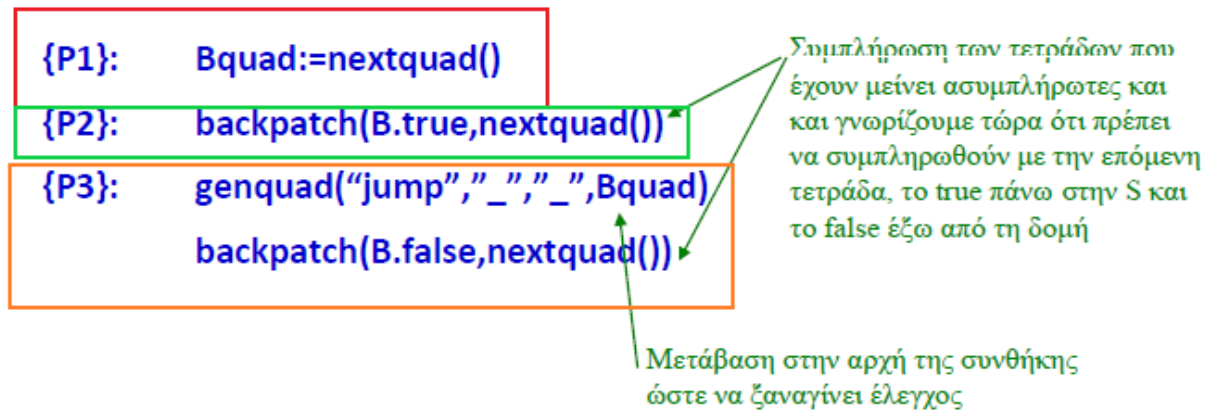
else:
    #{P3}: backpatch(ifList,nextquad())
    backPatch(ifList, nextQuad())

```

## While\_stat()

### Δομή while

$S \rightarrow \text{while } \{P1\} B \text{ do } \{P2\} S^1 \{P3\}$



```
#Den kanw if==while_key afou to exw dei stin structured_statement.  
#S -> while {P1} B do {P2} S1 {P3} sel 42
```

```
def while_stat():  
    global results_lex  
    global line  
    linetemp = line  
    results_lex= lex()  
  
    if(results_lex[2] == left_parenthesis):  
        linetemp = line  
        results_lex = lex()
```

```
#{P1}:Bquad:=nextquad()  
Bquad=nextQuad()  
  
cond = condition()
```

```
#{P2}:backpatch(B.true,nextquad())  
backPatch(cond[0], nextQuad())
```

```

if(results_lex[2] == colon):
    linetemp = line
    results_lex = lex()

if(results_lex[2] == left_curly_bracket):
    linetemp = line
    results_lex = lex()
    statements()

#{P3}:
    #genquad("jump", "_", "_", Bquad)
    #backpatch(B.false, nextquad())

    genQuad('jump', '_', '_', Bquad)
    backPatch(cond[1], nextQuad())

    if(results_lex[2] == right_curly_bracket):
        linetemp = line
        results_lex = lex()

    else:
        print("SYNTAX error: Missing '#' on while statements line:", linetemp)
        exit(-1)
else:
    statement()

#{P3}: (Ean den exeí '{' diladi akolouthei mono mia entoli)
    #genquad("jump", "_", "_", Bquad)
    #backpatch(B.false, nextquad())

    genQuad('jump', '_', '_', Bquad)
    backPatch(cond[1], nextQuad())

```



# Αναφορά ελέγχου ενδιάμεσου κώδικα.

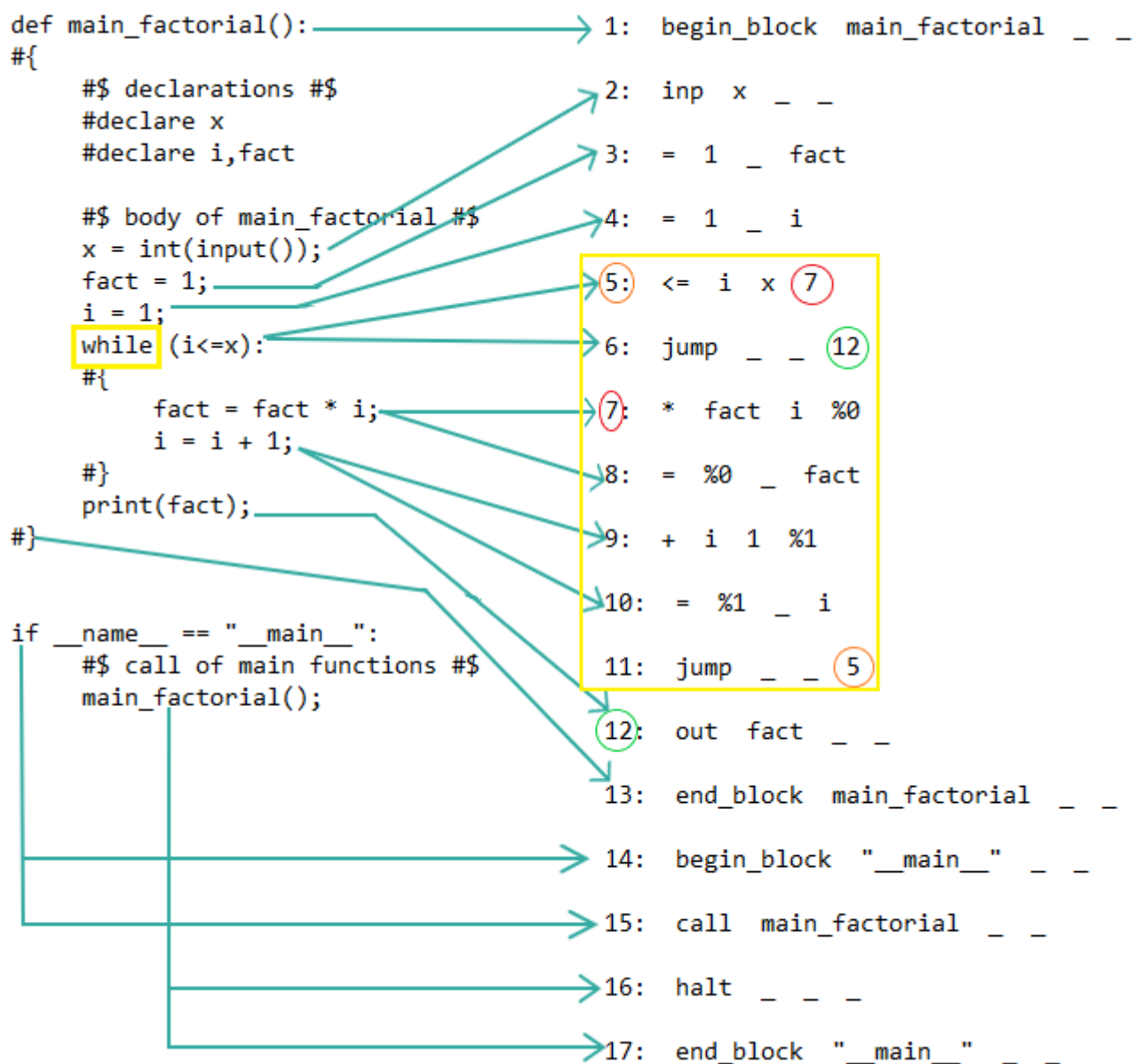
---

```
def main_factorial():
#{
    #$ declarations #$
    #declare x
    #declare i,fact

