Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών &
Πληροφορικής Πολυτεχνική Σχολή
Πανεπιστήμιο Ιωαννίνων

Αναφορά Προγραμματιστικής Εργασίας Εξαμήνου

Ομάδα:

Σολδάτου Χριστίνα Ολυμπία

Αικατερίνη Τσιτσιμίκλη

Μεταφραστές

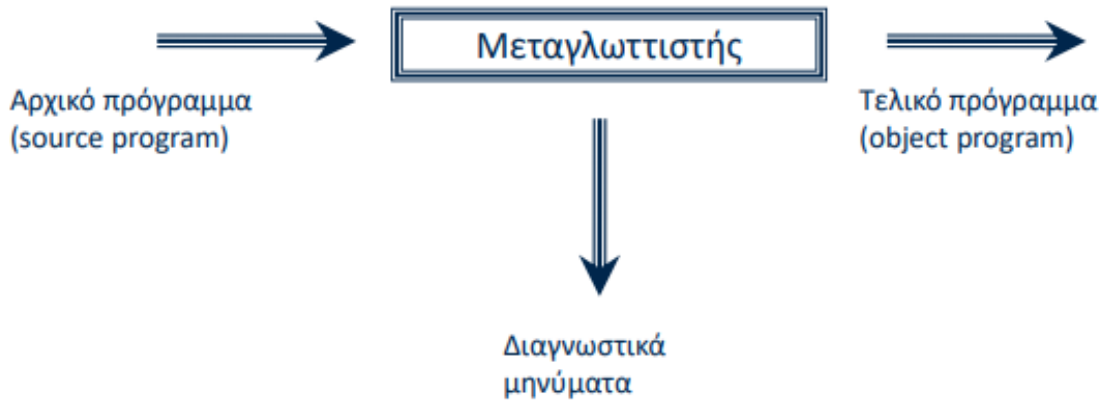
Διδάσκων: Γεώργιος Μανής

Εαρινό Εξάμηνο 2023

Στοιχεία Ομάδας:

Σολδάτου Χριστίνα Ολυμπία, Α.Μ. 4001, username: cs04001
Αικατερίνη Τσιτσιμίκλη, Α.Μ. 4821, username: cs04821

Περιεχόμενα



ΕΙΣΑΓΩΓΗ

Η παρακάτω αναφορά αναλύει την υλοποίηση ενός μεταγλωττιστή για την γλώσσα προγραμματισμού **CutePy**. Η CutePy είναι μια μικρή, εκπαιδευτική γλώσσα προγραμματισμού που θυμίζει τη γλώσσα Python, αλλά είναι πολύ μικρότερη από άποψη προγραμματιστικών δυνατοτήτων και υποστηριζόμενων δομών.

Λεκτικές μονάδες:

Το αλφάβητο της CutePy αποτελείται από:

- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (`A,…,Z` και `a,…,z`),
- τα αριθμητικά ψηφία (`0,…,9`),
- την κάτω παύλα (_),
- τα σύμβολα των αριθμητικών πράξεων (`+, -, *, //`),
- τους τελεστές συσχέτισης (`<, >, !=, <=, >=, ==`)
- το σύμβολο ανάθεσης (`=`)
- τους διαχωριστές (`;, “,”, :`)
- τα σύμβολα ομαδοποίησης (`[, ], (, ) , #{ , #}`)
- και διαχωρισμού σχολίων (`#$`)

Τα σύμβολα `[ , ]` χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα `( , )` στις αριθμητικές παραστάσεις.

Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα, ψηφία και κάτω παύλες, αρχίζοντας όμως από γράμμα. Ο μεταγλωττιστής λαμβάνει υπόψη του μόνο τα τριάντα πρώτα γράμματα. Αναγνωριστικά με περισσότερους από 30 χαρακτήρες θεωρούνται λανθασμένα.

Οι λευκοί χαρακτήρες (tab, space, return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια, να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, αριθμητικές σταθερές.

Το ίδιο ισχύει και για τα σχόλια, τα οποία πρέπει να βρίσκονται μέσα σε σύμβολα **#$**.

Μορφή προγράμματος:

Κάθε πρόγραμμα αποτελείται από δύο τμήματα. Στο πρώτο τμήμα δηλώνονται οι κύριες συναρτήσεις. Πρόκειται για συναρτήσεις χωρίς παραμέτρους οι οποίες δεν επιστρέφουν αποτέλεσμα. Ξεκινούν με το token **main__**. Δηλαδή, νόμιμα ονόματα για μία κύρια συνάρτηση μπορεί να είναι τα ακόλουθα: main_function, main_finonacci, main_foo.

Η δομή μίας κύριας συνάρτησης

```
def main_function()
#{
    declarations
    local_functions
    statements
#}
```

Κάθε κύρια *συνάρτηση* μπορεί να περιέχει μία ή περισσότερες φωλιασμένες *τοπικές συναρτήσεις*. Επιτρέπεται φωλιασμός τοπικών συναρτήσεων, δεν επιτρέπεται φωλιασμός ανάμεσα σε κύριες συναρτήσεις. Επίσης, μία κύρια συνάρτηση δεν μπορεί να καλέσει μία κύρια συνάρτηση.

Κάθε κύρια συνάρτηση μπορεί να δηλώσει μέσα της και να καλέσει τοπικές συναρτήσεις, σύμφωνα με τους κανόνες εμβέλειας. Έτσι, κάθε **function** στην παραπάνω περιγραφή μέσα σε μία κύρια συνάρτηση μπορεί να περιέχει μία ή περισσότερες τοπικές συναρτήσεις της μορφής:

```
local_functions  --> ( local_function )*
```

δηλαδή:

```
def local_function(formal_parameters)
#{
    declarations
    local_functions
    statements
#}
```

Τύποι και δηλώσεις μεταβλητών:

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η CutePy είναι οι *ακέραιοι αριθμοί.* Η δήλωση γίνεται με την εντολή **#declare**. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Επιτρέπεται να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της **#declare**.

Τελεστές και εκφράσεις:

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

- Πολλαπλασιαστικοί: `*, //`
- Προσθετικοί: `+, -`
- Σχεσιακοί: `==, <, >, !=, <=, >=`
- Λογικό `not`
- Λογικό `and`
- Λογικό `or`

Δομές της γλώσσας CutePy:

Οι **δομές** της γλώσσας χωρίζονται σε δύο κατηγορίες: τις *απλές εντολές* και τις *δομημένες*. Οι απλές εντολές τελειώνουν με ένα ελληνικό ερωτηματικό. Στις δομημένες δεν απαιτείται να τελειώνουν με ελληνικό ερωτηματικό. Απλές εντολές θεωρούμε τις εντολές εκχώρησης, επιστροφής τιμής συνάρτηση, εισόδου και εξόδου δεδομένων. Δομημένες εντολές στην CutePy θεωρούμε τις εντολές απόφασης και επανάληψης.

- **Εκχώρηση**:
  Χρησιμοποιείται για την ανάθεση τιμής σε μια μεταβλητής.
  Όπου expression είναι μία μαθηματική έκφραση.

```
var_name = expression
```

- **Απόφαση if:**
  Η εντολή απόφασης if εκτιμά εάν ισχύει η συνθήκη *condition* και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές $statements_1$ που το ακολουθούν. Το else δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται μέσα σε αγκύλες. Οι εντολές $statements_2$ που ακολουθούν το else εκτελούνται εάν η συνθήκη *condition* δεν ισχύει.

Σύνταξη:

```
if (condition):
    statements1
[ else:
    statements2 ]
```

  Τα statements μπορεί να είναι μία απλή ή μία δομημένη εντολή. Μπορεί επίσης να είναι μία σειρά από τέτοιες εντολές, οι οποίες είναι κλεισμένες ανάμεσα στα σύμβολα **#{** και **#}**.

- **Επανάληψη while:**
  Η εντολή επανάληψης while επαναλαμβάνει τις εντολές *statements*, όσο η συνθήκη *condition* ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η *condition*, το αποτέλεσμα της αποτίμησης είναι ψευδές, τότε οι *statements* δεν εκτελούνται ποτέ.

Σύνταξη:

```
while (condition):
    statements
```

  Το *statements* είναι το ίδιο με αυτό που περιγράφηκε στη δομή if.
  Μπορεί να είναι μία εντολή ή μία σειρά από εντολές, οι οποίες στη δεύτερη περίπτωση,είναι κλεισμένες ανάμεσα στα σύμβολα **#{** και **#}** .

❖ **Επιστροφή τιμής συνάρτησης:**
Χρησιμοποιείται μέσα σε τοπικές συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης, το οποίο είναι το αποτέλεσμα της αποτίμησης του *expression*.

```
return (expression)
```

❖ **Έξοδος δεδομένων:**
Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του *expression*.

```
print (expression)
```

❖ **Είσοδος δεδομένων:**
Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο. Η τιμή που θα δώσει θα μεταφερθεί στην μεταβλητή var, μέσω της *εκχώρησης*.

```
var = int(input())
```

❖ **Συναρτήσεις**:
Η CutePy υποστηρίζει κύριες και τοπικές συναρτήσεις. Η σύνταξη της δήλωσης των τοπικών συναρτήσεων είναι:
Η επιστροφή της τιμής μιας τοπικής συνάρτησης γίνεται με την εντολή return.
.

```
def function(formal_parameters)
#{
    declarations
    functions
    statements
#}
```

Μετάδοση παραμέτρων:

Η CutePy υποστηρίζει μετάδοση παραμέτρων ***με τιμή***, όπως η Python. Πιθανές αλλαγές στην τιμή της κατά την εκτέλεση της τοπικής συνάρτηση δεν επιστρέφονται στη συνάρτηση που την κάλεσε.

Κανόνες εμβέλειας:

Έχουμε τις καθολικές και τις τοπικές μεταβλητές. Οι καθολικές μεταβλητές δηλώνονται στο κυρίως πρόγραμμα και είναι προσβάσιμες σε ολόκληρο το πρόγραμμα. Οι τοπικές μεταβλητές δηλώνονται σε συναρτήσεις είναι προσβάσιμες μόνο μέσα στις συγκεκριμένες *συναρτήσεις*.

Κλήση κυρίων συναρτήσεων:

Ο τρόπος δήλωσης

```
if __name__ == "__main__":
```

Οι κύριες συναρτήσεις καλούνται μόνο από το κυρίως πρόγραμμα. Δεν επιστρέφουν τιμή, δεν δέχονται παραμέτρους, δεν καλούν άλλες κύριες συναρτήσεις, παρά μόνο εκτελούν τον δικό τους κώδικα και καλούν τις δικές τους απλές συναρτήσεις. Το κυρίως πρόγραμμα δηλώνεται μετά το τέλος των κυρίων συναρτήσεων.

ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

Η λεκτική ανάλυση αποτελεί την πρώτη φάση της μεταγλώττισης. Κατά τη φάση αυτή, διαβάζεται το *πηγαίο πρόγραμμα* και παράγονται οι λεκτικές μονάδες. Χρησιμοποιούμε τον όρο **λεκτική μονάδα** για να αναπαραστήσουμε οτιδήποτε έχει νόημα να θεωρηθεί ως αυτόνομο σύνολο συνεχόμενων χαρακτήρων που μπορεί να συναντηθεί σε ένα πρόγραμμα και βρίσκει σημασιολογία στην υπό υλοποίηση γλώσσα.

Ο λεκτικός αναλυτής

Ο λεκτικός αναλυτής υλοποιείται σαν μία συνάρτηση, που διαβάζει έναν-έναν τους χαρακτήρες από το αρχείο εισόδου και ελέγχει εάν ο εκάστοτε χαρακτήρας ανήκει σε μία από τις παρακάτω κατηγορίες. Στη συνέχεια επιστρέφει την επόμενη λεκτική μονάδα και εάν αναγνωρίσει κάποιο σφάλμα, τότε αυτόματα πρέπει να ενημερώσει τον συντακτικό αναλυτή ή να τερματίσει την μεταγλώττιση.

- identifier: ονόματα μεταβλητών, συναρτήσεων, σταθερών και ό,τι άλλο μπορεί να έχει ονομασία σε ένα πρόγραμμα. Στη φάση αυτή είναι πολύ νωρίς για να μπορέσουμε να διαχωρίσουμε αν η συμβολοσειρά που αναγνωρίσαμε αποτελεί όνομα μεταβλητής, συνάρτησης ή σταθεράς, ακόμα κι αν είναι νόμιμη (π.χ. δηλωμένη μεταβλητή ή όχι) και για τον λόγο αυτό δημιουργούμε μία γενική κατηγορία, κάτω από την οποία τοποθετούμε όλες αυτές τις περιπτώσεις.

- number: αριθμητικές σταθερές

- keyword: περιέχει τις λέξεις κλειδιά της γλώσσας, π.χ.: while, if, else

- addOperator: προσθετικοί αριθμητικοί τελεστές: +, -

- mulOperator: πολλαπλασιαστικοί αριθμητικοί τελεστές: * , /

- relOperator: λογικοί τελεστές π.χ.: ==, >=, <, <>

- assignment: τελεστής εκχώρησης (=)

- delimiter: διαχωριστές π.χ.: ,, ., ;

- groupSymbol: σύμβολα ομαδοποίησης π.χ.: (,), {, }, [,]

Για κάθε μία από τις παραπάνω κατηγορίες, ορίζουμε μεταβλητές που αναπαριστούν τα περιεχόμενα τους στο πρόγραμμά μας.

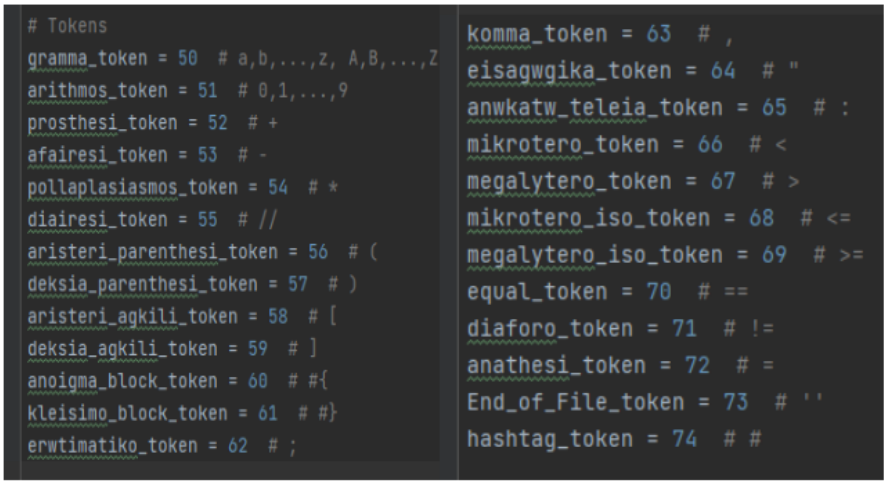
```
# XARAKTHRES
gramma = 0  # a,b,...,z, A,B,...,Z
arithmos = 1  # 0,1,...,9
prosthesi = 2  # +
afairesi = 3  # -
pollaplasiasmos = 4  # *
diairesi = 5  # //
aristeri_parenthesi = 6  # (
deksia_parenthesi = 7  # )
aristeri_agkili = 8  # [
deksia_agkili = 9  # ]
aristero_agkistro = 10  # {
deksi_agkistro = 11  # }
erwtimatiko = 12  # ;
komma = 13  # ,
eisagwgika = 14  # "
anw_katw_teleia = 15  # :
katw_paula = 16  # _
mikrotero = 17  # <
megalytero = 18  # >
ison = 19  # =
hashtag = 20  # #
dolario = 21  # $
tab = 22  # \t
keno = 23  # space
epomeni_grammi = 24  # return
End_of_File = 25  # EoF
mi_apodekto_symbolo = 26
thaumastiko = 27  # !
```

Εικόνα 1.

Ένα ακόμα πεδίο του αντικειμένου που αναπαριστά μια λεκτική μονάδα είναι ο αριθμός γραμμής στην οποία αναγνωρίστηκε. Η πληροφορία αυτή είναι χρήσιμη στον συντακτικό αναλυτή, έτσι ώστε να μπορεί να επιστρέψει εύστοχα μηνύματα σφάλματος, αν εντοπίσει κάποιο σφάλμα κατά τη συντακτική ανάλυση. Αν αυτή η πληροφορία δεν διατηρηθεί μέσα στο αντικείμενο που επιστρέφεται στον συντακτικό αναλυτή, τότε θα χαθεί, αφού ο συντακτικός αναλυτής δεν χρησιμοποιεί άλλη πληροφορία, πέρα από αυτήν που του επιστρέφεται από τον συντακτικό αναλυτή. Ας το ονομάσουμε **grammi**.

```
grammi = 1
```

Έτσι, μια λεκτική μονάδα μπορεί να αναπαρασταθεί με ένα token. Ένα token περιέχει την πληροφορία που μεταφέρει ο λεκτικός αναλυτής στον συντακτικό αναλυτή. Για να δούμε πώς ο λεκτικός αναλυτής δημιουργεί και επιστρέφει στον συντακτικό αναλυτή τα token, θα μελετήσουμε αναλυτικά την εσωτερική του λειτουργία



```
# Tokens
gramma_token = 50  # a,b,...,z, A,B,...,Z
arithmos_token = 51  # 0,1,...,9
prosthesi_token = 52  # +
afairesi_token = 53  # -
pollaplasiasmos_token = 54  # *
diairesi_token = 55  # //
aristeri_parenthesi_token = 56  # (
deksia_parenthesi_token = 57  # )
aristeri_agkili_token = 58  # [
deksia_agkili_token = 59  # ]
anoigma_block_token = 60  # #{
kleisimo_block_token = 61  # #}
erwtimatiko_token = 62  # ;
komma_token = 63  # ,
eisagwgika_token = 64  # "
anwkatw_teleia_token = 65  # :
mikrotero_token = 66  # <
megalytero_token = 67  # >
mikrotero_iso_token = 68  # <=
megalytero_iso_token = 69  # >=
equal_token = 70  # ==
diaforo_token = 71  # !=
anathesi_token = 72  # =
End_of_File_token = 73  # ''
hashtag_token = 74  # #
```

Εικόνα 2.

```
# Tokens Desmeumenwn Leksewn
def_token = 500  # function
declare_token = 501  # declare
if_token = 502  # if
else_token = 503  # else
while_token = 504  # while
return_token = 505  # return
and_token = 506  # and
or_token = 507  # or
not_token = 508  # not
input_token = 509  # input
print_token = 510  # print
int_token = 511  # int
name_token = 512  # name
main_token = 513  # main
```

Εικόνα 3.

Εσωτερική λειτουργία του λεκτικού αναλυτή:

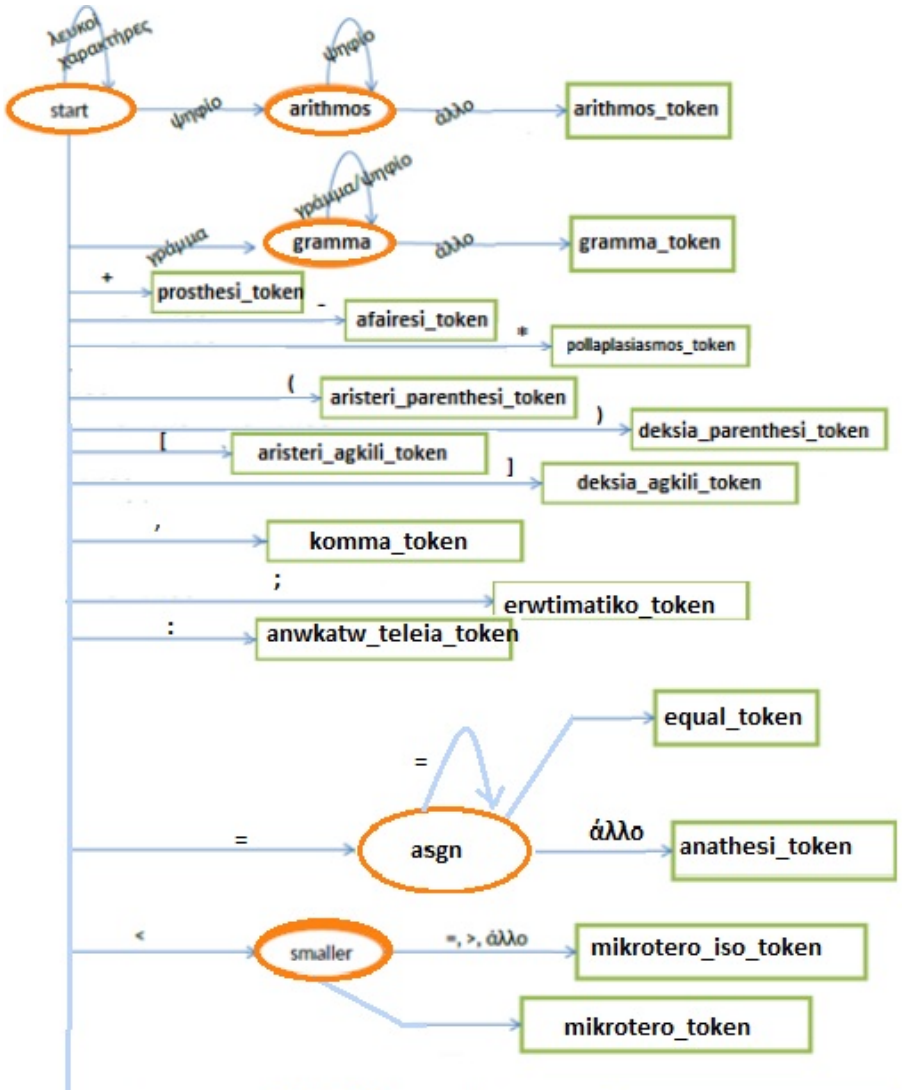
Ο λεκτικός αναλυτής λειτουργεί ως εξής:

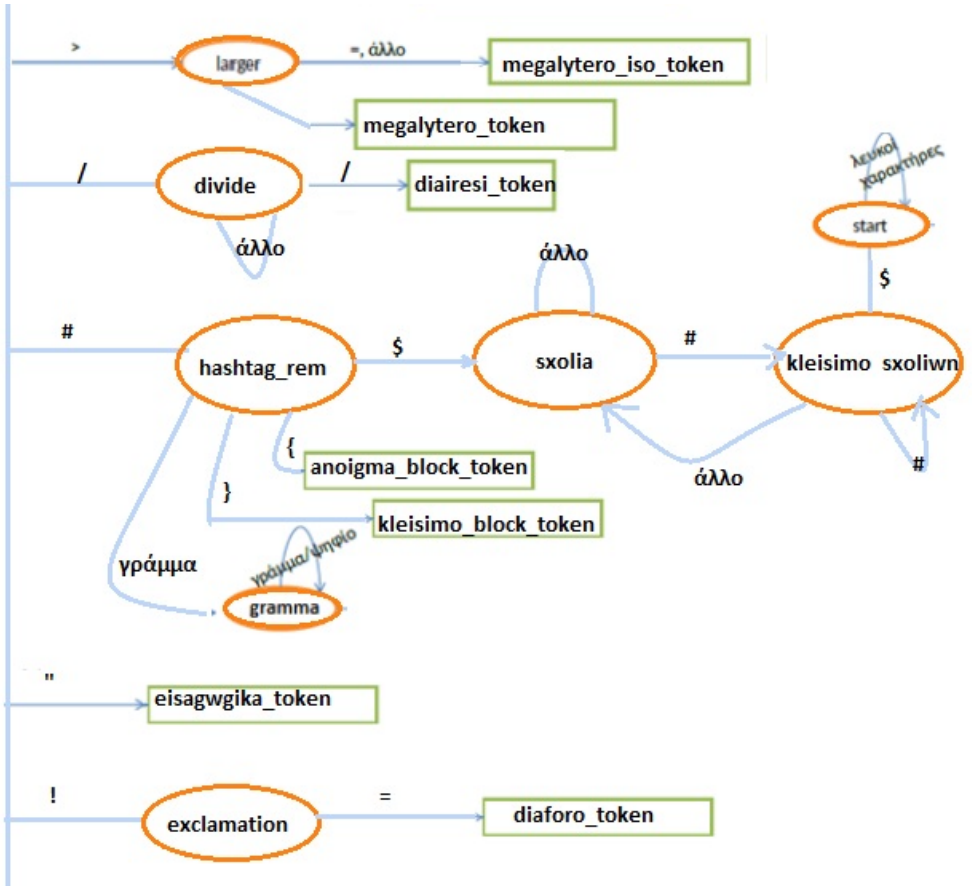
```
# KATASTASEIS
start = 0  # start
dig = 1  # arithmoi
idk = 2  # grammata/psifia
asgn = 3  # anathesh ison =
smaller = 4  # mikrotero apo
larger = 5  # megalytero apo
hashtag_rem = 6  # hashtag
sxolia = 7  # anoigoun sxolia #$
kleisimo_sxoliwn = 8  # kleinoun sxolia #$
divide = 9  # diairesi //
exclamation = 10  # thavmastiko
```

Ξεκινάμε πάντα από την κατάσταση **start** και ανάλογα τον χαρακτήρα που εμφανίζεται στην είσοδο θα μεταβεί είτε σε επόμενη κατάσταση είτε σε κάποια από τις παραπάνω κατηγορίες. Στο σχήμα η αρχική κατάσταση είναι η κατάσταση start, οι μη τελικές καταστάσεις συμβολίζονται με έλλειψη, ενώ οι τελικές με παραλληλόγραμμο.

Για παράδειγμα, εάν έχουμε να αναγνωρίσουμε την λεκτική μονάδα “a_2 “ τότε σύμφωνα με το παρακάτω σχήμα θα περάσουμε αρχικά στην κατάσταση **gramma** επειδή διαβάσαμε το *γράμμα* a, στην συνέχεια θα διαβάσουμε τον *αριθμό* 2 και *θα παραμείνουμε* στη **gramma** έως ότου διαβάσουμε το *κενό* και καταλήξουμε τελικά στην κατηγορία **gramma_token**.

Σχήμα λεκτικού αναλυτή

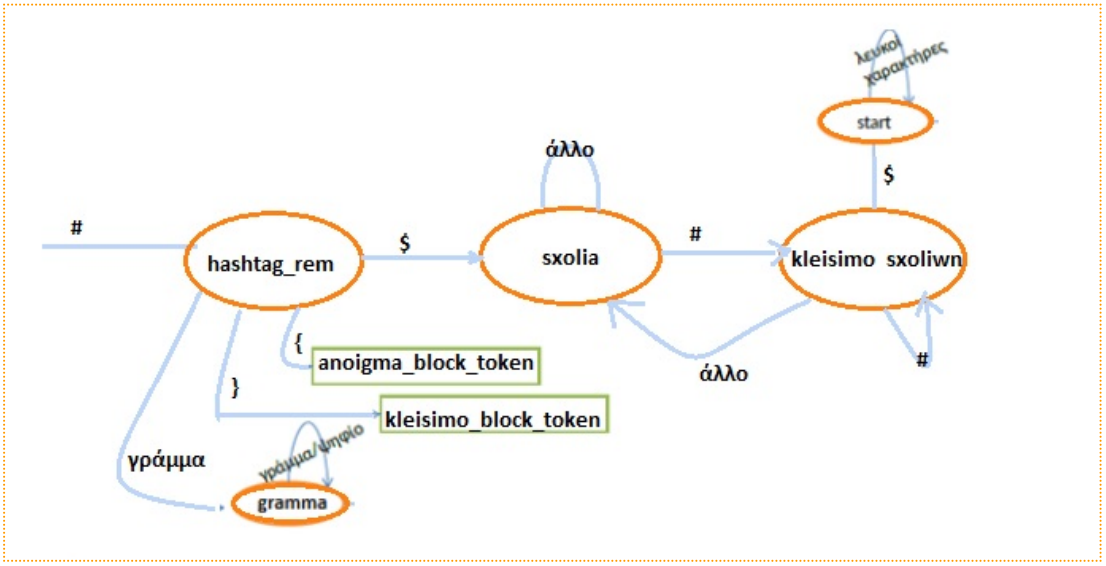




Διαχείριση σχολίων:

Ο σχολιασμός του κώδικα πρέπει να είναι αναπόσπαστο τμήμα της συγγραφής του.

Τα σχόλια τα διαχειρίζεται αποκλειστικά ο λεκτικός αναλυτής, τα αναγνωρίζει και τα *αφαιρεί από το πρόγραμμα.* Ο συντακτικός αναλυτής δεν χρειάζεται καν να πληροφορηθεί την ύπαρξή τους. Στην CutePy τα σχόλια αρχίζουν και τελειώνουν με τον χαρακτήρα #$. Όταν βρισκόμαστε στην αρχική κατάσταση και εμφανιστεί ο χαρακτήρας # τότε μεταβαίνουμε στην κατάσταση **hashtag_rem**. Εκεί παραμένουμε όσο στην είσοδο δεν εμφανίζεται πάλι ο χαρακτήρας #.