    #$ body of main_factorial #$
    x = int(input());
    fact = 1;
    i = 1;
    while (i<=x):
    #{
        fact = fact * i;
        i = i + 1;
    #}
    print(fact);
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_factorial();
```

Επιβεβαιώνουμε την σωστή λειτουργία του ενδιάμεσου κώδικα , δίνοντας ως είσοδο το πρόγραμμα factorial.cpy που μας δόθηκε μαζί με την εκφώνηση της παρούσας εργαστηριακής άσκησης όπως φαίνεται και στη διπλανή εικόνα. Τα αποτελέσματα δηλαδή οι τετράδες που μου δημιουργεί ενδιάμεσος κώδικας φαίνονται στην **εικόνα α**.



**Εικόνα α.**

Όπως βλέπουμε και στην παραπάνω εικόνα ο ενδιαμέσος κώδικας μας δημιουργεί τις σωστές τετράδες. Αρχικά ξεκίνα καλώντας την συνάρτηση `main_function` και δημιουργεί την τετράδα στη γραμμή ένα, στη συνέχεια προσπερνά τα σχόλια και τις δηλώσεις των μεταβλητών μέχρι να φτάσει στην αρχικοποίηση αυτών όπου καλεί τις κατάλληλες συναρτήσεις όπως είναι οι συνάρτηση `assignment_stat()`.

Ύστερα καλείται η συνάρτηση `while_stat()` αφού διαβάζετε η επαναλαμβανόμενη δομή `while` με τη σειρά της αυτή καλεί τις κατάλληλες συναρτήσεις ώστε να δημιουργηθούν οι τετράδες που βρίσκονται εντός του κίτρινου ορθογωνίου. Όπως φαίνεται και στα αποτελέσματα αρχικά το `while` δημιουργεί 2 τετράδες μία για κάθε αποτέλεσμα που μπορεί να φέρει η συνθήκη που εμπεριέχεται σε αυτό έτσι όπως φαίνεται στη γραμμή 5, γίνεται έλεγχος αν το `x` είναι μεγαλύτερο ή ίσο του `i` και αν είναι, θα γίνει άλμα στη γραμμή 7 ώστε να εκτελεστούν οι εντολές που εμπεριέχονται

στην επαναλαμβανόμενη δομή αφού το αποτέλεσμα της συνθήκης είναι αληθές. Η δεύτερη τετράδα που δημιουργείται έχει σκοπό να κάνει άλμα έξω από τη επαναλαμβανόμενη δομή while εφόσον το αποτέλεσμα της συνθήκης που εμπεριέχεται σε αυτό είναι ψευδές και έτσι θα κάνει άλμα στη γραμμή 12. Εάν τώρα το αποτέλεσμα είναι αληθές γίνονται οι εντολές που η περιέχονται στο while και τέλος γίνεται ένα άλμα πίσω στη γραμμή 5 του ενδιάμεσου κώδικα ώστε να ξανά ελεγχθεί εάν το  $x$  είναι μεγαλύτερο ή ίσο του  $i$ , αυτό είναι προφανές ότι γίνεται μόνο στη δομή while αφού είναι επαναλαμβανόμενη. Αφού τελειώσουμε το while τα πράγματα είναι πιο απλά έτσι έχουμε μία τετράδα που δημιουργείται από την `print_stat()` στη γραμμή 12. Επιπλέον βλέπουμε ότι δημιουργείται μια τετράδα που εμπεριέχει το `end_block` της συνάρτησης `main_factorial`. Τέλος δημιουργούνται οι κατάλληλες τετράδες στο κάλεσμα της κυρίας συνάρτησης.

# Εισαγωγή στον πίνακα συμβόλων.

Ο πίνακας συμβόλων είναι δυναμική δομή στην οποία αποθηκεύεται πληροφορία σχετιζόμενη με τα συμβολικά ονόματα που χρησιμοποιούνται στο υπό μεταγλώττιση πρόγραμμα. Η δομή αυτή παρακολουθεί τη μεταγλώττιση και μεταβάλλεται δυναμικά, με την προσθήκη ή αφαίρεση πληροφορίας σε και από αυτήν, ώστε σε κάθε σημείο της διαδικασίας της μεταγλώττισης να περιέχει ακριβώς την πληροφορία που εκείνη τη στιγμή. Η πληροφορία αυτή είναι χρήσιμη για έλεγχο σφαλμάτων, αλλά είναι διαθέσιμη να ανακτηθεί κατά τη φάση της παραγωγής του τελικού κώδικα.

Ο πίνακας συμβόλων διατηρεί διαφορετική πληροφορία για κάθε είδος συμβολικού ονόματος. Σκοπός του είναι να συγκεντρώσει όλη την πληροφορία που θα απαιτηθεί να αντληθεί για το κάθε συμβολικό όνομα μέχρι το τέλος της μεταγλώττισης.

Ας ξεκινήσουμε με τον πιο συχνά χρησιμοποιούμενο τύπο, αυτόν της **μεταβλητής**. Κάθε **μεταβλητή** έχει:

- Όνομα
- Τύπο
- Offset

**Offset** πληροφορία πολύ σημαντική για κάθε μεταβλητή είναι η θέση στην οποία αυτή θα βρίσκεται στην μνήμη. Η πληροφορία αυτή είναι απαραίτητη και θα αναζητηθεί στον πίνακα συμβόλων κατά την παραγωγή τελικού κώδικα.

Ένας άλλος τύπος εγγραφών σε πίνακα συμβόλων είναι η **παράμετρος**. Πρόκειται για τις παραμέτρους που περνούμε στις συναρτήσεις. Η παράμετρος μοιάζει πολύ με τη μεταβλητή. Ο τρόπος με τον οποίο τελικά θα διαχειριστούμε τις μεταβλητές και τις παραμέτρους είναι πολύ διαφορετικός, αλλά όσον αφορά τον πίνακα συμβόλων η πληροφορία που διατηρούμε είναι παρόμοια.

Κάθε **παράμετρος** έχει:

- Όνομα
- Τύπο
- Offset

Επιπλέον δεν υπάρχει κάποιο mode αφού έχουμε **ΜΟΝΟ πέρασμα με τιμή**.

Το τελευταίο πεδίο που χρειαζόμαστε είναι το offset. Όπως και στις μεταβλητές, πρέπει να γνωρίζουμε την απόσταση της παραμέτρου από την αρχή του εγγραφήματος δραστηριοποίησης, ώστε να μπορούμε να εντοπίζουμε την παράμετρο στη μνήμη.

Ένας άλλος τύπος εγγραφών σε πίνακα συμβόλων είναι η **συνάρτηση**.

Χρησιμοποιούνται για να σημειώσουν την ύπαρξη ενός υποπρογράμματος στον κώδικα.

Κάθε **συνάρτηση** έχει:

- Όνομα
- Τύπο
- Offset
- startQuad
- ListArgument
- frameLength

**ListArgument** : Λίστα με τις τυπικές παραμέτρους του υποπρογράμματος. Η σειρά με την οποία συναντούμε τις τυπικές παραμέτρους στο υποπρόγραμμα είναι και η σειρά με την οποία εμφανίζονται αυτές στη λίστα.

**startQuad** : Στο πεδίο αυτό αποθηκεύουμε την ετικέτα της πρώτης εκτελέσιμης τετράδας του ενδιαμέσου κώδικα που αντιστοιχεί στη συνάρτηση ή τη διαδικασία αυτή. Με άλλα λόγια, εκεί σημειώνουμε την τετράδα στην οποία πρέπει η καλούσα συνάρτηση να κάνει άλμα προκειμένου να εκκινήσει η εκτέλεση της κληθείσας.

**frameLength** : Μήκος (σε bytes) του εγγραφήματος δραστηριοποίησης της συνάρτησης.

Έτσι δημιουργούμε κάποιες κλάσεις σύμφωνα με τα παραπάνω.

Αρχικά έχουμε την **κλάση scope**

```
#klasi gia ta Scopes
class Scope():
    def __init__(self):
        self.ListEntity = []      #H lista me ta entities mou.
        self.nestingLevel = 0    #Bathos fwliasmatos.
        self.name = ""           #To onoma tou scope.
```

Έχουμε την **κλάση Argument**

```
#klasi gia ta Arguments
class Argument():
    def __init__(self):
        self.name = ""
```

Τέλος έχουμε την κλάση Entity

```
#klasi gia ta Entities mou
class Entity():

    def __init__(self):
        self.type = ""          #0 tupos (function,variable,temporary,parameter).
        self.name = ""         #To onoma tou entity
        self.variable = self.Variable()
        self.parameter = self.Parameter()
        self.tempVar = self.TemporaryVariable()
        self.function = self.Function()

#klasi gia tis MH proswrines metablites
class Variable:
    def __init__(self):
        self.offset = 0

#klasi gia tis parametrous
class Parameter:
    def __init__(self):
        self.offset = 0

#Klasi gia tis proswrines metablites (T1,T2..)
class TemporaryVariable:
    def __init__(self):
        self.offset = 0