Πρέπει σε αυτό το σημείο να τονίσουμε την σημαντικότητα της κατάστασης αυτής. Πιο συγκεκριμένα, εάν έρθει στο αυτόματο:

- **{** : τότε ανοίγει ένα block κώδικα καθώς θα έχουμε **#{**
- **}** : τότε κλείνει ένα block κώδικα καθώς θα έχουμε **#}**
- **γράμμα** : τότε μεταβαίνει στην κατάσταση gramma καθώς θα μπορούσε να είναι δεσμευμένη λέξη όπως για παράδειγμα **#declare**.

Εάν μετά το # έρθει **$** τότε μπαίνουμε στην κατάσταση των σχολίων. Σε αυτή την κατάσταση μπορεί να έρθει οτιδήποτε. Ο συντακτικός αναλυτής αγνοεί τελείως ότι υπάρχει ανάμεσα στα #$. Τέλος, αναμένεται να εμφανιστεί ξανά το #$. Όταν δούμε ξανά το # μεταβαίνουμε στην κατάσταση **kleisimo_sxoliwn**. Μόλις έρθει $ τότε βγαίνουμε από την κατάσταση και μεταβαίνουμε στην **start**.

Περίπτωση σφάλματος:

```
# Arxikopoihsh ERRORS
Error_Arithmos_Gramma = -1
Error_Ektos_Oriwn = -2
Error_Mono_Ena_Hashtag = -3
Error_Eof_Anoigma_Comment = -4
Error_Mono_Aristero_Agkistro = -5
Error_Mono_Deksi_Agkistro = -6
Error_Mono_Ena_Slash = -7
Error_Mono_Ena_Dollar = -8
Error_Plus_30_Xaraktires = -9
Error_Katw_Paula_Prin_Identifier = -10
Error_Lathos_Symbolo = -11
Error_Mono_Ena_Thaumastiko = -12
```

Στη φάση της λεκτικής ανάλυσης αναγνωρίζονται τα πρώτα από τα σφάλματα που μπορεί να ανακαλύψει ένας μεταγλωττιστής. Φυσικά, πρόκειται για σφάλματα τα οποία σχετίζονται με λεκτικές μονάδες. Σφάλματα που σχετίζονται με σύνταξη ή σημασιολογικά θα εντοπιστούν σε φάσεις που θα ακολουθήσουν.

Τα σφάλματα αυτής της φάσης προκύπτουν από συνθήκες που μπορεί να εμφανιστούν κατά τη διάσχιση του αυτόματου. Θα χρησιμοποιήσουμε το αυτόματο για την CutePy για να εντοπίσουμε τέτοιες περιπτώσεις και θα δείξουμε και τα μηνύματα λάθους που δομούνται στον συντακτικό αναλυτή.

Τα σφάλματα που συναντάμε είναι τα ακόλουθα:

1. Σύμφωνα με την περιγραφή της CutePy, υπάρχουν δύο περιορισμοί, οι οποίοι θα πρέπει να ελεγχθούν εάν το μήκος ενός identifier είναι το πολύ 30 χαρακτήρες και εάν μία ακέραια αριθμητική σταθερά βρίσκεται μέσα στο επιτρεπτό εύρος τιμών. Έτσι, θα γίνουν οι απαραίτητοι έλεγχοι. Στην περίπτωση που οι έλεγχοι είναι επιτυχημένοι, τότε θα έχουμε επιτυχή αναγνώριση και επιστροφή του αποτελέσματος στον συντακτικό αναλυτή. Εάν ο έλεγχος δεν είναι επιτυχημένος, τότε ο λεκτικός αναλυτής θα οδηγείται σε κατάσταση σφάλματος.

```
elif (katastasi == Error_Plus_30_Xaraktires):
    print('ERROR: Emfanistikan panw apo 30 xaraktires sti leksi !!!')
```

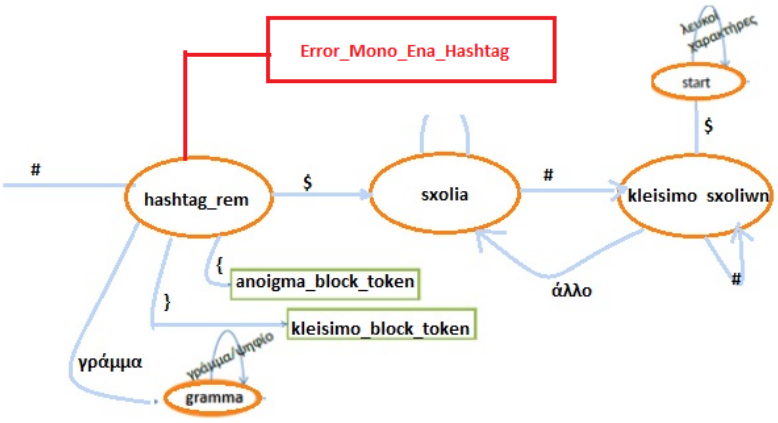
2. Στη λεκτική ανάλυση θα κατατάσσαμε και την αναγνώριση λεκτικών μονάδων που, ενώ ξεκινούν από ψηφίο, μέσα στη λεκτική μονάδα εμφανίζονται γράμματα. Αυτό είναι κάτι που θα μπορούσε να αναγνωριστεί και να οδηγήσει σε σφάλμα και κατά τη διάρκεια της συντακτικής ανάλυσης. Είναι φανερό, όμως, ότι σαν σφάλμα είναι λεκτικό και πρέπει να ενταχθεί εδώ.

```
if (katastasi == Error_Arithmos_Gramma):
    print('ERROR: Exw arxika arithmo kai meta emfanizetai gramma !!!')
```

3. Ενώ βρισκόμαστε στην κατάσταση **sxolia** έχουν ανοίξει δηλαδή σχόλια, εμφανίζεται τέλος του αρχείου εισόδου. Έχουμε δηλαδή σχόλια τα οποία ανοίγουν, αλλά δεν κλείνουν ποτέ.

```
elif (katastasi == Error_Eof_Anoigma_Comment):
    print('ERROR: Exw telos arxeiou (EOF) kai ta sxolia enw exoun anoiksei, den exoun kleisei !!!')
```

4. Ενώ βρισκόμαστε στην κατάσταση **hashtag_rem** ενώ έχει ανοίξει ένα # μετά δεν ακολουθεί κάτι όπως γράμμα ή { ή } ή $.




```
elif (katastasi == Error_Mono_Ena_Hashtag):
    print('ERROR: Emfanistike mono ena hashtag (#) xwris na akolouthei kati !!!')
```

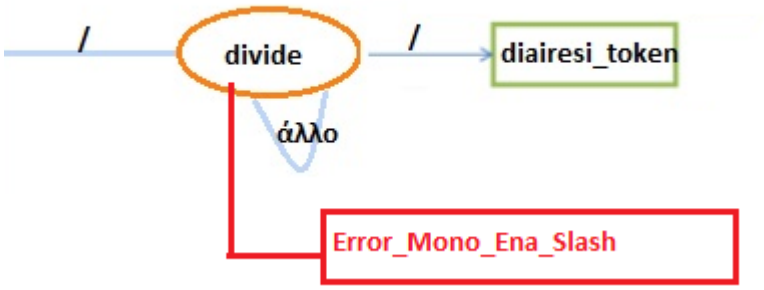
5. Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Οι ακέραιες σταθερές πρέπει να έχουν τιμές από –(232 − 1) έως 2 32 − 1. Εάν ο αριθμός βρίσκεται εκτός ορίων εμφανίζεται αντίστοιχο μήνυμα.

```
elif (katastasi == Error_Ektos_Oriwn):
    print('ERROR: O arithmos vrisketai ektos oriwn !!!')
```

6. Όταν έχουμε αριστερό/ δεξί άγκιστρο { αλλά χωρίς hashtag.

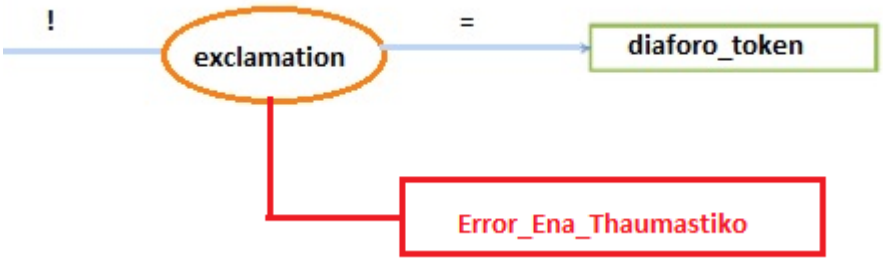
```
elif (katastasi == Error_Mono_Aristero_Agkistro):
    print('ERROR: Emfanistike mono to aristero agkistro ({) xwris hashtag !!!')
elif (katastasi == Error_Mono_Deksi_Agkistro):
    print('ERROR: Emfanistike mono to deksi agkistro (}) xwris hashtag !!!')
```

7. Στην κατάσταση divide υπάρχει μόνο ένα slash.




```
elif (katastasi == Error_Mono_Ena_Slash):
    print('ERROR: Emfanistike mono ena slash (/) !!!')
```

8. Στην κατάσταση exclamation υπάρχει μόνο ένα θαυμαστικό και δεν έχει έρθει ίσον..




```
elif (katastasi == Error_Mono_Ena_Thaumastiko):
    print('ERROR: Emfanistike mono ena thaumastiko (!) xwris ison !!!')
```

9. Τέλος, έχουμε τα παρακάτω λάθη:

```
elif (katastasi == Error_Katw_Paula_Prin_Identifier):
    print('ERROR: Den epitrepetai mia leksi na ksekina me katw paula (_) !!!')
elif (katastasi == Error_Lathos_Symbolo):
    print('ERROR: Emfanistike mh apodekto symbolo !!!')
```

```
elif (katastasi == Error_Mono_Ena_Dollar):
    print('ERROR: Emfanistike mono ena dollario ($) xwris hastag !!!')
```

ΥΛΟΠΟΙΗΣΗ ΜΕ ΠΙΝΑΚΑ:

Για την υλοποίηση της λειτουργίας του λεκτικού αναλυτή δημιουργούμε τον **πίνακα μεταβάσεων**. Πρόκειται για έναν πίνακα, μέσω του οποίου μεταβαίνουμε από κατάσταση σε κατάσταση και τελικά μας βοηθά να καταλήξουμε στην εύρεση των **token**, δηλαδή σε ποια κατηγορία ανήκει κάποιος χαρακτήρας.

Αρχικά, αρχικοποιούμε κάθε κατάσταση - χαρακτήρα σε αριθμούς – κωδικούς, ξεκινώντας την αρίθμηση από το μηδέν διότι αναφέρονται στις γραμμές και στις στήλες αντίστοιχα, του **πίνακα μεταβάσεων**. Επίσης, προκειμένου να γνωρίζουμε σε ποια από τις κατηγορίες ανήκει κάποιος χαρακτήρας, αρχικοποιούμε τα διάφορα tokens σε τυχαίους αριθμούς – κωδικούς, καθώς επίσης και τα tokens των δεσμευμένων λέξεων. Τέλος αρχικοποιούμε κάποιες μεταβλητές, προκειμένου να μπορούμε να διαχειριστούμε τα διάφορα errors - σφάλματα, που τυχόν υπάρχουν. Όπως αναφέραμε παραπάνω, ο συγκεκριμένος πίνακας περιέχει όσες γραμμές είναι οι καταστάσεις και όσες στήλες είναι οι χαρακτήρες. Σε κάθε θέση του, γράφουμε με βάση το παραπάνω σχήμα σε ποια κατάσταση θα μεταβούμε.

```
# Metavaseis
array = [
    # START - start
    [idk, dig, prosthesi_token, afairesi_token, pollaplasiasmos_token, divide,
     aristeri_parenthesi_token, deksia_parenthesi_token, aristeri_agkili_token, deksia_agkili_token,
     Error_Mono_Aristero_Agkistro, Error_Mono_Deksi_Agkistro, erwtimatiko_token, komma_token,
     eisagwgika_token, anwkatw_teleia_token, idk, smaller, larger, asgn, hashtag_rem, Error_Mono_Ena_Dollar,
     start, start, start, End_of_File_token, Error_Lathos_Symbolo, exclamation],

    # ARITHMOS - dig
    [Error_Arithmos_Gramma, dig, arithmos_token, arithmos_token, arithmos_token, arithmos_token,
     arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token,
     arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token,
     arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token, arithmos_token,
     Error_Lathos_Symbolo, arithmos_token],
```

```
# GRAMMA - idk
[idk, idk, gramma_token, gramma_token, gramma_token, gramma_token, gramma_token, gramma_token,
 gramma_token, gramma_token, gramma_token, gramma_token, gramma_token, gramma_token, gramma_token,
 gramma_token, idk, gramma_token, gramma_token, gramma_token, gramma_token,
 gramma_token, gramma_token, gramma_token, gramma_token, gramma_token, Error_Lathos_Symbolo, gramma_token],

# ANATHESI - asgn
[anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token,
 anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token,
 anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token,
 equal_token,
 anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token, anathesi_token,
 Error_Lathos_Symbolo, anathesi_token],

# MIKROTERO APO - smaller
[mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token,
 mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token,
 mikrotero_token,
 mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token,
 mikrotero_iso_token,
 mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token, mikrotero_token,
 Error_Lathos_Symbolo, mikrotero_token],
```

```
# MEGALYTERO APO - larger
[megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token,
 megalytero_token,
 megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token,
 megalytero_token,
 megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_iso_token,
 megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token, megalytero_token,
 Error_Lathos_Symbolo, megalytero_token],

# HASHTAG - hashtag_rem
[idk, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag,
 Error_Mono_Ena_Hashtag,
 Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag,
 anoigma_block_token, kleisimo_block_token,
 Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag,
 Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag,
 Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, sxolia, Error_Mono_Ena_Hashtag,
 Error_Mono_Ena_Hashtag,
 Error_Mono_Ena_Hashtag, Error_Mono_Ena_Hashtag, Error_Lathos_Symbolo, Error_Mono_Ena_Hashtag],
```

```
# SXOLIA
[sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia,
 sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia,
 sxolia, sxolia, sxolia, sxolia, kleisimo_sxoliwn, sxolia, sxolia,
 sxolia, sxolia, Error_Eof_Anoigma_Comment, sxolia, sxolia],

# KLEISIMO_SXOLIWN
[sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia,
 sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia, sxolia,
 sxolia, sxolia, sxolia, sxolia, kleisimo_sxoliwn, start, sxolia,
 sxolia, sxolia, Error_Eof_Anoigma_Comment, sxolia, sxolia],

# DIAIRESI - divide
[Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash,
 diairesi_token,
 Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash,
 Error_Mono_Ena_Slash,
 Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash,
 Error_Mono_Ena_Slash,
 Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Mono_Ena_Slash,
 Error_Mono_Ena_Slash,
 Error_Mono_Ena_Slash, Error_Mono_Ena_Slash, Error_Lathos_Symbolo, Error_Mono_Ena_Slash],
```

```
# THAVMASTIKO - exclamation
[Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko,
 diaforo_token,
 Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko, Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko,
 Error_Mono_Ena_Thaumastiko, Error_Lathos_Symbolo, Error_Mono_Ena_Thaumastiko]
```

Πιο συγκεκριμένα:

- Η **start** αποτελεί την πρώτη γραμμή του πίνακα, δηλαδή βρίσκομαι στην αρχική κατάσταση και για κάθε χαρακτήρα σκέφτομαι που θα μεταβώ.
- Η **dig** αποτελεί την δεύτερη γραμμή του πίνακα και όταν βρεθώ εδώ, σημαίνει ότι είμαι σε κατάσταση που υπάρχει ψηφίο και σκέφτομαι τι επιλογές έχω.
- Η **idk** αποτελεί την τρίτη γραμμή του πίνακα μεταβάσεων και όταν βρεθώ εδώ, σημαίνει ότι είμαι σε κατάσταση που υπάρχει γράμμα και σκέφτομαι τι επιλογές έχω.

Αξίζει να σημειωθεί πως για να συμπληρώσουμε τον πίνακα μεταβάσεων σκεφτήκαμε ως εξής: Κάθε γραμμή του πίνακα, την συμπληρώσαμε με 27 στοιχεία όσο και το πλήθος των χαρακτήρων που έχουμε ορίσει. Στη συνέχεια, για κάθε κατάσταση – γραμμή, παίρνουμε όλες τις πιθανές επιλογές που μπορούμε να έχουμε. Για παράδειγμα, αν βρισκόμαστε στην γραμμή 2, δηλαδή στην κατάσταση που έχουμε δει ψηφίο (**dig**), όσο συνεχίζουμε να διαβάζουμε ψηφίο παραμένουμε στην ίδια κατάσταση (**dig**). Αν διαβάσουμε οποιονδήποτε άλλο από τους χαρακτήρες που έχουμε ορίσει τότε μεταβαίνουμε σε κατάσταση token (**arithmos_token**). Βέβαια στην συγκεκριμένη κατάσταση αν διαβάσουμε πρώτα ένα ψηφίο και στη συνέχεια ένα γράμμα, τότε έχουμε σφάλμα ή ακόμα αν διαβάσουμε κάποιο σύμβολο το οποίο δεν έχουμε ορίσει (μη αποδεκτό σύμβολο).

ΓΡΑΜΜΑΤΙΚΕΣ ΧΩΡΙΣ ΣΥΜΦΡΑΖΟΜΕΝΑ

Γραμματική τύπου **LL(1)**:

- **L: Left to righ**t ---> σχετίζεται με τον τρόπο που σαρώνεται η συμβολοσειρά εισόδου, από τα αριστερά στα δεξιά.
- **L: Left most derivation** ---> η κατασκευή του συντακτικού δέντρου αντιστοιχεί στην αριστερότερη παραγωγή.
- **(1)**: one look ahead symbol ---> επιλέγουμε ανάμεσα σε κανόνες ποιος θα πρέπει να είναι ο επόμενος που θα πρέπει να εφαρμοσθεί, αρκεί να ξέρουμε το επόμενο τερματικό σύμβολο στην είσοδο.

ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

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 της Cimple βασίζεται πάνω στην γραμματική χωρίς συμφραζόμενα και περιγράφεται από ένα σύνολο κανόνων, όπου για κάθε κανόνα δημιουργούμε αντίστοιχη συνάρτηση που τον υλοποιεί. Επομένως, ο συντακτικός αναλυτής παίρνει σαν είσοδο ακολουθία από λεκτικές μονάδες και στην έξοδό του δίνει το συντακτικό δέντρο το οποίο αντιστοιχεί στο αρχικό πρόγραμμα. Δηλαδή, η λειτουργία του πέρα από την αναγνώριση σφαλμάτων είναι να δώσει το περιβάλλον πάνω στο οποίο θα βασιστεί η παραγωγή του ενδιάμεσου κώδικα. Παρακάτω στην εικόνα βλέπουμε τον συντακτικό αναλυτή κατά την διαδικασία της μεταγλώττισης.

Η γραμματική της CutePy:

Ακολουθεί η γραμματική της CutePy η οποία δίνει και την ακριβή περιγραφή της γλώσσας:

```
startRule
        :   def_main_part
            call_main_part
        ;

def_main_part
        :   ( def_main_function )+
        ;

def_main_function
        :   'def' ID '(' ')' ':'
            '#{'
                    declarations
                    ( def_function )*
                    statements
            '#}'
        ;
```

```
def_function
        :   'def' ID '(' id_list ')' ':'
            '#{'
                    declarations
                    ( def_function )*
                    statements
            '#}'
        ;

declarations
        :   ( declaration_line )*
        ;


declaration_line
        :   '#declare' id_list
        ;

statement
        :   simple_statement
        |   structured_statement
        ;

statements
        :   statement+
        ;

simple_statement
        :   assignment_stat
        |   print_stat
        |   return_stat
        ;

structured_statement
        :   if_stat
        |   while_stat
        ;


assignment_stat
        :   ID '='
            (   expression ';'
            |   'int' '(' 'input' '(' ')' ')' ';'
            )
        ;
```

```
print_stat
        :   'print' '(' expression ')' ';'
        ;

return_stat
        :   'return' '(' expression ')' ';'
        ;

if_stat
        :   'if' '(' condition ')' ':'
                (   statement
                |   '#{' statements '#}'
                )
            (   'else' ':'
                (   statement
                |   '#{' statements '#}'
                )
            )?
        ;

while_stat
        :   'while' '(' condition ')' ':'
                (   statement
                |   '#{' statements '#}'
                )
        ;

id_list
        :    ID ( ',' ID )*
        |
        ;


expression
        :   optional_sign term
            ( ADD_OP term )*
        ;

term
        :   factor
            ( MUL_OP factor )*
        ;

factor
        :   INTEGER
        |   '(' expression ')'
        |    ID idtail
        ;
```

```
idtail
        : '(' actual_par_list ')'
        |
        ;

actual_par_list
        :     expression (  ',' expression )*
        |
        ;

optional_sign
        :    ADD_OP
        |
        ;

condition
        :   bool_term  ( 'or' bool_term )*
        ;

bool_term
        :   bool_factor ( 'and' bool_factor )*
        ;

bool_factor
        :   'not' '[' condition ']'
        |   '[' condition ']'
        |   expression REL_OP expression
        ;

call_main_part
        :
        'if' '__name__' '==' '"__main__"' ':'
            (  main_function_call )+
        ;

main_function_call
        :   ID '(' ')' ';'
        ;
```

Αναλυτική Εξήγηση Κώδικα:

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

    def_main_part()
    call_main_part()
```

- Η συνάρτηση **startRule()** δημιουργεί ένα νέο εύρος, καλεί το def_main_part() και στη συνέχεια καλεί την call_main_part().

- Η **def_main_part()** είναι μια συνάρτηση που εκτελεί επαναλαμβανόμενα τη def_main_function() εφόσον ισχύει μια συγκεκριμένη συνθήκη.

```
def def_main_part():
    global eksodos

    def_main_function()
    while (eksodos[0] == def_token):
        def_main_function()
```

Ο βρόχος while που ακολουθεί
while (eksodos[0] == def_token):
υποδεικνύει ότι ο κώδικας μέσα στο βρόχο θα εκτελεστεί επανειλημμένα όσο το πρώτο στοιχείο της λίστας eksodos είναι ίσο με την τιμή του def_token. Εντός του βρόχου while, πραγματοποιείται μια άλλη κλήση στη def_main_function().

- Η συνάρτηση **def_main_function()** έχει ως σκοπό την ανάλυση και επεξεργασία μιας συγκεκριμένης δομής σύνταξης που αντιστοιχεί στην CutePy. Εκτελεί διάφορους ελέγχους, δημιουργεί καταχωρήσεις στον πίνακα συμβόλων, χειρίζεται πεδία και επεξεργάζεται δηλώσεις εντός της συνάρτησης. Καλούνται οι συναρτήσεις def_function() και statements().

- Η συνάρτηση **def_function()** είναι υπεύθυνη για το χειρισμό της ανάλυσης και της επεξεργασίας των ένθετων συναρτήσεων εντός της κύριας συνάρτησης. Επαληθεύει τη σύνταξη και τη δομή του ορισμού της συνάρτησης, διασφαλίζοντας ότι ακολουθεί τους καθορισμένους γραμματικούς κανόνες. Η συνάρτηση εξάγει το όνομα της συνάρτησης, ελέγχει τη λίστα παραμέτρων και δημιουργεί μια οντότητα για τη συνάρτηση στον πίνακα συμβόλων. Στη συνέχεια προχωρά στην ανάλυση των δηλώσεων και των δηλώσεων εντός του μπλοκ της συνάρτησης. Επιπλέον, χειρίζεται τη δημιουργία τετράδων για τα μπλοκ έναρξης και τέλους της συνάρτησης. Η συνάρτηση υποστηρίζει την ένθεση πολλαπλών συναρτήσεων η μία μέσα στην άλλη καλώντας τον εαυτό της αναδρομικά όταν συναντάμε ορισμούς ένθετων συναρτήσεων. Συνολικά, η def_function() παίζει ζωτικό ρόλο στη φάση ανάλυσης και σημασιολογικής ανάλυσης της γλώσσας, επιτρέποντας τον σωστό χειρισμό των ένθετων συναρτήσεων στο πρόγραμμα.

```
def declarations():
    global eksodos

    while (eksodos[0] == declare_token):
        declaration_grammi()
```

- Η συνάρτηση **declarations()** είναι υπεύθυνη για την επεξεργασία πολλαπλών γραμμών δηλώσεων. Χρησιμοποιεί έναν βρόχο while για επανάληψη, εφόσον το πρώτο στοιχείο της λίστας eksodos είναι ίσο με το declare_token. Μέσα στο βρόχο, καλεί τη συνάρτηση declaration_grammi() για να επεξεργαστεί κάθε μεμονωμένη γραμμή δήλωσης.

```
def declaration_grammi():
    global eksodos
    global grammi

    if (eksodos[0] == declare_token):
```

- Η συνάρτηση **declaration_grammi()** ορίζει όλες τις μεταβλητές του προγράμματος, οι οποίες εμφανίζονται μετά την δεσμευμένη λέξη “#declare”. Στη συνέχεια καλείται η συνάρτηση id_list().

- Η συνάρτηση **statement()** καθορίζει τον τύπο της δήλωσης με βάση το πρώτο token στη λίστα eksodos. Ελέγχει εάν το πρώτο token είναι gramma_token, return_token ή print_token και αν ισχύει κάτι από αυτά καλεί τη συνάρτηση simple_statement(). Αλλιώς ελέγχει άν το πρώτο token είναι if_token ή while_token και αν ισχύει καλεί τη συνάρτηση structured_statement(). Εάν δεν πληρείται καμία από τις παραπάνω προϋποθέσεις, εκτυπώνει ένα μήνυμα σφάλματος που υποδεικνύει μη έγκυρη δήλωση και βγαίνει από το πρόγραμμα.

```
def statement():
    global eksodos
    global grammi

    if (eksodos[0] == gramma_token or eksodos[0] == return_token or eksodos[0] == print_token):
        simple_statement()
    elif (eksodos[0] == if_token or eksodos[0] == while_token):
        structured_statement()
    else:
        print('ERROR: Den emfanizetai swsto statement.', 'Grammi:', grammi)
        exit(-1)
```

- Η συνάρτηση **statements()** είναι υπεύθυνη για την επεξεργασία πολλαπλών δηλώσεων στη σειρά. Ξεκινά καλώντας τη συνάρτηση statement() για να επεξεργαστεί την πρώτη δήλωση. Στη συνέχεια, χρησιμοποιεί έναν βρόχο while για επανάληψη, εφόσον το πρώτο token στη λίστα eksodos είναι gramma_token, return_token, print_token, if_token ή while_token. Μέσα στον βρόχο, καλεί τη συνάρτηση statement() για να επεξεργαστεί κάθε επόμενη δήλωση.

```
def statements():
    global eksodos

    statement()
    while (eksodos[0] == gramma_token or eksodos[0] == return_token or eksodos[0] == print_token or eksodos[
        0] == if_token or eksodos[0] == while_token):
        statement()
```

- Η συνάρτηση **simple_statement()** χειρίζεται απλές δηλώσεις, οι οποίες μπορεί να είναι μια δήλωση ανάθεσης, δήλωση εκτύπωσης ή δήλωση επιστροφής. Ελέγχει τον τύπο του πρώτου token στη λίστα eksodos για να καθορίσει τον κατάλληλο τύπο δήλωσης.
  - ➢ Εάν το πρώτο token είναι gramma_token, καλεί τη συνάρτηση assignment_stat().
  - ➢ Εάν είναι print_token, καλεί τη συνάρτηση print_stat().
  - ➢ Εάν είναι return_token, καλεί τη συνάρτηση return_stat().

```
def simple_statement():
    global eksodos

    if (eksodos[0] == gramma_token):
        assignment_stat()
    elif (eksodos[0] == print_token):
        print_stat()
    elif (eksodos[0] == return_token):
        return_stat()
```

- Η συνάρτηση **structured_statement()** χειρίζεται δομημένες εντολές, ειδικά εντολές if και while. Ελέγχει τον τύπο του πρώτου token στη λίστα eksodos για να προσδιορίσει αν είναι if_token ή while_token.
  - Εάν είναι ένα if_token, καλεί τη συνάρτηση if_stat() για να επεξεργαστεί τη δήλωση if.
  - Εάν είναι while_token, καλεί τη συνάρτηση while_stat() για να επεξεργαστεί τη δήλωση while.