#klasi gia tis sunartiseis mas.
class Function:
    def __init__(self):
        self.startQuad = 0      #Etiketa tis prwtis tetrades tis synartiseis.
        self.ListArgument = []  #H lista parametrwn.
        self.frameLength = 0     #Mikos eggrafimatos drastiriopoiisis.
```

Όπου είναι υπερκλάση των κλάσεων Variable, Parameter, TemporaryVariable, Function. Έτσι αυτές οι κλάσεις κληρονομούν τα πεδία type, name.

# Βοηθητικές συναρτήσεις.

def add_scope(nameOfScope):	Προσθήκη ενός νέου scope.
def delete_scope():	Διαγραφή ενός scope.
def add_entity(new_ent):	Προσθήκη ενός νέου entity.
def add_argument(new_arg):	Προσθήκη ενός νέου argument .
def add_parameters():	Για κάθε παράμετρο δημιουργούμε ένα νέο entity στο παραπάνω επίπεδο.
def find_offset():	Βρίσκω το offset.
def find_framelength():	Βρίσκω το framelength της συνάρτησης
def find_startQuad():	Βρίσκω το startQuad.
def write_Symbol_Table(file2):	Τυπώνω τον πίνακα συμβόλων.

**add\_scope(nameOfScope):** όταν ξεκινάμε τη μετάφραση μιας νέας συνάρτησης.

**delete\_scope():** όταν τελειώνουμε τη μετάφραση μιας συνάρτησης - με τη διαγραφή διαγράφουμε την εγγραφή (record) του Scope και όλες τις λίστες με τα Entity και τα Argument που εξαρτώνται από αυτήν.

**add\_entity(new\_ent)**

- όταν συναντάμε δήλωση μεταβλητής
- όταν δημιουργείται νέα προσωρινή μεταβλητή
- όταν συναντάμε δήλωση νέας συνάρτησης
- όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης

**add\_argument(new\_arg):** όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης.

# Αλλαγές στις συν/σεις του συντακτικού που αφορούν των πίνακα συμβόλων.

*Κυρίως οι αλλαγές αφορούν συναρτήσεις όπου έχουν να κάνουν με τη δημιουργία κάποιου **scope** , **entity** , **argument**.* Έτσι δεν έχουν γίνει αλλαγές στις συναρτήσεις που αφορούν δομές ελέγχου όπως είναι η δομή του if επίσης δεν έχουν γίνει αλλαγές σε συναρτήσεις που αφορούν αριθμητικές ή λογικές πράξεις.

Γενικότερα έχει προστεθεί κώδικας που δημιουργεί κάθε φορά το κατάλληλο αντικείμενο **scope** , **entity** , **argument** σημαντικό είναι όταν έχουμε συναρτήσεις να βρίσκουμε το εκάστοτε **offset**.

Αρχικά όπως είναι λογικό έχουμε προσθέσει **ένα νέο scope με το που ξεκινήσει ο συντακτικός έλεγχος** στην συνάρτηση **startRule()** αφού εκεί δημιουργείται το **scope της main**.

Στη συνέχεια στην κλήση της **main\_function** που ακολουθεί την **startRule()** έχουμε δημιουργήσει ένα νέο **entity** αφού επρόκειτο για μια συνάρτηση ιδιαίτερα για μια κυρία συνάρτηση έτσι όπως φαίνεται στη διπλανή εικόνα.

Δημιουργούμε αυτό το νέο **entity** τύπου **function** και με όνομα το εκάστοτε όνομα της συνάρτησης. Στη συνέχεια προσθέτουμε αυτό το **entity** στη λίστα με τα **entities** μας, και δημιουργούμε στο **παραπάνω επίπεδο ένα νέο scope** με το όνομα της συνάρτησης. Στη συνέχεια βρίσκουμε τις τιμές των **startQuad**, **framelength**. Τέλος αφού έχει τελειώσει και έχει κλείσει το **block** της συνάρτησης τυπώνουμε τον πίνακά μας και διαγράφουμε το **scope**.

```
#pinakas symbolwn def def_main_function()
#Ftiachnw ena Entity
myNewEntity = Entity()
myNewEntity.type = 'function'
myNewEntity.name = func_name
#####ent.function.type = 'Function'
add_entity(myNewEntity)
add_scope(func_name)

declarations()

while(results_lex[2] == def_key):
    def_function()

#endiamesos (arxi kai telos block sel 8)
genQuad('begin_block',func_name,'_', '_')

find_startQuad() #pinakas symbolwn briskw to Quad.

statements()
find_framelength() #pinakas symbolwn

genQuad('end_block',func_name,'_', '_')

write_Symbol_Table(pinakas_sum) #pinakas symbolwn
delete_scope() #pinakas symbolwn
```



Στη συνέχεια έχουμε κάνει παρόμοια διαδικασία δηλαδή παρόμοιες αλλαγές για την function() όπου αφορά απλές και όχι κύριες συναρτήσεις. Έτσι δημιουργούμε ένα νέο entity όπως κάναμε και πριν. Αφού εδώ μιλάμε για **απλές συναρτήσεις** που **μπορεί να περιέχουν παραμέτρους** καλούμε την id\_list() όπως και πριν αφού το περιγράφει η γραμματική της γλώσσας μας. Έτσι έχουμε προσθέσει σχεδόν τον ίδιο κώδικα με πριν απλά εδώ με την ιδιαιτερότητα ότι

```
#pinakas symbolwn
myNewEntity = Entity()
myNewEntity.type = 'function'
myNewEntity.name = func_name
add_entity(myNewEntity)
id_list('argum')

def function():

if(results_lex[2] == right_parenthesis):
    linetemp = line
    results_lex = lex()

if(results_lex[2] == colon):
    linetemp = line
    results_lex = lex()

if(results_lex[2] == left_curly_bracket):
    linetemp = line
    results_lex = lex()

#pinakas symbolwn
add_scope(func_name)
add_parameters()

declarations()
while(results_lex[2] == def_key):
    def_function()

#endiamesos (arxi kai telos block sel 8
genQuad('begin_block',func_name,'_', '_')
find_startQuad() #pinakas symbolwn
statements()
find_framelength() #pinakas symbolwn
genQuad('end_block',func_name,'_', '_')

#pinakas symbolwn
write_Symbol_Table(pinakas_sum)
delete_scope()
```

προσθέτουμε και τις παραμέτρους που μπορεί να έχει μια συνάρτηση όπως και πριν **δημιουργούμε στο παραπάνω επίπεδο ένα νέο scope** με το όνομα της συνάρτησης . Στη συνέχεια βρίσκουμε τις τιμές των startQuad, framelength . Τέλος αφού έχει τελειώσει και έχει κλείσει το block της συνάρτησης τυπώνουμε τον πίνακά μας και διαγράφουμε το scope.

Επιπλέον έχει γίνει τροποποίηση της βοηθητικής συναρτήσεων του ενδιαμέσου κώδικα newTemp() όπου δημιουργεί μια νέα προσωρινή μεταβλητή %i αφού **κάθε φορά που δημιουργείται μια νέα προσωρινή μεταβλητή πρέπει να δημιουργούμε ένα νέο entity.**

Δημιουργούμε το νέο entity με τις κατάλληλες παραμέτρους και το προσθέτουμε στη λίστα μας όπως φαίνεται στην παρακάτω εικόνα.

```

#Ftiachw ta %1 , %i pou xreiazomai.
def newTemp():
    global i
    global variablesList
    temporary = ""
    tempVar = temporary+str(i)
    i = i + 1

    #pinakas symbolwm
    #Afou ftiachnw mia proswrini metabliti ftiachnw ena entity gia auti.
    myNewEntity = Entity()
    myNewEntity.name = tempVar
    myNewEntity.type = 'temporary'
    myNewEntity.tempVar.offset = find_offset()
    add_entity(myNewEntity)

    return tempVar

```

Επιπλέον θα τροποποιήσαμε την συνάρτηση `id_list` ώστε να δημιουργούμε είτε κάποια νέα arguments εάν έχει καλεστεί η συνάρτηση με παράμετρο το string = "argum" από την συνάρτηση `function()`.

Αν όμως γίνεται κάποια **δήλωση μεταβλητής τότε πρέπει να δημιουργήσουμε ένα νέο *entity*** με τα κατάλληλα στοιχεία όπως φαίνεται στην εικόνα στην παρακάτω σελίδα.

```

if(results_lex[2] == id):
    func_name = results_lex[1]
    results_lex = lex()
    linetemp = line

    # str == "argum" to kalese i def_function etsi einai ena argument
    if (str == "argum"):

        #pinakas sumbolwn ftiaxnw argument
        myNewArguent = Argument()
        myNewArguent.name = func_name
        add_argument(myNewArguent)

    #to kalese i declaration_line etsi einai Entity
    else:

        #pinakas sumbolwn ftiaxnw entity
        myNewEntity = Entity()
        myNewEntity.name = func_name
        myNewEntity.variable.offset = find_offset()
        myNewEntity.type = 'variable'
        add_entity(myNewEntity)

while(results_lex[2] == comma):
    results_lex = lex()

    if(results_lex[2] == id and linetemp == line ):
        func_name = results_lex[1]
        results_lex = lex()

        if (str == "argum"):

            #pinakas sumbolwn ftiaxnw argument.
            myNewArguent = Argument()
            myNewArguent.name = func_name
            add_argument(myNewArguent)
        else:
            #pinakas sumbolwn ftiaxnw entity
            myNewEntity = Entity()
            myNewEntity.name = func_name
            myNewEntity.variable.offset = find_offset()
            myNewEntity.type = 'variable'
            add_entity(myNewEntity)

```

Τέλος έχουμε στη συνάρτηση call\_main\_part()τυπώνουμε τον πίνακα και διαγράφουμε το τελευταίο scope.

```

def call_main_part():
    #pinakas sumbolwn
    write_Symbol_Table(pinakas_sum)
    #os.system("pause")
    delete_scope()

```

# Αναφορά ελέγχου πίνακα συμβόλων.

Για τον έλεγχο της σωστής λειτουργίας του πίνακα συμβόλου **δεν χρησιμοποιήθηκε το ίδιο πρόγραμμα ως είσοδο** που είχε χρησιμοποιηθεί στις προηγούμενες αναφορές ελέγχου. Στην παρούσα αναφορά χρησιμοποιήσαμε το αρχείο fibonacci.cpy.

Παρακάτω φαίνεται το αρχείο που δώσαμε ως είσοδο και δεξιά ο πίνακας σύμβολο τυπωμένος.

```
def main_fibonacci():
#{
  #declare x
  def fibonacci(x):
  #{
    if (x<=1):
      return(x);
    else:
      return (fibonacci(x-1)+fibonacci(x-2));
  #}
  x = int(input());
  print(fibonacci(x));
#}

if __name__ == "__main__":
  main_fibonacci();
```

#####

SCOPE: name: fibonacci    nestingLevel:2

ENTITIES:

ENTITY: name:x    type:parameter    offset:12

ENTITY: name:%0    type:temporary    offset:16

ENTITY: name:%1    type:temporary    offset:20

ENTITY: name:%2    type:temporary    offset:24

ENTITY: name:%3    type:temporary    offset:28

ENTITY: name:%4    type:temporary    offset:32

SCOPE: name:main\_fibonacci    nestingLevel:1

ENTITIES:

ENTITY: name:x    type:variable    offset:12

ENTITY: name: fibonacci    type:function    startQuad:2    frameLength:36

ARGUMENTS:

ARGUMENT: name:x

SCOPE: name:"\_\_main\_\_"    nestingLevel:0

ENTITIES:

ENTITY: name:main\_fibonacci    type:function    startQuad:0    frameLength:0

ARGUMENTS:

#####

Αρχικά δημιουργείται ένα scope αυτό της main στη συνέχεια το πρόγραμμά μας διαβάσει την κύρια συνάρτηση και δημιουργεί ένα entity για αυτή, αφού δημιουργήσει αυτό το entity δημιουργεί ένα νέο scope στο παραπάνω επίπεδο όπως φαίνεται στην εικόνα, στη συνέχεια δημιουργούμε ένα νέο entity αφού συναντάμε μια δήλωση μεταβλητής της μεταβλητής x . Στη συνέχεια όμως βρίσκουμε μια νέα συνάρτηση τη συνάρτηση fibonacci έτσι δημιουργούμε ένα νέο entity και ένα νέο scope στο παραπάνω επίπεδο όπως ακριβώς και στην εικόνα. Επιπλέον δημιουργούμε ένα νέο argument αφού η συνάρτηση fibonacci δέχεται ως όρισμα το argument x . Στη συνέχεια δημιουργούνται 5 νέα entities αφού καλούνται διάφορες άλλες συναρτήσεις που δημιουργούν κατά την παραγωγή ενδιάμεσου κώδικα προσωρινές μεταβλητές.

Στη συνέχεια θα γίνει διαγραφή ένα προς ένα του ανώτερου κάθε φορά scope μέχρι να φτάσουμε στο τέλος του προγράμματος στο scope δηλαδή της main.



# Εισαγωγή στον τελικό κώδικα.

Έτσι φτάσαμε αισίως στο τελευταίο στάδιο του μεταφραστή μας που είναι η παραγωγή του τελικού κώδικα βασίζεται πάνω στον ενδιάμεσο και στον πίνακα συμβόλων αφού **από μια εντολή ενδιάμεση κώδικα παράγουμε τις κατάλληλες ή καλύτερα τις αντίστοιχες εντολές του τελικού κώδικα.** Έτσι λοιπόν θα δημιουργήσουμε τελικό κώδικα για τον επεξεργαστή **RISC V**.

Ο παραπάνω επεξεργαστής περιέχει κάποιους καταχωρητές που είναι χρήσιμοι για την παραγωγή του τελικού κώδικα έτσι λοιπόν παραδείγματος χάρη για τις προσωρινές μεταβλητές χρησιμοποιούνται οι καταχωρητές `t0..t6`, επιπλέον οι καταχωρητές που αφορούν κλήσεις συναρτήσεων είναι οι `s1...s11` , και οι καταχωρητές ορισμάτων `a0...a7`.

Τέλος σημαντικοί είναι οι καταχωρητές που αφορούν τις ιδιότητες των συναρτήσεων δηλαδή των `entity` αυτοί είναι:

`stack pointer sp`

`frame pointer fp`

`return address ra`

`global pointer gp`

Έτσι όπως βλέπουμε και στις διαφάνειες η παραγωγή του τελικού κώδικα διαφέρει κατά περίπτωση γενικότερα όμως αποτελείται από τις εντολές `lw,sw,li,mv,addi`. Έτσι μερικές από αυτές τις εντολές χρειάστηκε να τις δημιουργήσουμε.

# Βοηθητικές συναρτήσεις.

Έτσι όπως είπαμε και παραπάνω δημιουργήσαμε κάποιες βοηθητικές συναρτήσεις που παρουσιάζονται στον παρακάτω πίνακα.

<code>def gnlvcode(name)</code>	<ul style="list-style-type: none"><li>• Μεταφέρει στον <code>t0</code> την διεύθυνση της μη τοπικής μεταβλητής <code>name</code>.</li><li>• Από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει.</li></ul>
<code>def loadvr(v,r)</code>	Μεταφορά δεδομένων στον καταχωρητή <code>r</code> .
<code>def storerv(r,v)</code>	Μεταφορά δεδομένων από τον καταχωρητή <code>r</code> στη μνήμη (μεταβλητή <code>v</code> ).

Οι παραπάνω συναρτήσεις δεν είναι τόσο εύκολες όσο φαίνεται στον παραπάνω πίνακα, αφού για αυτές πρέπει κανείς να διακρίνει πολλές περιπτώσεις.

Επιπλέον δημιουργήθηκε και μια νέα στην συνάρτηση η `search_entity(name)` όπου ψάχνει μια παράμετρο με όνομα `name` μέσα στη λίστα με τα `entities`. Αυτή η συνάρτηση παρουσιάζεται στον πίνακα σύμβολο στις δικές σας σημειώσεις αλλά κρίναμε σκόπιμο ότι είναι καλό να το αναφέρουμε εδώ αφού χρησιμοποιείται μόνο στην παραγωγή του τελικού κώδικα.