```
def structured_statement():
    global eksodos

    if (eksodos[0] == if_token):
        if_stat()
    elif (eksodos[0] == while_token):
        while_stat()
```

- Η συνάρτηση **assignment_stat()** επεξεργάζεται δηλώσεις ανάθεσης. Ελέγχει αν το πρώτο token στη λίστα eksodos είναι gramma_token (ID). Εάν είναι, εκχωρεί το 2ο token της λιστας eksodos στη μεταβλητή myid και προχωρά στον έλεγχο των επόμενων tokens για να διασφαλίσει τη σωστή σύνταξη. Επειτα επαληθεύει την παρουσία του πρόσημου ίσου, μιας έκφρασης και είτε ενός ερωτηματικού, είτε ενός ακέραιου token, ακολουθούμενο από συγκεκριμένες παρενθέσεις και ερωτηματικά.

```
if (eksodos[0] == gramma_token):
    myid = eksodos[1]
```

- Η συνάρτηση **print_stat()** επεξεργάζεται δηλώσεις εκτύπωσης. Ελέγχει εάν το πρώτο token στη λίστα eksodos είναι ένα print_token. Επαληθεύει την παρουσία παρενθέσεων και ερωτηματικού γύρω από μια έκφραση.

- Η συνάρτηση **return_stat()** επεξεργάζεται δηλώσεις επιστροφής. Ελέγχει εάν το πρώτο token στη λίστα eksodos είναι ένα return_token. Επαληθεύει την παρουσία παρενθέσεων και ερωτηματικού γύρω από μια έκφραση.

- Η συνάρτηση **if_stat()** χειρίζεται την ανάλυση μιας δήλωσης if στον κώδικα. Ελέγχει εάν το πρώτο token στη λίστα eksodos είναι ένα if_token και στη συνέχεια προχωρά στην ανάλυση της συνθήκης που περικλείεται σε παρένθεση. Μετά από αυτό, αναμένει ένα σύμβολο *άνω και κάτω τελείας* ακολουθούμενο είτε από μια μεμονωμένη πρόταση είτε από ένα μπλοκ εντολών που περικλείονται στα **#{** και **#}.** Χειρίζεται επίσης το προαιρετικό τμήμα "else", όπου ελέγχει για μια λέξη-κλειδί "else" ακολουθούμενη από άνω και κάτω τελεία και είτε μια μεμονωμένη πρόταση, είτε ένα μπλοκ εντολών.

- Η συνάρτηση **while_stat()** χειρίζεται την ανάλυση μιας πρότασης βρόχου while. Ελέγχει εάν το πρώτο token στη λίστα eksodos είναι ένα while_token και στη συνέχεια προχωρά στην ανάλυση της συνθήκης που περικλείεται σε παρένθεση. Μετά από αυτό, αναμένει ένα σύμβολο άνω και κάτω τελείας και είτε μία μεμονωμένη πρόταση είτε ένα μπλοκ εντολών που περικλείονται στα **#{** και **#}.**

- Η συνάρτηση **id_list()** χειρίζεται την ανάλυση μιας *λίστας αναγνωριστικών*. Αναμένει τουλάχιστον ένα αναγνωριστικό (ID) και αν υπάρχουν περισσότερα, χωρίζονται με κόμματα. Αυτή η συνάρτηση χρησιμοποιείται για την ανάλυση της λίστας μεταβλητών ή ορισμάτων σε δηλώσεις συναρτήσεων.

- Η συνάρτηση **expression()** είναι υπεύθυνη για την ανάλυση μιας αριθμητικής παράστασης. Ξεκινά ελέγχοντας για ένα προαιρετικό σύμβολο (+/-) και στη συνέχεια αναλύει τον πρώτο όρο της έκφρασης. Συνεχίζει την ανάλυση πρόσθετων όρων εάν υπάρχουν τελεστές πρόσθεσης ή αφαίρεσης (+/-) μεταξύ τους. Κάθε όρος αποθηκεύεται σε μια *προσωρινή μεταβλητή* και δημιουργείται μια νέα προσωρινή μεταβλητή για την αποθήκευση του αποτελέσματος της λειτουργίας. Η συνάρτηση επιστρέφει το τελικό αποτέλεσμα της έκφρασης.

```
def expression():
    global eksodos
    global grammi

    optional_sign()

    T1place = term()

    while (eksodos[0] == prosthesi_token or eksodos[0] == afairesi_token):
        prosthesi_afairesi = ADD_OP()
        T2place = term()

        w = newtemp()
        genquad(prosthesi_afairesi, T1place, T2place, w)
        T1place = w

    Eplace = T1place

    return Eplace
```

- Η συνάρτηση **term()** χειρίζεται την ανάλυση ενός όρου στην έκφραση. Αναλύει τον πρώτο παράγοντα και στη συνέχεια συνεχίζει την ανάλυση πρόσθετων παραγόντων εάν υπάρχουν τελεστές πολλαπλασιασμού ή διαίρεσης (*) ή (/) μεταξύ τους. Παρόμοια με τη συνάρτηση **expression** , κάθε παράγοντας αποθηκεύεται σε μια προσωρινή μεταβλητή και δημιουργείται μια νέα προσωρινή μεταβλητή για την αποθήκευση του αποτελέσματος της λειτουργίας. Η συνάρτηση επιστρέφει το τελικό αποτέλεσμα της έκφρασης.

```
def term():
    global eksodos
    global grammi

    F1place = factor()

    while (eksodos[0] == pollaplasiasmos_token or eksodos[0] == diairesi_token):
        pollaplasiasmos_diairesi = MUL_OP()
        F2place = factor()

        w = newtemp()
        genquad(pollaplasiasmos_diairesi, F1place, F2place, w)
        F1place = w

    Tplace = F1place

    return Tplace
```

- Η συνάρτηση **factor()** χειρίζεται την ανάλυση ενός παράγοντα, ο οποίος μπορεί να είναι ένας ακέραιος αριθμός, μια έκφραση που περικλείεται σε παρενθέσεις ή μια λέξη ακολουθούμενη από μια idtail. Εάν ο παράγοντας είναι ακέραιος, η τιμή του αποθηκεύεται *απευθείας*. Εάν ο παράγοντας περικλείεται σε παρένθεση, η συνάρτηση **expression** καλείται αναδρομικά για να αναλύσει την έκφραση μέσα στις παρενθέσεις. Εάν ο παράγοντας είναι λέξη, καλείται η συνάρτηση **idtail**. Η συνάρτηση επιστρέφει την τιμή του παράγοντα.

- Η συνάρτηση **idtail()** χειρίζεται την ανάλυση μιας ουράς αναγνωριστικού, η οποία μπορεί να περιλαμβάνει *κλήσεις συναρτήσεων ή δείκτη πίνακα.* Εάν η ουρά ξεκινά με αριστερή παρένθεση, υποδεικνύει μια κλήση συνάρτησης. Στη συνέχεια, η συνάρτηση αναλύει την πραγματική λίστα παραμέτρων καλώντας τη συνάρτηση

actual_par_list, η οποία χειρίζεται την ανάλυση των expressions για ορίσματα συνάρτησης.

- Η συνάρτηση **actual_par_list()** χειρίζεται την ανάλυση της πραγματικής λίστας παραμέτρων για μια κλήση συνάρτησης. Ελέγχει εάν υπάρχουν εκφράσεις (ορίσματα συνάρτησης) προς ανάλυση και για κάθε έκφραση καλεί τη συνάρτηση expression για να την αναλύσει.

- Η συνάρτηση **optional_sign()** χειρίζεται την ανάλυση ενός προαιρετικού σημείου (+/-) που μπορεί να προηγείται ενός όρου σε μια παράσταση. Εάν υπάρχει το σύμβολο, αναλύεται καλώντας τη συνάρτηση **ADD_OP**. Εάν το σήμα δεν υπάρχει, δεν γίνεται καμία ενέργεια.

```
def optional_sign():
    global eksodos
    global grammi

    if (eksodos[0] == prosthesi_token or eksodos[0] == afairesi_token):
        ADD_OP()
```

- Η συνάρτηση **ADD_OP()** χειρίζεται την ανάλυση των τελεστών *πρόσθεσης και αφαίρεσης*. Ελέγχει εάν το επόμενο token είναι είτε ο τελεστής πρόσθεσης είτε αφαίρεσης. Εάν είναι, εκχωρεί τον τελεστή στη μεταβλητή addOp, καταναλώνει το token και ενημερώνει τις μεταβλητές eksodos και grammi. Τέλος, επιστρέφει τον εκχωρημένο τελεστή addOp.

```
def ADD_OP():
    global eksodos
    global grammi

    if (eksodos[0] == prosthesi_token):
        addOp = eksodos[1]
        eksodos = lex_analitis()
        grammi = eksodos[2]


    elif (eksodos[0] == afairesi_token):
        addOp = eksodos[1]
        eksodos = lex_analitis()
        grammi = eksodos[2]

    return addOp
```

```
def MUL_OP():
    global eksodos
    global grammi

    if (eksodos[0] == pollaplasiasmos_token):
        oper = eksodos[1]
        eksodos = lex_analitis()
        grammi = eksodos[2]

    elif (eksodos[0] == diairesi_token):
        oper = eksodos[1]
        eksodos = lex_analitis()
        grammi = eksodos[2]

    return oper
```

- Η συνάρτηση **MUL_OP()** χειρίζεται την ανάλυση των τελεστών *πολλαπλασιασμού και διαίρεσης*. Ελέγχει αν το επόμενο token είναι είτε ο τελεστής πολλαπλασιασμού είτε διαίρεσης. Εάν είναι, εκχωρεί τον τελεστή στη μεταβλητή oper, καταναλώνει το token και ενημερώνει τις μεταβλητές eksodos και grammi. Τέλος, επιστρέφει τον εκχωρημένο τελεστή oper.

- Η συνάρτηση **condition()** χειρίζεται την ανάλυση των παραστάσεων υπό όρους στη CutePy. Αρχικοποιεί δύο λίστες, Ctrue και Cfalse, για να αποθηκεύσει τους αληθείς και ψευδείς κλάδους της συνθήκης. Καλεί τη συνάρτηση **bool_term()** για να αναλύσει

τον πρώτο δυαδικό όρο και εκχωρεί τους προκύπτοντες αληθείς και ψευδείς κλάδους σε Ctrue και Cfalse.

- Η συνάρτηση **bool_term()** χειρίζεται την ανάλυση *boolean* όρων. Αρχικοποιεί δύο λίστες, BTtrue και BTfalse, για να αποθηκεύσει τους true και false κλάδους του όρου. Καλεί τη συνάρτηση **bool_factor()** για να αναλύσει τον πρώτο δυαδικό παράγοντα και εκχωρεί τους αληθείς και ψευδείς κλάδους που προκύπτουν σε BTtrue και BTfalse.

- Η συνάρτηση **bool_factor()** χειρίζεται την ανάλυση *boolean* παραγόντων. Αρχικοποιεί δύο λίστες, BFtrue και BFfalse, για να αποθηκεύσει τους αληθείς και ψευδείς κλάδους του παράγοντα. Αρχικά ελέγχει εάν το επόμενο token είναι ο λογικός τελεστής NOT. Αν είναι, καταναλώνει το token και ελέγχει αν το επόμενο token είναι αριστερή αγκύλη. Εάν είναι, καταναλώνει το token, καλεί τη συνάρτηση συνθήκης για να αναλύσει την ένθετη συνθήκη και ελέγχει εάν το επόμενο token είναι μια δεξιά αγκύλη. Εάν είναι, καταναλώνει το token και εκχωρεί τους αληθείς και ψευδείς κλάδους της ένθετης συνθήκης σε BFtrue και BFfalse αντίστοιχα. Εάν το επόμενο token είναι μια αριστερή αγκύλη χωρίς προηγούμενο τελεστή NOT, ακολουθεί μια παρόμοια διαδικασία όπως παραπάνω, αλλά εκχωρεί τους κλάδους της ένθετης συνθήκης με αντίστροφη σειρά. Εάν το επόμενο token δεν είναι ούτε λογικός τελεστής NOT ούτε αριστερή αγκύλη, υποδεικνύει μια έκφραση σύγκρισης. Καλεί τη συνάρτηση **expression** δύο φορές για να αναλύσει δύο παραστάσεις, ακολουθούμενη από τη συνάρτηση REL_OP για να αναλύσει τον σχεσιακό τελεστή. Δημιουργεί τετραπλάσια για τη σύγκριση και ενημερώνει τους κλάδους BFtrue και BFfalse ανάλογα.

- Η συνάρτηση **REL_OP()** χειρίζεται την ανάλυση των *σχεσιακών τελεστών*. Ελέγχει εάν το επόμενο token είναι ένας από τους υποστηριζόμενους σχεσιακούς τελεστές, όπως ίσος, μικρότερος ή ίσος , μεγαλύτερος ή ίσος, όχι ίσος, μεγαλύτερος από ή μικρότερος από. Εάν ταιριάζει με κάποιον από αυτούς τους τελεστές, εκχωρεί τον τελεστή σε μεταβλητή relop, καταναλώνει το token και ενημερώνει τις μεταβλητές eksodos και grammi. Διαφορετικά, υποδεικνύει συντακτικό σφάλμα και βγαίνει από το πρόγραμμα. Τέλος, επιστρέφει τον εκχωρημένο σχεσιακό τελεστή relop.

- Η συνάρτηση **call_main_part()** είναι το σημείο εισόδου του προγράμματος.Ελέγχει εάν το πρώτο token στη λίστα eksodos είναι ένα if_token, ακολουθούμενο από το αναγνωριστικό "**name**", τον τελεστή "**==**" και τη συμβολοσειρά "**main**". Εάν πληρούνται όλες αυτές οι προϋποθέσεις, προχωρά στην ανάλυση ενός μπλοκ δηλώσεων **main_function_call**. Κάθε main_function_call περικλείεται σε παρένθεση. Μετά την ανάλυση του μπλοκ, δημιουργεί ενδιάμεσο κώδικα για τις κλήσεις κύριας συνάρτησης, συμπεριλαμβανομένης μιας δήλωσης "begin_block", των κλήσεων της κύριας συνάρτησης, μιας δήλωσης "halt" και μιας δήλωσης "end_block". Τέλος, βγάζει τον πίνακα συμβόλων,διαγράφει το τρέχων scope και βγαίνει από το πρόγραμμα.