```
836 #Psaxnei ena entity - argument , me basi to onoma tou.
837 def search_entity(name):
838     global scopesTable
839
840     scope_on_top =scopesTable[len(scopesTable)-1]
841     while scope_on_top:
842         for entities in scope_on_top.ListEntity:
843             if(entities.name == name):
844                 return (scope_on_top,entities)
845             scope_on_top=scopesTable[len(scopesTable)-2]
846
847     print("Den uparxei entity me onoma " + str(x))
848     exit()
849
```

## gnlvcode :

```
853 #gnlvcode sel 22 - 24
854 #μεταφέρει στον t0 την διεύθυνση μιας μη τοπικής μεταβλητής
855 #από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική
856 #μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει
857 def gnlvcode(name):
858     global telikos_arxeio
859
860     telikos_arxeio.write("lw" + " t0" + "-4(sp)" + "\n") #στοίβα του γονέα
861
862     #ψαχνουμε στον πίνακα συμβόλων να βρει το "ζευγαρι" (scope_i,entity_i) της μη τοπικής μεταβλητής.
863     (scope_i,entity_i)=search_comb(name)
864
865     #βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή
866     floors= scopesTable[len(scopesTable)-1].nestingLevel - scope_i.nestingLevel;
867
868     #Αφαιρω ένα επίπεδο αφού έχω "παι σε αυτο εκτελοντας την εντολη lw t0, 4(sp) παραπάνω".
869     floors=floors -1
870
871     #όσες φορές χρειαστεί:
872
873     for i in range(floors):
874
875         #στοίβα του προγόνου που έχει τη μεταβλητή οσες φορές χρειάζεται για να φτασω.
876         telikos_arxeio.write("lw" + " t0" + "-4(t0)" + "\n")
877
878         if entity_i.type=='parameter':
879             #διεύθυνση της μη τοπικής μεταβλητής.
880             telikos_arxeio.write("addi" + " t0"+"","t0" + "-"+str(entity_i.parameter.offset) + "\n")
881
882         if entity_i.type=='variable':
883
884             #διεύθυνση της μη τοπικής μεταβλητής.
885             telikos_arxeio.write("addi" + " t0"+"","t0" + "-"+str(entity_i.variable.offset) + "\n")
```



## loadvr(v,r):

```
887 #loadvr sel 26-31
888 def loadvr(v,r):
889     global telikos_arxeio
890     global scopesTable
891     #Αν v είναι σταθερά li tr,v sel 26
892     if v.isdigit():
893         telikos_arxeio.write("li " +str(r) + "," +str(v) + " \n")
894
895     #Αν v είναι καθολική μεταβλητή
896     else:
897         (scope_i,entity_i)=search_entity(v)
898
899         #Αν v είναι καθολική μεταβλητή δηλαδή ανήκει στο κυρίως πρόγραμμα. sel 26.
900         #lw tr, offset gp
901         #scope_i.nestingLevel==0 to scope tis main.
902         if scope_i.nestingLevel==0 and entity_i.type=='variable':
903             telikos_arxeio.write("lw " +str(r) + "," + "-" +str(entity_i.variable.offset)+"(gp)"+" \n")
904
905         #Αν v είναι καθολική μεταβλητή δηλαδή ανήκει στο κυρίως πρόγραμμα. sel 26.
906         #lw tr, offset gp
907         elif scope_i.nestingLevel==0 and entity_i.type=='temporary':
908             telikos_arxeio.write("lw " +str(r) + "," + "-" +str(entity_i.tempVar.offset)+"(gp)"+" \n")
909
```

```
910 #Αν η v έχει δηλωθεί στη συνάρτηση που αυτή τη στιγμή εκτελείται και είναι τοπική sel 28
911 #μεταβλητή, ή τυπική παράμετρος , ή προσωρινή μεταβλητή.
912 #Δηλαδή το βαθος φωλιάσματος είναι ίσο με την συνάρτηση που εκτελείται.
913 ~ elif scope_i.nestingLevel == scopesTable[len(scopesTable)-1].nestingLevel:
914
915     #τοπική μεταβλητή
916 ~     if entity_i.type=='variable':
917
918         telikos_arxeio.write("lw " +str(r) + "," + "-" +str(entity_i.variable.offset)+"(sp)"+" \n")
919
920     #τυπική παράμετρος
921 ~     elif entity_i.type=='parameter':
922
923         telikos_arxeio.write("lw " +str(r) + "," + "-" +str(entity_i.parameter.offset)+"(sp)"+" \n")
924
925     #προσωρινή μεταβλητή
926 ~     elif entity_i.type=='temporary':
927
928         telikos_arxeio.write("lw " +str(r) + "," + "-" +str(entity_i.tempVar.offset)+"(sp)"+" \n")
929
930     #αν η v έχει δηλωθεί σε κάποιο πρόγνονο και εκεί είναι τοπική μεταβλητή, ή τυπική παράμετρος sel 30
931     #gnlvcode /// lw tr,(t0)
932     #Αρα το βαθος φωλιάσματος είναι μικρότερο από της τρεχουσας συναρτησης.
933 ~     elif scope_i.nestingLevel < scopesTable[len(scopesTable)-1].nestingLevel:
934
935         #τοπική μεταβλητή
936 ~         if entity_i.type=='variable':
937             gnlvcode(v)
938             telikos_arxeio.write("lw " +str(r) + "," + "(t0)" + " \n")
939
940         #τυπική παράμετρος
941 ~         elif entity_i.type=='parameter':
942             gnlvcode(v)
943             telikos_arxeio.write("lw " +str(r) + "," + "(t0)" + " \n")
```

## storerv(r,v):

```
946 #storerv sel 33-36
947 #μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v)
948 def storerv(r,v):
949     global telikos_arxeio
950
951     (scope_i,entity_i)=search_entity(v)
952
953     #αν v είναι καθολική μεταβλητή - δηλαδή ανήκει στο κυρίως πρόγραμμα sel 34
954     #sw tr, -offset(gp)
955     if scope_i.nestingLevel==0 and entity_i.type=='variable':
956
957         telikos_arxeio.write("sw " +str(r) + "," + "-" +str(entity_i.variable.offset)+"(gp)"+"\\n")
958
959     elif scope_i.nestingLevel==0 and entity_i.type=='temporary':
960
961         telikos_arxeio.write("sw " +str(r) + "," + "-" +str(entity_i.tempVar.offset)+"(gp)"+"\\n")
962
963     #αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος και βάθος φωλιάσματος
964     #ίσο με το τρέχον, ή προσωρινή μεταβλητή. sel 35
965     #sw tr, -offset(sp)
966
967     elif scope_i.nestingLevel == scopesTable[len(scopesTable)-1].nestingLevel:
968
969         #τοπική μεταβλητή
970         if entity_i.type=='variable':
971
972             telikos_arxeio.write("sw " +str(r) + "," + "-" +str(entity_i.variable.offset)+"(sp)"+"\\n")
973
974         #τυπική παράμετρος
975         elif entity_i.type=='parameter':
976
977             telikos_arxeio.write("sw " +str(r) + "," + "-" +str(entity_i.parameter.offset)+"(sp)"+"\\n")
978
979         #προσωρινή μεταβλητή
980         elif entity_i.type=='temporary':
981
982             telikos_arxeio.write("sw " +str(r) + "," + "-" +str(entity_i.tempVar.offset)+"(sp)"+"\\n")
```

```
984 #αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος και βάθος φωλιάσματος μικρότερο από το τρέχον sel 36
985 #gnlvcode(v) ///// sw tr, (t0)
986 elif scope_i.nestingLevel < scopesTable[len(scopesTable)-1].nestingLevel:
987
988     #τοπική μεταβλητή
989     if entity_i.type=='variable':
990
991         gnlvcode(v)
992         telikos_arxeio.write("sw " +str(r) + "," + "(t0)" + "\\n")
993
994     #τυπική παράμετρος
995     elif entity_i.type=='parameter':
996
997         gnlvcode(v)
998         telikos_arxeio.write("sw " +str(r) + "," + "(t0)" + "\\n")
```

# Η κύρια συνάρτηση του τελικού κώδικα

## generate\_final():

```
parameter_list = []
parameter_flag = False
def generate_final():
    global quadList, j
    global telikos_arxeio
    global scopesTable ,parameter_list,parameter_flag
    for i in range(len(quadList)):
        telikos_arxeio.write("\n"+"L" + str(quadList[i][0]) + ": \n")
#~~~~~Είσοδος και έξοδος δεδομένων SEL 15~~~~~
        if (quadList[i][1] == "out"):
            loadvr(quadList[i][2], "t1")
            telikos_arxeio.write('mv a0,t1'+'\n')
            telikos_arxeio.write("li a7,1"+'\n')
            telikos_arxeio.write("ecall"+'\n')

        elif (quadList[i][1] == "inp"):
            telikos_arxeio.write("li a7,5"+'\n')
            telikos_arxeio.write("ecall"+'\n')
            telikos_arxeio.write('mv t1,a0'+'\n')
            storerv("t1",quadList[i][2])
```

Όπως φαίνεται και στην παραπάνω εικόνα ή η συνάρτηση αυτή αποτελείται από τις αντίστοιχες περιπτώσεις με τη σειρά που προβάλλονται στις σημειώσεις, ξεκινώντας λοιπόν από τις **εισόδου- εξόδου**.

Έπειτα συνεχίσαμε με την περίπτωση του **τερματισμού προγράμματος με επιστροφή τιμής 0**.

Συνεχίζουμε τις εντολές αλμάτων.

```
#~~~~~Εντολές Αλμάτων SEL 38~~~~~  
#loadvr(x,t1)  
#loadvr(y,t2)  
  
elif (quadList[i][1] == "jump"):  
    telikos_arxeio.write("j L"+str(quadList[i][4])+"\n")  
  
elif (quadList[i][1] == ">"):  
    loadvr(quadList[i][2],"t1")  
    loadvr(quadList[i][3],"t2")  
    telikos_arxeio.write("bgt,t1,t2,L"+str(quadList[i][4])+"\n")  
  
elif (quadList[i][1] == ">="):  
    loadvr(quadList[i][2],"t1")  
    loadvr(quadList[i][3],"t2")  
    telikos_arxeio.write("bge,t1,t2,L"+str(quadList[i][4])+"\n")  
  
elif (quadList[i][1] == "<"):  
    loadvr(quadList[i][2],"t1")  
    loadvr(quadList[i][3],"t2")  
    telikos_arxeio.write("blt,t1,t2,L"+str(quadList[i][4])+"\n")  
  
elif (quadList[i][1] == "<="):  
    loadvr(quadList[i][2],"t1")  
    loadvr(quadList[i][3],"t2")  
    telikos_arxeio.write("ble,t1,t2,L"+str(quadList[i][4])+"\n")  
  
elif (quadList[i][1] == "!="):  
    loadvr(quadList[i][2],"t1")  
    loadvr(quadList[i][3],"t2")  
    telikos_arxeio.write("bne,t1,t2,L"+str(quadList[i][4])+"\n")  
  
elif (quadList[i][1] == "=="):  
    loadvr(quadList[i][2],"t1")  
    loadvr(quadList[i][3],"t2")  
    telikos_arxeio.write("beq,t1,t2,L"+str(quadList[i][4])+"\n")  
  
#~~~~~ΤΕΛΟΣ Εντολές Αλμάτων SEL 38~~~~~
```

Στη συνέχεια ακολουθούν οι περιπτώσεις της **εκχώρησης** και των **αριθμητικών πράξεων**.

```
#~~~~~ΤΕΛΟΣ Εντολές Αλμάτων SEL 38~~~~~  
  
#Εκχώρηση sel 39  
elif (quadList[i][1] == "="):  
    loadvr(quadList[i][2], "t1")  
    storerv("t1", quadList[i][4])  
  
#~~~~~Εντολές Αριθμητικών Πράξεων sel 40~~~~~  
  
#loadvr(x, t1)  
#loadvr(y, t2)  
#op, t1, t1, t2  
#storerv(t1, z)  
  
elif (quadList[i][1] == "//"):  
    loadvr(quadList[i][2], "t1")  
    loadvr(quadList[i][3], "t2")  
    telikos_arxeio.write("div,t1,t1,t2"+"\\n")  
    storerv("t1", quadList[i][4])  
  
elif (quadList[i][1] == "*"):  
    loadvr(quadList[i][2], "t1")  
    loadvr(quadList[i][3], "t2")  
    telikos_arxeio.write("mul,t1,t1,t2"+"\\n")  
    storerv("t1", quadList[i][4])  
  
elif (quadList[i][1] == "-"):  
    loadvr(quadList[i][2], "t1")  
    loadvr(quadList[i][3], "t2")  
    telikos_arxeio.write("sub,t1,t1,t2"+"\\n")  
    storerv("t1", quadList[i][4])  
  
elif (quadList[i][1] == "+"):  
    loadvr(quadList[i][2], "t1")  
    loadvr(quadList[i][3], "t2")  
    telikos_arxeio.write("add,t1,t1,t2"+"\\n")  
    storerv("t1", quadList[i][4])  
  
#~~~~~ΤΕΛΟΣ Αριθμητικών Πράξεων sel 40~~~~~
```

Παρομοίως ακολουθούν η περίπτωση **επιστροφής τιμή συνάρτησης** και η περίπτωση των **παραμέτρων της νέας συνάρτησης**.

```
#~~~~~Επιστροφή Τιμής Συνάρτησης sel 41~~~~~

#αποθηκεύεται ο x στη διεύθυνση που είναι αποθηκευμένη στην 3 η θέση του
#εγγραφήματος δραστηριοποίησης
elif (quadList[i][1] == "retv"):
    loadvr(quadList[i][2],"t1")
    telikos_arxeio.write("lw t0,-8(sp)\n")
    telikos_arxeio.write("sw t1,(t0)\n")
    telikos_arxeio.write("lw ra, (sp)\n") # sel 61
    telikos_arxeio.write("jr ra \n")      #Μέσω του ra επιστρέφουμε στην καλούσα.

#~~~~~ Παράμετροι Συνάρτησης sel 42-54~~~~~

elif (quadList[i][1] == "par"):
    #πριν από την πρώτη παράμετρο, τοποθετούμε τον $fp να δείχνει
    #στην στοίβα της συνάρτησης που θα δημιουργηθεί
    if parameter_flag == False:
        temp = i
        Counter = 0
        #Βρίσκω ποιος καλεσε την συναρτηση .
        while temp:
            if (quadList[temp][1] == 'call'):
                caller_name = str(quadList[temp][2])
                break
            Counter = Counter + 1
            parameter_list.append(Counter)
            temp=temp+1

        (scope_i,entity_i)=search_entity(caller_name)
        #βρίσκω το frame length αυτής.
        telikos_arxeio.write("addi fp,sp,"+str(entity_i.function.frameLength)+"\n")
        parameter_flag = True
    #par,x,CV , sel 44
    if (quadList[i][3] == "CV"):
        loadvr(quadList[i][2],"t0")      #Loadvr(x,t0) sw t0,-(12+4i)(fp)
        parameter_index = parameter_list.pop(0) - 1
        #i = j ο αύξων αριθμός της παραμέτρου
        j = 12+4*parameter_index
        telikos_arxeio.write("sw t0,"+"-"+str(j)+"(fp) \n")

    #par,x,RET,_
    #γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης
    #με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή
    elif (quadList[i][3] == "RET"):
        #Ψάχνω την συναρτηση για να βρω το offset της ΠΡΟΣΩΡΙΝΗΣ ΜΕΤΑΒΛΗΤΗΣ...
        (scope_i,entity_i)=search_entity(quadList[i][2])
        telikos_arxeio.write("addi t0,sp,-"+str(entity_i.tempVar.offset)+"\n")
        telikos_arxeio.write("sw t0,-8(fp)\n")
```

Τέλος έχουμε την περίπτωση της **κλήσης μιας συνάντησης** που διακρίνεται και αυτή από περιπτώσεις.

```
~~~~~ Κλήση Συνάρτησης sel 55 - 61 ~~~~~
#call,_,_,f
elif (quadList[i][1] == "call"):
    parameter_flag = False
    parameter_list = []

    (callee_scope,callee_entity)=search_entity(quadList[i][2])

    #αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα
    if scopesTable[len(scopesTable)-1].nestingLevel==callee_entity.function.nestingLevel:
        telikos_arxeio.write("lw t0,-4(sp)\n")
        telikos_arxeio.write("sw t0,-4(fp)\n")

    #αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας
    else:
        telikos_arxeio.write("sw sp,-4(fp)\n")

    #στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα σελ 60
    telikos_arxeio.write("addi sp,sp,"+str(callee_entity.function.frameLength)+"\n")
    #καλούμε τη συνάρτηση σελ 60
    telikos_arxeio.write("jal "+" L"+str(callee_entity.function.startQuad)+"\n")
    #όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα σελ 60
    telikos_arxeio.write("addi sp,sp,"+str(callee_entity.function.frameLength)+"\n")

    #στην αρχή κάθε συνάρτησης - "υποπρογράμματος" αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης
    #την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον ra
    elif (quadList[i][1] == "begin_block"):

        #Αν είμαστε στο begin block της MAIN.
        #στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος
        #~~~~~ MAIN σελ 62 ~~~~~

        if(scopesTable[len(scopesTable)-1].nestingLevel==0):
            #Παω στην πρώτη γραμμή στην θέση 3 και βαζω το νουμερο της 4αδας
            telikos_arxeio.seek(3)
            telikos_arxeio.write(str((quadList[i][0])))
            telikos_arxeio.seek(0, os.SEEK_END)
            #πρέπει να κατεβάσουμε τον sp κατά frameLength της main
            telikos_arxeio.write("addi sp,sp,"+str(find_offset())+"\n")
            #και να σημειώσουμε στον gp το εγγράφημα δραστηριοποίησης της main ώστε να
            #έχουμε εύκολη πρόσβαση στις global μεταβλητές
            telikos_arxeio.write("mv gp,sp\n")

        #Αν είμαστε σε οποιαδήποτε άλλη συνάρτηση.
        else:
            telikos_arxeio.write("sw ra,(sp)\n") #sel 61 διεύθυνση επιστροφής της συναρτησης
```