```
call_main_part
        :
        'if' '__name__' '==' '"__main__"' ':'
            (  main_function_call )+
        ;
```

- Η συνάρτηση **main_function_call()** αναλύει μια δήλωση κλήσης κύριας συνάρτησης. Ελέγχει εάν το επόμενο token είναι μια λέξη, ακολουθούμενη από το άνοιγμα και το κλείσιμο παρενθέσεων. Εάν η σύνταξη είναι σωστή, δημιουργεί ενδιάμεσο κώδικα για την κλήση συνάρτησης και προχωράει στο επόμενο token. Εάν υπάρχει ερωτηματικό μετά τη δεξιά παρένθεση, προχωρά στην επόμενη κλήση κύριας συνάρτησης. Διαφορετικά, υποδεικνύει συντακτικό σφάλμα και βγαίνει από το πρόγραμμα.

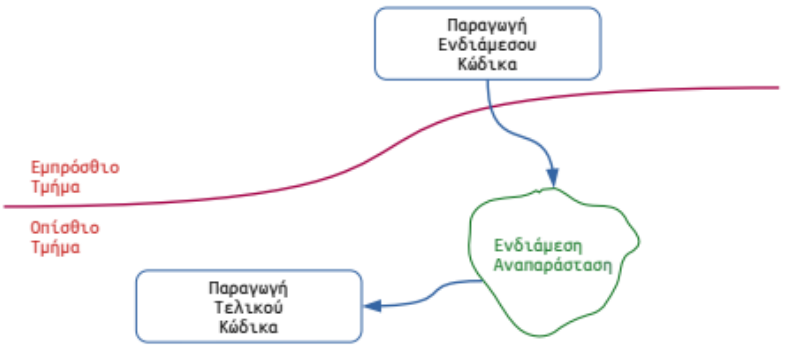
- Η συνάρτηση **intCode(written_file)** παίρνει ένα αρχείο εγγραφής (written_file) ως όρισμα και εκτελεί μια επανάληψη για κάθε στοιχείο του all_QuadsFinal. Εντός της επανάληψης, αποθηκεύει το τρέχον στοιχείο του all_QuadsFinal στην προσωρινή μεταβλητή quad. Στη συνέχεια, η συνάρτηση εγγράφει τα διάφορα στοιχεία του quad στο αρχείο written_file, χρησιμοποιώντας τη μέθοδο write(). Κάθε στοιχείο του quad αντιστοιχεί σε έναν αριθμό ή μια συμβολοσειρά, και ανάμεσα τους προστίθενται κενά διαστήματα για διαχωρισμό. Την λίστα με όλες τις τετράδες τις γράφουμε στο αρχείο "**file_with_quads.int**"

```
def intCode(written_file):
    for i in range(len(all_QuadsFinal)):
        quad = all_QuadsFinal[i]
        written_file.write(str(quad[0]))
        written_file.write(":  ")
        written_file.write(str(quad[1]))
        written_file.write("  ")
        written_file.write(str(quad[2]))
        written_file.write("  ")
        written_file.write(str(quad[3]))
        written_file.write("  ")
        written_file.write(str(quad[4]))
        written_file.write("\n")
```

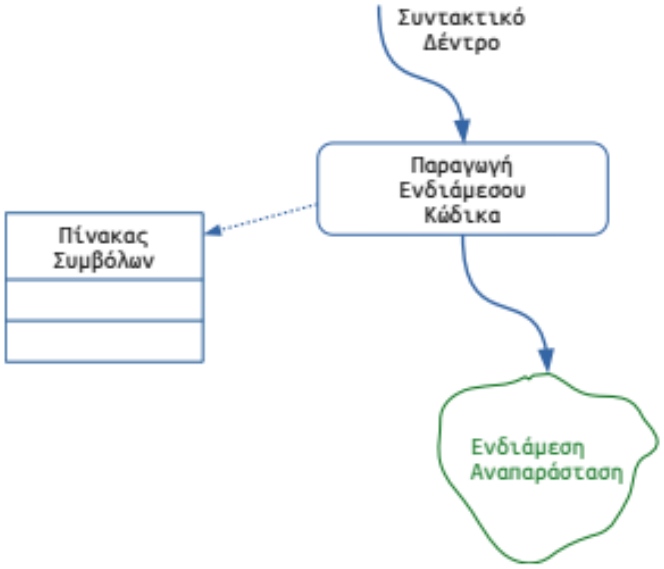
```
file_with_quads = open('file_with_quads.int', 'w')
```

ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

Η μετατροπή του κώδικα από την αρχική γλώσσα στην τελική *δεν γίνεται απευθείας*. Μεσολαβεί η μετατροπή του σε μία **ενδιάμεση γλώσσα**, την οποία συνηθίζουμε να λέμε και *ενδιάμεση αναπαράσταση*, η οποία εξακολουθεί να θεωρείται γλώσσα υψηλού επιπέδου, αλλά οι δομές της δεν είναι τόσο σύνθετες, όσο αυτές της αρχικής γλώσσας, ενώ η συντακτική της ανάλυση είναι τετριμμένη.



Ο ενδιάμεσος κώδικας αποτελεί το **ενδιάμεσο** στάδιο μετατροπής της αρχικής μας γλώσσας σε γλώσσα μηχανής (assembly). Σε συνδυασμό με τον **πίνακα συμβόλων**, που θα σχολιάσουμε στη συνέχεια, μας βοηθά να παράγουμε τον **τελικό κώδικα**, δηλαδή κώδικα σε γλώσσα μηχανής.



Η σχεδίαση του μεταγλωττιστή απλοποιείται σημαντικά και χωρίζεται σε δύο ανεξάρτητες φάσεις:

- τη φάση πριν την παραγωγή του ενδιάμεσου κώδικα, το εμπρόσθιο τμήμα (front end).
- τη φάση μετά την παραγωγή του ενδιάμεσου κώδικα. το οπίσθιο τμήμα (back end)

Η ενδιάμεση αναπαράσταση αποτελεί μέσο επικοινωνίας ανάμεσα στο εμπρόσθιο και το οπίσθιο τμήμα του μεταγλωττιστή.

Ο ενδιάμεσος κώδικας παράγει ένα σύνολο από ***τετράδες*** της μορφής:

1: + , a ,b ,T_1 το οποίο αντιστοιχεί σε T_1 = a + b
2: * , t_1,2 ,T_2 το οποίο αντιστοιχεί σε T_2 = t_1 * 2

Όπου T_1 και T_2 προσωρινές μεταβλητές.

Δηλαδή έχουμε έναν τελεστή και τρία τελούμενα. Οι τετράδες είναι αριθμημένες σύμφωνα με την σειρά εμφάνισης τους στο πρόγραμμα εισόδου, έτσι ώστε σε κάθε τετράδα να μπορούμε να αναφερθούμε χρησιμοποιώντας τον αριθμό της ως ετικέτα. Έτσι, αν μετρήσουμε και την ετικέτα, πρόκειται για μία τετράδα που αποτελείται από ... πέντε πράγματα. Οι τετράδες παράγονται για κάθε γραμμή του κώδικα και εκτελούνται με την σειρά αρίθμησης του, εκτός αν κάποια τετράδα υποδείξει μεταπήδηση σε κάποια άλλη (πχ σε κάποια συνθήκη).

Έχουμε τους εξής **τελεστές**:

- **Τελεστής αριθμητικών πράξεων**
  op, x, y, z
  όπου το op μπορεί να είναι ένα εκ των: + , - , * , /
  τα τελούμενα x,y μπορεί να είναι:
  - ονόματα μεταβλητών
  - αριθμητικές σταθερές

  και το τελούμενο z μπορεί να είναι όνομα μεταβλητής.
  Εφαρμόζεται ο τελεστής op στα τελούμενα x και y και το αποτέλεσμα τοποθετείται στο τελούμενο z.
- **Τελεστής εκχώρησης**
  =, x, _, z
  το τελούμενο x μπορεί να είναι:
  - όνομα μεταβλητής,
  - αριθμητική σταθερά,

  και το τελούμενο z μπορεί να είναι όνομα μεταβλητής.
  Η τιμή του x που εκχωρείται στη μεταβλητή z αντιστοιχεί στη εκχώρηση z=x.
- **Τελεστής άλματος χωρίς συνθήκη**
  jump, _, _, z
  μεταπήδηση χωρίς όρους στη θέση z.
- **Τελεστής άλματος με συνθήκη**
  relop, x, y, z
  όπου relop είναι ένας από τους τελεστές:
  =, >, <, <>, >=, <=
  αν ισχύει η x relop y τότε μεταπήδηση στη θέση z.

Η λίστα με τις τετράδες αποτελεί το μέσο επικοινωνίας της φάσης της παραγωγής ενδιάμεσου κώδικα και της παραγωγής τελικού κώδικα, αφού ο τελικός κώδικας παράγεται με βάση αυτή τη λίστα και πληροφορία που αντλεί από τον πίνακα συμβόλων.

Αρχή και τέλος ενότητας:

- begin_block, name, _, _
  αρχή υποπρογράμματος ή προγράμματος με το όνομα name.
- end_block, name, _, _
  τέλος υποπρογράμματος ή προγράμματος με το όνομα name.
- halt , _, _, _
  τερματισμός προγράμματος.

Ομαδοποίηση κώδικα:

```
begin_block, name, _, _
end_block, name, _, _
```

Οι εντολές **begin_block** και **end_block** χρησιμοποιούνται για να ομαδοποιήσουμε εντολές ενδιάμεσου κώδικα. Βασικά, αυτό που θέλουμε να οριοθετήσουμε με τις begin_block και end_block είναι η αρχή και το τέλος του ενδιάμεσου κώδικα που παρήχθη για μια συνάρτηση ή για το κυρίως πρόγραμμα. Έτσι, στην αρχή του κώδικα μιας συνάρτησης ή του κυρίως προγράμματος και πριν την πρώτη εκτελέσιμη εντολή, τοποθετούμε μία begin_block με το όνομα της συνάρτησης ή του κυρίως προγράμματος και στο τέλος μία end_block.

Συναρτήσεις:

- par, x, m, _
  όπου x παράμετρος συνάρτησης και m ο τρόπος μετάδοσης
  - CV : μετάδοση με τιμή
  - REF: μετάδοση με αναφορά
  - RET: επιστροφή τιμής συνάρτησης

- call, name, _, _
  κλήση συνάρτησης name

- ret, x, _, _
  επιστροφή τιμής συνάρτησης

Παράδειγμα κλήσης συνάρτησης

x = foo (in a, inout b)

θα μπορούσε ισοδύναμα να είχαμε:

χωρίς να σημαίνει ότι αυτό ακριβώς θα βγάλει και ο μεταγλωττιστής

100: par , a , cv , _
110: par , b , ref , _
120: par , T_1 , ret , _
130: call , foo , _ , _ , _
140: ... τιμή στο x

Παράδειγμα κλήσης διαδικασίας

call foo (in a, inout b)

θα μπορούσε ισοδύναμα να είχαμε:

χωρίς να σημαίνει ότι αυτό ακριβώς θα βγάλει και ο μεταγλωττιστής

100: par , a , cv , _
110: par , b , ref , _
120: call , foo , _ , _ , _
130 ...

Βοηθητικές Συναρτήσεις:

- ☐ **nextquad()**
  Επιστρέφει τον αριθμό της επόμενης τετράδας που παράγεται.

```
global all_Quads  # lista me oles tis tetrades
all_Quads = []

countquad = 1  # o arithmos pou mpainei aristera apo tin tetrada


def nextquad():  # Epistrefei ton arithmo ths epomenis tetradas pou prokeitai na paraxthei
    global countquad

    return countquad

all_QuadsFinal = []
```

☐ genquad(op, x, y, z)

Δημιουργεί την επόμενη τετράδα (op, x, y, z)

```
def genquad(op, x, y, z):  # Dhmiourgei tin epomeni tetrada (op, x, y, z)
    global countquad
    global all_Quads
    global all_QuadsFinal

    quad = [nextquad(), op, x, y, z]
    all_Quads.append(quad)
    countquad = countquad + 1
    all_QuadsFinal.append(quad)

    return quad
```

☐ newtemp()

Δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή
που είναι της μορφής: T_1, T_2, T_3…

```
T_i = 0
all_Temp_Var = []  # lista me oles tis proswrines metavlites
3 usages
def newtemp():  # Dhmiourgei kai epistrefei mia nea proswrini metavliti
    # oi proswrines metavlites einai tis morfhs T_1, T_2, T_3...
    global all_Temp_Var
    global T_i

    temp_Var = '%' + str(T_i + 1)  # Dhmiourgei nea proswrini metavliti me symvolo %
    all_Temp_Var.append(temp_Var)
    T_i = T_i + 1

    entity = Entity()
    entity.type = 'PROSWRINH_METAVLHTH'
    entity.name = temp_Var
    entity.ProswriniMetavliti.offset = compute_offset()
    new_entity(entity)

    return temp_Var
```

☐ emptylist()

Δημιουργεί μία κενή λίστα τετράδων.

```
def emptylist():  # Dhmiourgei mia keni lista etiketwn tetradwn
    empty_List = []

    return empty_List
```

☐ makelist(x)

Δημιουργεί μία λίστα τετράδων που περιέχει μόνο το x.

```
def makelist(x):  # Dhmiourgei mia lista etiketwn tetradwn pou periexei mono to x
    xlist = [x]

    return xlist
```

☐ merge($list_1$, $list_2$)

Δημιουργεί μία λίστα τετράδων από τη συνένωση των λιστών $list_1$, $list_2$.

```
def merge(list1, list2):  # Dhmiourgei mia lista etiketwn tetradwn apo th synenwsh twn listwn list1, list2
    list = []
    list = list + list1 + list2

    return list
```

☐ backpatch(list,z)

Η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η backpatch επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

```
def backpatch(list,z):  # H lista list apoteleitai apo deiktes se tetrades twn opoiwn to teleutaio teloumeno den einai symplhrwmeno
    # h backpatch episkeptetai mia mia tis tetrades autes kai sumplirwnei me thn etiketa z
    global all_Quads

    for quad in all_Quads:
        if (quad[0] in list and quad[4] == '_'):
            quad[4] = z

    return
```

Αφού υλοποιήσουμε τις παραπάνω βοηθητικές συναρτήσεις, μπορούμε να ξεκινήσουμε την μετάφραση σε ενδιάμεσο κώδικα. Αυτό που απαιτείται είναι να *προσθέσουμε εντολές στον συντακτικό αναλυτή*, χρησιμοποιώντας τις παραπάνω συναρτήσεις ώστε να καταφέρουμε να παράγουμε κατάλληλες τετράδες, δηλαδή υλοποίηση ενδιάμεσου κώδικα.

Δομές Ενδιάμεσου Κώδικα

Αριθμητικές Παραστάσεις:

Κάθε **κανόνας** της γραμματικής περιγράφει τις αριθμητικές εκφράσεις σαν ένα σύνολο υπολογισμών που υλοποιούνται από τον ενδιάμεσο κώδικα. Σύμφωνα με τη γραμματική της γλώσσας CutePy, υποστηρίζονται τέσσερις **αριθμητικές πράξεις** (πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση) καθώς και η ομαδοποίηση/προτεραιότητα ανάμεσα σε αυτές ορίζεται με τη χρήση παρενθέσεων. Η γραμματική ακολουθεί:

```
# addition
E   →   T(1) ( + T(2) )*
# multiplication
T   →   F(1) ( * F(2) )*
# priority with parentheses
F   →   ( E )
# terminal symbols
F   →   ID
```

Τα $T^{(1)}$ και $T^{(2)}$ δεν αποτελούν διαφορετικό κανόνα, απλά διαφορετική εμφάνιση του κανόνα στη γραμματική και χρειαζόμαστε έναν συμβολισμό για να αναφερόμαστε και να διαχωρίζουμε εύκολα τις εμφανίσεις αυτές. Πρόκειται για διαφορετικές κλήσεις της ίδιας συνάρτησης.

Κάθε κανόνας, με βάση τα δεδομένα που θα συλλέξει από τα μη τερματικά σύμβολα και με βάση τα τερματικά σύμβολα που θα αναγνωρίσει, θα κάνει τα εξής:

- θα παραγάγει ενδιάμεσο κώδικα, όπου και εάν απαιτείται. Διαισθητικά μπορούμε να φανταστούμε ότι μία αριθμητική παράσταση πρέπει να παραγάγει κώδικα όταν εκτελείται μία πρόσθεση, ένας πολλαπλασιασμός ή εκχωρείται τιμή σε μία μεταβλητή, είτε λόγω κάποιου υπολογισμού, είτε λόγω αναγνώρισης κάποιου τερματικού συμβόλου.
- θα προετοιμάσει και θα προωθήσει πληροφορία στον κανόνα που τον κάλεσε. Την πληροφορία αυτήν την αναμένει ο κανόνας που τον κάλεσε προκειμένου να συνθέσει τον δικό του ενδιάμεσο κώδικα ή να συνθέσει την πληροφορία που αυτός θα προετοιμάσει και προωθήσει με τη σειρά του στον κανόνα που τον κάλεσε.

Τα *αποτελέσματα των υπολογισμών αποθηκεύονται σε μεταβλητές* που έχει δηλώσει ο προγραμματιστής ή σε προσωρινές μεταβλητές. Οι μεταβλητές αυτές ονομάζονται ***place***.

Πιο αναλυτικά οι αριθμητικές πράξεις:

- **Πρόσθεση (με μείον (-) αντίστοιχα για αφαίρεση)**

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

  Για το {P1} αρχικά δημιουργούμε μια προσωρινή μεταβλητή w με την newTemp() που θα κρατάει το μέχρι στιγμής αποτέλεσμα. Στην συνέχεια δημιουργούμε την τετράδα που υλοποιεί την πρόσθεση ή αφαίρεση ως εξής genquad(‘’+’’, T1.place,T2.place,w) και βάζουμε το αποτέλεσμα στην T1 ώστε χρησιμοποιηθεί αν έχουμε και άλλα T2. Για το {P2} βάζουμε στο E το αποτέλεσμα και έχουμε E.place = T1.place.

❖ **Πολλαπλασιασμός (με (/) αντίστοιχα για διαίρεση)**

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

Για τον πολλαπλασιασμό και την διαίρεση ακολουθούμε την ίδια λογική με την πρόσθεση/αφαίρεση.

{P1}: w = newTemp()
genquad("×",F1.place,F2.place,w)
F1.place=w

{P2}: T.place=F1.place

❖ **Προτεραιότητα στις παρενθέσεις**

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

Για την συγκεκριμένη δομή έχουμε μια απλή μετάθεση του E στο F με την εντολή F.place = E.place.

❖ **Τερματικά σύμβολα**

$F \rightarrow ID \{P1\}$

Για την συγκεκριμένη δομή έχουμε μια απλή μετάθεση του id στο F με την εντολή F.place = id.place.

Λογικές Παραστάσεις – OR:

$B \rightarrow Q^1 \{P1\} (or \{P2\} Q^2 \{P3\})^*$

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

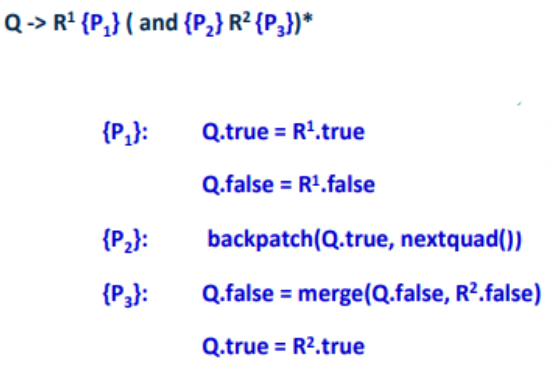
$\{P_1\}$: B.true = Q^1.true
B.false = Q^1.false
$\{P_2\}$: backpatch(B.false, nextquad())
$\{P_3\}$: B.true = merge(B.true, Q^2.true)
B.false = Q^2.false

Για το or με έναν αληθή όρο η λογική παράσταση αποτιμάται αληθής αλλιώς αποτιμάται ψευδής.
Αρχικά θα πρέπει να μεταφέρουμε τις τετράδες από την λίστα Q^1 στην λίστα B και για τις δύο αποτιμήσεις (true και false).
Οπότε για το {P1} έχουμε B.true = Q^1 .true και B.false = Q^1 .false. Για το {P2} χρειάζεται ένα **backpatch(B.false,nextquad()**. Τέλος για το {P3} κάνουμε ένα merge στην λίστα με τις τετράδες που αντιστοιχούν σε αληθή αποτίμηση. Τέλος βάζουμε στην λίστα B.false την τετράδα Q^2 .false που περιέχει την τετράδα με την μη αληθή αποτίμηση της λογικής παράστασης.

Λογικές Παραστάσεις – AND:

B -> R^1 {P1} (and {P2} R^2 {P3})*

Για το and με έναν ψευδή όρο η λογική παράσταση αποτιμάται ψευδής αλλιώς αποτιμάται αληθής. Αρχικά θα πρέπει να μεταφέρουμε τις τετράδες από την λίστα R^1 στην λίστα Q και για τις δύο αποτιμήσεις (true και false) . Οπότε για το {P1} έχουμε Q.true = R^1 .true και Q.false = R^1 .false. Για το {P2} χρειάζεται ένα **backpatch(Q.true,nextquad()**). Τέλος για το {P3} κάνουμε ένα merge στην λίστα με τις τετράδες που αντιστοιχούν σε μη αληθή αποτίμηση. Τέλος βάζουμε στην λίστα Q.true την τετράδα R^2 .true που περιέχει την τετράδα με την αληθή αποτίμηση της λογικής παράστασης.



Λογικές Παραστάσεις – ΜΕΤΑΦΟΡΑ:

R -> (B) {P1}

Είναι ανάλογος με τον κανόνα του ορισμού προτεραιοτήτων με τις παρενθέσεις στις αριθμητικές εκφράσεις και χρειαζόμαστε μια απλή μεταφορά των τετράδων από την λίστα B στην λίστα R.

- o R.true=B.true
- o R.false=B.false

Λογικές Παραστάσεις – NOT:

R -> not (B) {P1}

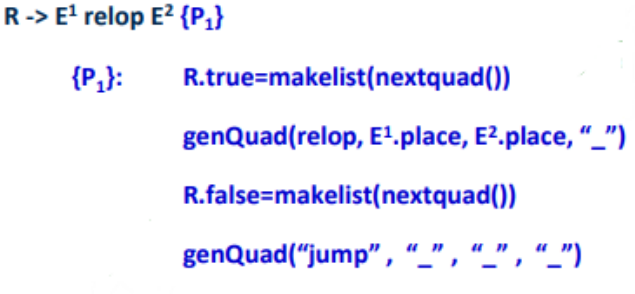
Το μόνο που χρειαζόμαστε για το not είναι μια αντιστροφή και μια μεταφορά των τετράδων από την λίστα B στην λίστα R.

- o R.true=B.false
- o R.false=B.true

Λογικές Παραστάσεις – RELOP:

R -> E^1 relop E^2 {P1}

Για την relop χρειάζεται να δημιουργήσουμε λίστες και για την αληθή και για την ψευδή αποτίμηση της συνθήκης δηλαδή **R.true = makelist(nextquad())** και **R.false=makelist(nextquad())**. Στην συνέχεια χρειάζεται να



δημιουργήσουμε την τετράδα για την αληθή αποτίμησης (genquad(relop,E1.place,E2.place,''_'')) και τετράδα για την μη αληθή αποτίμηση (genquad(''jump'',''_'',''_'',''_'',''_'')).

Κλήση Συνάρτησης:

Για την κλήση συνάρτησης σχηματίζουμε αρχικά τετράδες για τα ορίσματα της ανάλογα με τον τρόπο περάσματος τους π.χ. genquad(par,a,CV,''_'') για πέρασμα με τιμή. Στην συνέχεια δημιουργούμε μια προσωρινή μεταβλητή w με την newTemp() για να κρατήσουμε την τιμή επιστροφής και δημιουργούμε τις εξής δυο τετράδες: μια για την τιμή επιστροφής (genquad(par, w, RET, _)) και μια για την κλήση συνάρτησης (genquad(call, assign_v , _, _)).

Κλήση συνάρτησης:

error = assign_v (in a, inout b)

par, a, CV, _

par, b, REF, _

w = newTemp()

par, w, RET, _

call, assign_v , _, _

Εντολή return:

S -> return(E) {P1}

Χρειαζόμαστε μια εντολή για την επιστροφή τιμής μιας συνάρτησης. Το μόνο που χρειαζόμαστε είναι η δημιουργία μιας τετράδας με την genquad(''retv'', E.place,''_'',''_'') η οποία θα μας επιστρέφει την μεταβλητή Ε.

Εκχώρηση:

S -> id = E {P1}

Για την εκχώρηση μιας τιμής Ε σε μια μεταβλητή id χρειάζεται να δημιουργήσουμε μόνο μια τετράδα genquad(''='',E.place,''_'',id) η οποία θα εκχωρήσει την τιμή Ε στην μεταβλητή id.

Δομή while:

S -> while {P1} B do {P2} S[1]{P3}

Αρχικά χρειάζεται να αποθηκεύσουμε σε μια μεταβλητή την τετράδα στην οποία θα μεταβεί ξανά ο έλεγχος. Άρα έχουμε {P1} : Bquad = nextquad().
Για το {P2} θα χρειαστεί να κάνουμε backpatch όταν η συνθήκη είναι true έτσι ώστε να βρεθούμε στο S^1, άρα έχουμε backpatch(B.true,nextquad()).

S -> while {P1} B do {P2} S[1] {P3}

{P1}: Bquad:=nextquad()

{P2}: backpatch(B.true,nextquad())

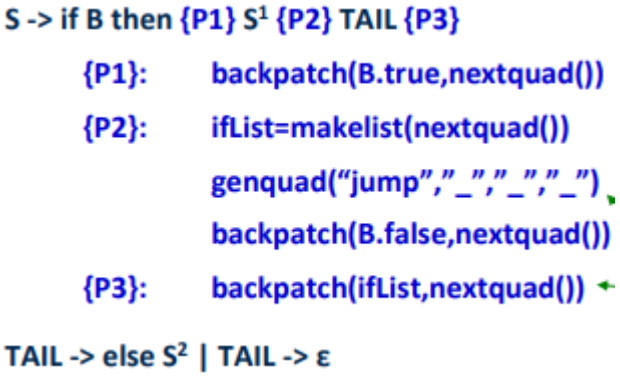
{P3}: genquad("jump","_","_",Bquad)

backpatch(B.false,nextquad())

Για το {P3} χρειάζεται αρχικά να δημιουργήσουμε την τετράδα genquad(‘’jump’’,”_”,”_”, Bquad) για να μεταβούμε στην αρχή της συνθήκης και να ξαναγίνει ο έλεγχος. Επίσης πρέπει να κάνουμε backpatch(B.false,nextquad()) και για την false αποτίμηση της συνθήκης έτσι ώστε να βρεθούμε εκτός while.

Δομή if:

S-> if B then {P1} S^1 {P2} TAIL {P3}
TAIL -> else S^2 | TAIL -> ε



Αρχικά θα χρειαστεί να κάνουμε backpatch όταν η συνθήκη είναι true έτσι ώστε να βρεθούμε στο S1, άρα έχουμε backpatch(B.true,nextquad()). Για το {P2} αρχικά δημιουργούμε μια λίστα ifList η οποία θα μας βοηθήσει να εκτελεστούν μόνο οι συνθήκες του if είτε του else οπότε έχουμε **ifList=makelist(nextquad())**.
Δημιουργούμε μια τετράδα **genquad(‘’jump”,”_”,”_”,”_”)** ώστε να μεταβούμε στην κατάλληλη τετράδα ανάλογα με την συνθήκη (false ή true) και κάνουμε backpatch(B.false,nextquad()) για την false αποτίμηση της συνθήκης B του if. Τέλος για το {P3} χρειάζεται μόνο ένα backpatch(ifList,nextquad())

Είσοδος- Έξοδος:

- Είσοδος **S -> input (id) {P1}**
  Για το input χρειάζεται μόνο ο ορισμός μιας τετράδας για να αποθηκεύσουμε στο id την τιμή που θέλει ο χρήστης.
  - ο genquad(‘’inp’’,’’id.place’’, ’’_’’,’’_’’).

- Έξοδος **S -> print (E) {P2}**
  Παρόμοια για την έξοδο χρειαζόμαστε πάλι μια τετράδα για να τυπώνουμε την μεταβλητή E.
  - ο genquad(‘out’’,’’E.place, ’’_’’,’’_’’).

Αφού τελειώσει η παραγωγή του ενδιάμεσου κώδικα παράγεται, με τη βοήθεια της συνάρτησης **intCode(written_file)**, το αρχείο **“file_with_quads.int”** , το οποίο περιέχει τις τετράδες του ενδιάμεσου κώδικα.

Παρακάτω βλέπουμε ενδεικτικό παράδειγμα τετράδων για την συνάρτηση factorial όπως φαίνεται στο αρχείο file_with_quads αφού τρέξουμε τον κώδικα μας.