Και η διαχείριση του begin block ενός προγράμματος.

```
#στην αρχή κάθε συνάρτησης - "υποπρογράμματος" αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης
#την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον ra
elif (quadList[i][1] == "begin_block"):

    #Αν είμαστε στο begin block της MAIN.
    #στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος
    #~~~~~ MAIN σελ 62 ~~~~~

    if(scopesTable[len(scopesTable)-1].nestingLevel==0):
        #Παω στην πρώτη γραμμή στην θέση 3 και βαζω το νουμερο της 4αδας
        telikos_arxeio.seek(3)
        telikos_arxeio.write(str((quadList[i][0])))
        telikos_arxeio.seek(0, os.SEEK_END)
        #πρέπει να κατεβάσουμε τον sp κατά framelength της main
        telikos_arxeio.write("addi sp,sp,"+str(find_offset())+"\n")
        #και να σημειώσουμε στον gp το εγγράφημα δραστηριοποίησης της main ώστε να
        #έχουμε εύκολη πρόσβαση στις global μεταβλητές
        telikos_arxeio.write("mv gp,sp\n")

    #Αν είμαστε σε οποιαδήποτε άλλη συνάρτηση.
    else:
        telikos_arxeio.write("sw ra,(sp)\n") #σελ 61 διεύθυνση επιστροφής της συνάρτησης

elif (quadList[i][1] == "end_block"):
    if(scopesTable[len(scopesTable)-1].nestingLevel!=0):
        telikos_arxeio.write("lw ra,(sp)\n")
        telikos_arxeio.write("jr ra\n")

#Στο τέλος κάθε end block διαγραφω τα quads για να μη τα ξανα παραγω.
quadList = []
```



Για τον έλεγχο της σωστής λειτουργίας του πίνακα συμβόλου χρησιμοποιήθηκε το ίδιο πρόγραμμα ως είσοδο που είχε χρησιμοποιηθεί την δοκιμή του πίνακα συμβόλων το αρχείο **fibonacci.cpy**.

```
def main_fibonacci():  
#{  
    #declare x  
    def fibonacci(x):  
        #{  
            if (x<=1):  
                return(x);  
            else:  
                return (fibonacci(x-1)+fibonacci(x-2));  
        }  
        x = int(input());  
        print(fibonacci(x));  
    }  
}  
  
if __name__ == "__main__":  
    main_fibonacci();
```

#####

SCOPE: name: fibonacci nestingLevel: 2

ENTITIES:

ENTITY: name: x type: parameter offset: 12

ENTITY: name: %0 type: temporary offset: 16

ENTITY: name: %1 type: temporary offset: 20

ENTITY: name: %2 type: temporary offset: 24

ENTITY: name: %3 type: temporary offset: 28

ENTITY: name: %4 type: temporary offset: 32

SCOPE: name: main\_fibonacci nestingLevel: 1

ENTITIES:

ENTITY: name: x type: variable offset: 12

ENTITY: name: fibonacci type: function startQuad: 2 frameLength: 36

ARGUMENTS:

ARGUMENT: name: x

SCOPE: name: "\_\_main\_\_" nestingLevel: 0

ENTITIES:

ENTITY: name: main\_fibonacci type: function startQuad: 0 frameLength: 0

ARGUMENTS:

#####

Έτσι στην παρακάτω εικόνα βλέπουμε στην αριστερή της πλευρά τον ενδιάμεσο κώδικα που παράγεται και στην δεξιά πλευρά αυτής τον τελικό κώδικα που παράγεται έχουμε χρωματίσει τις γραμμές δηλαδή τις τετράδες του ενδιάμεσου κώδικα με τέτοιο τρόπο που αντιστοιχούν σε χρώμα με τις γραμμές του τελικού κώδικα ώστε να είναι λίγο πιο εύκολο κατανοητό.

Αρχικά στην πρώτη γραμμή βλέπουμε ένα άλμα αυτό παράγεται από την κλίση της main της.

1: begin_block fibonacci _ _	j L24	L9:	L16:	L25:
2: <= x 1 4		lw t0,-4(sp)	lw ra,(sp)	sw sp,-4(fp)
3: jump _ _ 6		sw t0,-4(fp)	jr ra	addi sp,sp,20
4: retv x _ _	L1:	addi sp,sp,36		jal L17
5: jump _ _ 16	sw ra,(sp)	jal L1	L17:	addi sp,sp,-20
6: - x 1 %0		addi sp,sp,-36	sw ra,(sp)	
7: par %0 CV _	L2:			L26:
8: par %1 RET _	lw t1,-12(sp)	L10:	L18:	li a0,0
9: call fibonacci _ _	li t2,1	lw t1,-12(sp)	li a7,5	li a7,93
10: - x 2 %2	ble,t1,t2,L4	li t2,2	ecall	ecall
11: par %2 CV _		sub,t1,t1,t2	mv t1,a0	
12: par %3 RET _	L3:	sw t1,-24(sp)	sw t1,-12(sp)	
13: call fibonacci _ _	j L6			
14: + %1 %3 %4		L11:	L19:	
15: retv %4 _ _	L4:	addi fp,sp,36	addi fp,sp,36	
16: end_block fibonacci _ _	lw t1,-12(sp)	lw t0,-24(sp)	lw t0,-12(sp)	
17: begin_block main_fibonacci _ _	lw t0,-8(sp)	sw t0,-12(fp)	sw t0,-12(fp)	
18: inp x _ _	sw t1,(t0)			
19: par x CV _	lw ra, (sp)	L12:	L20:	
20: par %5 RET _	jr ra	addi t0,sp,-28	addi t0,sp,-16	
21: call fibonacci _ _		sw t0,-8(fp)	sw t0,-8(fp)	
22: out %5 _ _	L5:			
23: end_block main_fibonacci _ _	j L16	L13:	L21:	
24: begin_block "__main__" _ _		lw t0,-4(sp)	sw sp,-4(fp)	
25: call main_fibonacci _ _	L6:	sw t0,-4(fp)	addi sp,sp,36	
26: halt _ _	lw t1,-12(sp)	addi sp,sp,36	jal L1	
27: end_block "__main__" _ _	li t2,1	jal L1	addi sp,sp,-36	
	sub,t1,t1,t2	addi sp,sp,-36		
	sw t1,-16(sp)		L22:	
		L14:	lw t1,-16(sp)	
	L7:	lw t1,-20(sp)	mv a0,t1	
	addi fp,sp,36	lw t2,-28(sp)	li a7,1	
	lw t0,-16(sp)	add,t1,t1,t2	ecall	
	sw t0,-12(fp)	sw t1,-32(sp)		
			L23:	
	L8:	L15:	lw ra,(sp)	
	addi t0,sp,-20	lw t1,-32(sp)	jr ra	
	sw t0,-8(fp)	lw t0,-8(sp)		
		sw t1,(t0)	L24:	
		lw ra, (sp)	addi sp,sp,12	
		jr ra	mv gp,sp	

Στη συνέχεια βλέπουμε ότι ξεκινά η συνάρτηση fibonacci και έτσι πρέπει να αποθηκεύσουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής την οποία έχει τοποθετήσει στον ra.

Στην δεύτερη τετράδα βλέπουμε μια συνθήκη έτσι αποτελεί μια εντολή άλματος οπότε πρέπει να παράξουμε τις κατάλληλες γραμμές τελικού κώδικα όπως φαίνεται και παραπάνω.

Στην τέταρτη τετράδα βλέπουμε μια εντολή επιστροφής τιμής να συνάρτησης και έτσι παράγεται ο τελικός κώδικας προσοχή όμως εδώ πρέπει να πάρουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης στη διεύθυνση επιστροφής της συνάρτησης και να τη βάλουμε στον ra. Έτσι μέσω του ra επιστρέφουμε στην καλούσα.

Επιπλέον ιδιαίτερο ενδιαφέρον στην παραπάνω εικόνα εμφανίζεται στις τετράδες 11 και 12 αφού ναι μεν είναι παράμετροι συνάρτησης αλλά όπως φαίνεται και στις τετράδες η μια έχει πέρασμα με τιμή και η άλλη έχει πέρασμα με τιμή επιστροφής συνάρτησης.

Έτσι έχουμε 2 διαφορετικές περιπτώσεις οπότε όπως βλέπουμε στην γραμμή 11 αρχικά τοποθετούμε τον `fr` να δείχνει στην στοίβα της συνάρτησης που δημιουργηθεί και ύστερα παράγουμε τις υπόλοιπες γραμμές τελικού κώδικα.

Ενώ στη γραμμή 12 γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή.

Τέλος μετά από ερώτηση που σας κάναμε και μας υποδείξετε ότι ο ένας έξυπνος και πιο ωραίος τρόπος να δείξουμε ότι ο τελικός κώδικας μας λειτουργεί σωστά είναι να τρέξουμε τον κώδικα αυτό στον προσομοιωτή RARS. Έτσι τρέξαμε το τελικό κώδικα αυτόν με είσοδο 2 και έδωσε έξοδο 1. Που είναι και το επιθυμητό αποτέλεσμα. Αφού για είσοδο 2 το fibonacci είναι το 1.

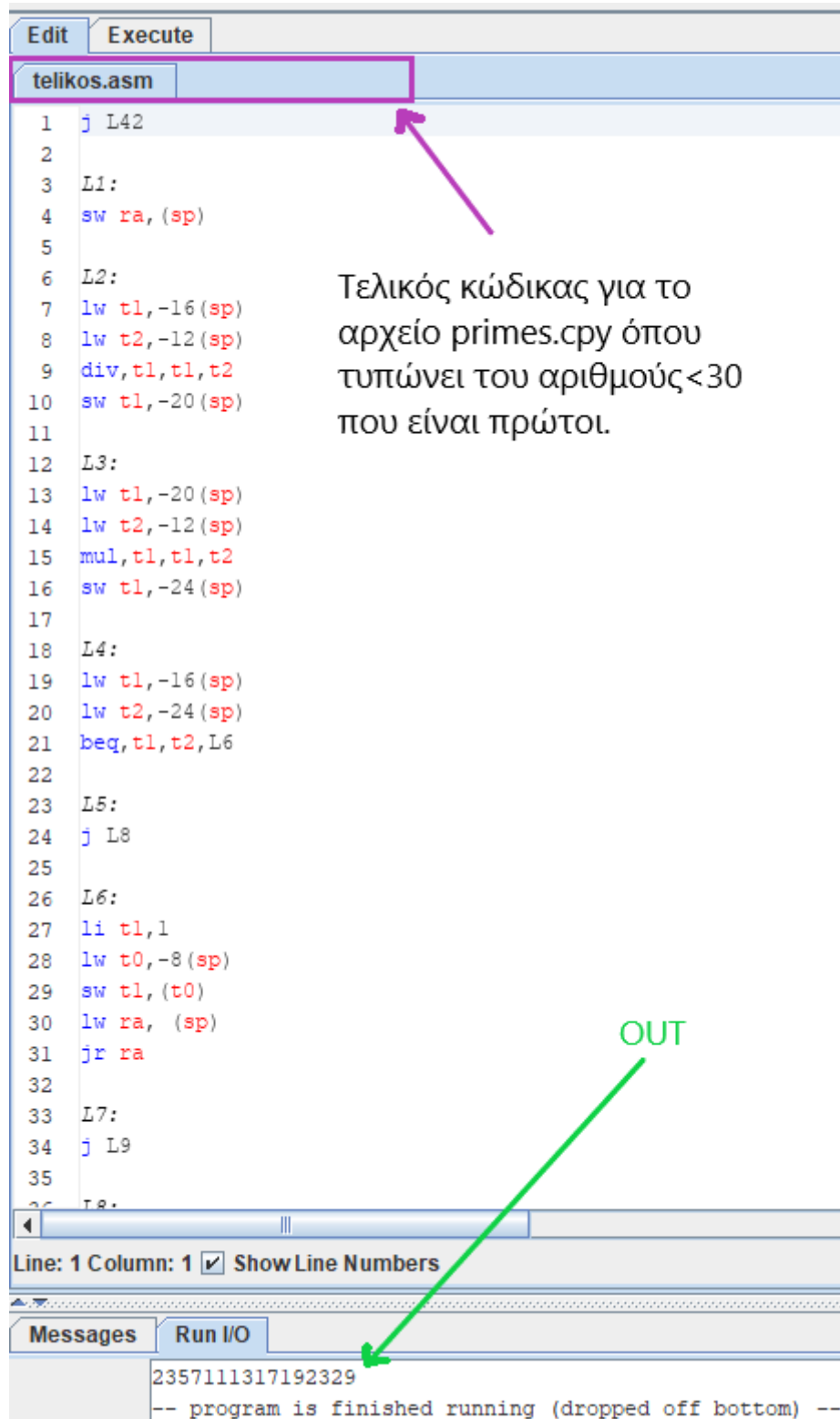
```
telikos.asm
1  j L24
2
3  L1:
4  sw ra, (sp)
5
6  L2:
7  lw t1, -12(sp)
8  li t2, 1
9  ble, t1, t2, L4
10
11  L3:
12  j L6
13
14  L4:
15  lw t1, -12(sp)
16  lw t0, -8(sp)
17  sw t1, (t0)
18  lw ra, (sp)
19  jr ra
20
21  L5:
22  j L16
23
24  L6:
25  lw t1, -12(sp)
26  li t2, 1
27  sub, t1, t1, t2
28  sw t1, -16(sp)
29
30  L7:
31  addi fp, sp, 36
32  lw t0, -16(sp)
33  sw t0, -12(fp)
34
35  L8:
36  addi t0, sp, -20
```

Line: 1 Column: 1 ☒ Show Line Numbers

Messages Run I/O

2  
1  
-- program is finished running (dropped off bottom) --

Άλλος ένας τελικός κώδικας που δοκιμάσαμε στον προσομοιωτή RARS είναι αυτός του αρχείου `primes.cry` που όπως φαίνεται και στην εικόνα τυπώνει τους αριθμούς που είναι πρώτοι και είναι μικρότερη το 30.



```
1 j L42
2
3 L1:
4 sw ra, (sp)
5
6 L2:
7 lw t1, -16(sp)
8 lw t2, -12(sp)
9 div, t1, t1, t2
10 sw t1, -20(sp)
11
12 L3:
13 lw t1, -20(sp)
14 lw t2, -12(sp)
15 mul, t1, t1, t2
16 sw t1, -24(sp)
17
18 L4:
19 lw t1, -16(sp)
20 lw t2, -24(sp)
21 beq, t1, t2, L6
22
23 L5:
24 j L8
25
26 L6:
27 li t1, 1
28 lw t0, -8(sp)
29 sw t1, (t0)
30 lw ra, (sp)
31 jr ra
32
33 L7:
34 j L9
35
36 L8:
```

Line: 1 Column: 1 ☒ Show Line Numbers

Messages Run I/O

2357111317192329  
-- program is finished running (dropped off bottom) --

OUT

# Σύνοψη.

---

Έτσι αυτή η εργασία έφτασε στο τέλος της και εδώ θα σας παρουσιάσουμε πώς μπορεί κανείς να τρέξει την άσκηση και τα αρχεία εξόδου που θα δημιουργηθούν.

Εκτέλεση του προγράμματος με την εντολή:

**python cutePY\_4426\_4573.py file.cpy**

Αφού δώσουμε την παραπάνω εντολή θα δημιουργηθούν 3 αρχεία τα οποία είναι:

Αρχείο ενδιάμεσου κώδικα : **Endiamesos.int**

Αρχείο πίνακα συμβόλων : **Pinakas.symb**

Αρχείο τελικού κώδικα : **telikos.asm**

Ευχαριστούμε για το χρόνο που αφιερώσατε να διαβάσετε την παραπάνω αναφορά.