```
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();
```

Αρχείο factorial.cpy

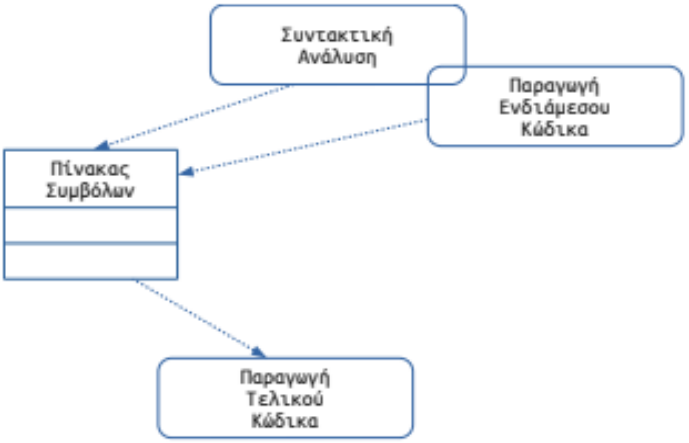
```
1:  begin_block  main_factorial  _  _
2:  inp  x  _  _
3:  =  1  _  fact
4:  =  1  _  i
5:  <=  i  x  7
6:  jump  _  _  12
7:  *  fact  i  %1
8:  =  %1  _  fact
9:  +  i  1  %2
10:  =  %2  _  i
11:  jump  _  _  5
12:  out  fact  _  _
13:  end_block  main_factorial  _  _
14:  begin_block  main  _  _
15:  call  main_factorial  _  _
16:  halt  _  _  _
17:  end_block  main  _  _
```

Αρχείο που περιέχει τετράδες (file_with_quads)

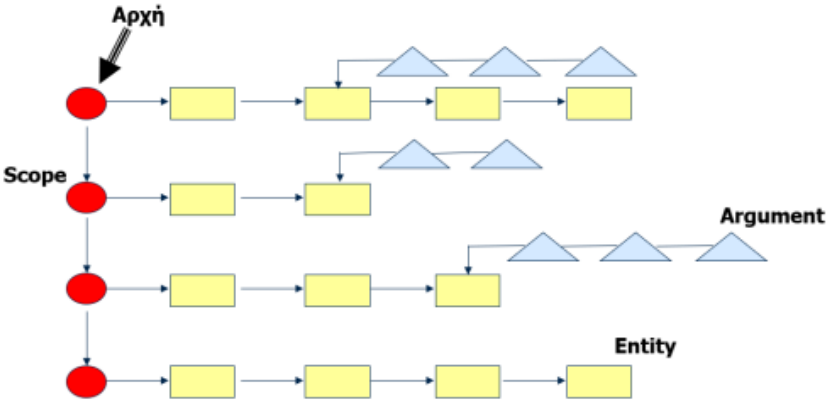
ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

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 πρέπει να έχει.

Σε ένα πίνακα συμβόλων διατηρούμε πληροφορία για τα συμβολικά ονόματα που εμφανίζονται στο πρόγραμμα. Έτσι, σε έναν πίνακα συμβόλων αποθηκεύεται πληροφορία που σχετίζεται με τις μεταβλητές του προγράμματος, τις συναρτήσεις, τις παραμέτρους και με τα ονόματα των σταθερών. Για κάθε ένα από αυτά υπάρχει διαφορετική εγγραφή στον πίνακα και στην εγγραφή αυτή αποθηκεύεται διαφορετική πληροφορία, ανάλογα με το είδος του συμβολικού ονόματος. Η πληροφορία αυτή είναι χρήσιμη για έλεγχο σφαλμάτων, αλλά είναι διαθέσιμη να ανακτηθεί κατά τη φάση της παραγωγής του τελικού κώδικα.



Ο πίνακας συμβόλων αντλεί πληροφορία από τις φάσεις της συντακτικής ανάλυσης και της παραγωγής ενδιάμεσου κώδικα και διαθέτει την πληροφορία που έχει συλλέξει για την παραγωγή τελικού κώδικα.



Στον πίνακα συμβόλων για κάθε *μεταβλητή*, *συνάρτηση*, *παράμετρο(in)* και *προσωρινή μεταβλητή* δημιουργούνται οντότητες (**Entities**). Κάθε εγγραφή στον πίνακα συμβόλων αποτελεί ένα Entity και χαρακτηρίζεται από το επίπεδο (**Scope**) στο οποίο το συναντάμε. Για κάθε συνάρτηση αποθηκεύω το όνομα των παραμέτρων τους και τον τρόπο περάσματος (**Argument**). Παρακάτω απεικονίζεται η μορφή του πίνακα συμβόλων.

- `name`: το όνομα της μεταβλητής
- `datatype`: ο τύπος δεδομένων της μεταβλητής
- `offset`: η απόσταση της μεταβλητής από την αρχή του εγγραφήματος δραστηριοποίησης

Για την υλοποίηση του πίνακα συμβόλων δημιουργούμε τις παρακάτω κλάσεις :

- **class Entity():** η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων κάθε νέα οντότητα (**Entity**) και έχει έξι πεδία. Αρχικά έχουμε το name, το όνομα της οντότητας, το οποίο αρχικοποιείται όταν ο συντακτικός αναλυτής αναγνωρίσει οντότητα. Στην συνέχεια έχουμε το type, τον τύπο της οντότητας, η οποία μπορεί να είναι ή Μεταβλητή, ή Συνάρτηση, ή Παράμετρος, ή Προσωρινή Μεταβλητή.

```
class Entity():
    def __init__(self):
        self.name = ''
        self.type = ''
        self.Metavliti = self.Metavliti()
        self.Synartisi = self.Synartisi()
        self.Parametros = self.Parametros()
        self.ProswriniMetavliti = self.ProswriniMetavliti()
```

Επομένως η κλάση **Entity** περιέχει τις κλάσεις:

- Metavliti
- Synartisi
- Parametros
- ProswriniMetavliti

➢ **class Metavliti:** η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων τις μεταβλητές του προγράμματος εισόδου και έχει ένα πεδίο, το offset, δηλαδή την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης.

```
class Metavliti:
    def __init__(self):
        self.offset = 0   # Apostash apo thn arxh tou eggrafhmatos drasthriopoihshs
```

➢ **class Synartisi**: η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων τις συναρτήσεις του προγράμματος εισόδου και έχει τέσσερα πεδία. Αρχικά έχουμε τη startQuad, την ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης. Έπειτα αποθηκεύουμε τη list_arguments (μια λίστα παραμέτρων), αποθηκεύουμε επίσης και το framelength της συνάρτησης, δηλαδή το μήκος του εγγραφήματος δραστηριοποίησης και τέλος το nestingLevel, ή αλλιώς βάθος φωλιάσματος.

```
class Synartisi:
    def __init__(self):
        self.startQuad = 0  # Etiketa ths prwths tetradas tou kwdika ths synarthshs
        self.list_arguments = []  # Lista parametrwn
        self.framelength = 0  # Mhkos eggrafhmatos drasthriopoihshs
        self.nestingLevel = 0
```

➢ **class Parametros**: η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων τις παραμέτρους των συναρτήσεων του προγράμματος εισόδου ως μεταβλητές στο επίπεδου της κληθείσας συνάρτησης (τρέχων scope + 1) και έχει ένα πεδίο,το offset της παραμέτρου.

```
class Parametros:
    def __init__(self):
        self.offset = 0  # Apostash apo thn koryfh ths stoivas
```

➢ **class ProswriniMetavliti:** η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων τις προσωρινές μεταβλητές του προγράμματος εισόδου έχει ένα πεδίο, το offset της προσωρινής μεταβλητής.

```
class ProswriniMetavliti:
    def __init__(self):
        self.offset = 0  # Apostash apo thn koryfh ths stoivas
```

- **class Scope()**: η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων κάθε νέο επίπεδο (**Scope**) και έχει τέσσερα πεδία. Αρχικά έχουμε το name, το όνομα του επιπέδου το οποίο το αντλούμε από το όνομα είτε του main προγράμματος, είτε της αντίστοιχης συνάρτησης. Επίσης διατηρούμε μια λίστα με όλα τα **Entities** του συγκεκριμένου επιπέδου, τη list_entities, καθώς και έναν αυξανόμενο αριθμό που αντιστοιχεί στο βάθος φωλιάσματος του επιπέδου, το nestingLevel. Τέλος, αποθηκεύουμε και το outerScope, δηλαδή το προτελευταίο στοιχείο της λίστας των scopes.

```
class Scope():
    def __init__(self):
        self.name = ''
        self.list_entities = []  # Lista apo Entities
        self.nestingLevel = 0  # Vathos fwliasmatos
        self.outerScope = None
```

- **Class Argument()**: η συγκεκριμένη κλάση προσθέτει στον πίνακα συμβόλων τις παραμέτρους (**Arguments**) των συναρτήσεων του προγράμματος εισόδου και έχει δύο πεδία. Αρχικά έχουμε όπως και πριν το name, δηλαδή το όνομα της παραμέτρου και τέλος έχουμε το type, δηλαδή τον τύπο('int') της παραμέτρου.

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

Ενέργειες στον Πίνακα Συμβόλων

- **Αναζήτηση Scope:**
  Αρχικά υλοποιούμε την συνάρτηση **get_scopes()**. Η get_scopes παίρνει τις συναρτήσεις από το αρχείο που έχουμε ανοίξει και τις βάζει σε μία λίστα. Πιο συγκεκριμένα:
  - πρώτα παίρνει μια διαδρομή αρχείου ως είσοδο και *διαβάζει τα περιεχόμενα του αρχείου*.

```
def get_scopes(file_path):  # H get_scopes pairnei tis synarthseis apo to arxeio pou exoume anoiksei kai tis vazei se mia lista
    list_of_scopes = []

    file_obj = open(file_path, "r")
    contents = file_obj.read()
    lines = contents.split("\n")
    current_scope = ""
```

- Στη συνέχεια αναζητά ορισμούς συναρτήσεων (γραμμές που ξεκινούν με "def", ή οποιοδήποτε αριθμό κενών, ή τη φράση "if __name__ == \"__main__\":") και τους εξάγει μαζί με τα σχετικά μπλοκ κώδικα.

```
for line in lines:
    if line.startswith("def "):  # otan h grammi arxizei me def
        if current_scope:
            list_of_scopes.append(current_scope)
        current_scope = line
    elif current_scope and line.startswith(" "):  # otan arxizoun me opoiodhpote arithmo kenwn
        keno_paragrafou = len(line) - len(line.lstrip())
        if keno_paragrafou > len(current_scope) - len(current_scope.lstrip()):
            current_scope = current_scope + "\n" + line.strip()
    elif line.startswith("if __name__ == \"__main__\":"):  # otan kaleitai h main, dhladh otan diavazetai h sygkekrimenh grammi
        current_scope = current_scope + "\n" + line
        next_line = lines[lines.index(line) + 1]
        if next_line.startswith("    "):
            current_scope = current_scope + "\n" + next_line
```

- Αυτά τα πεδία συναρτήσεων αποθηκεύονται σε μια λίστα που ονομάζεται list_of_scopes.

```
# prosthetw to teleutaio scope
if current_scope:
    list_of_scopes.append(current_scope)

globalScope = list_of_scopes[-1]  # to teleutaio stoixeio ths listas
outerScope = None
```

- Η συνάρτηση προσδιορίζει επίσης το *τελευταίο στοιχείο στο list_of_scopes*, και το εκχωρεί στη μεταβλητή **globalScope**. Επιπλέον, καθορίζει το *προτελευταίο στοιχείο στη list_of_scopes*, και το εκχωρεί στη μεταβλητή **outerScope**.

```
globalScope = list_of_scopes[-1]  # to teleutaio stoixeio ths listas
outerScope = None

# elegxos
if len(list_of_scopes) > 1:
    outerScope = list_of_scopes[-2]  # to proteleutaio stoixeio ths listas

file_obj.close()
return globalScope, outerScope
```

❖ **Προσθήκη νέου Scope:** όταν ξεκινάμε τη μετάφραση μιας νέας συνάρτησης.

```
def new_scope(name):  # Dhmiourgoume ena neo scope
    global globalScope

    nextScope = Scope()
    nextScope.name = name
    nextScope.outerScope = globalScope

    if (globalScope != None):
        nextScope.nestingLevel = globalScope.nestingLevel + 1
    else:
        nextScope.nestingLevel = 0

    globalScope = nextScope
```

Η συνάρτηση new_scope:

- Δηλώνει το global **globalScope** για να διασφαλίσει ότι η συνάρτηση μπορεί να έχει πρόσβαση και να τροποποιήσει το global scope.
- Δημιουργεί ένα νέο στιγμιότυπο της κλάσης Scope που ονομάζεται nextScope και εκχωρεί το παρεχόμενο όνομα στο χαρακτηριστικό του name.
- Η συνάρτηση ορίζει το outer scope του νέου εύρους εκχωρώντας το τρέχον global scope στο χαρακτηριστικό *outerScope* του nextScope.
- Για να προσδιορίσει το **nestingLevel**, η συνάρτηση ελέγχει εάν υπάρχει υπάρχον global scope. Εάν το *globalScope* δεν είναι **None**, υποδεικνύεται ότι υπάρχει outer scope, το **nesting level** του nextScope αυξάνεται κατά 1 μονάδα. Διαφορετικά, εάν δεν υπάρχει υπάρχον global scope, το **nesting level** του nextScope ορίζεται στο 0.
- Η συνάρτηση ενημερώνει το global scope εκχωρώντας το nextScope στο globalScope.

❖ **Διαγραφή Scope**: όταν τελειώνουμε τη μετάφραση μιας συνάρτησης - με τη διαγραφή διαγράφουμε την εγγραφή (record) του Scope και όλες τις λίστες, τα Entity και τα Argument που εξαρτώνται από αυτή.

```
def delete_scope():
    global globalScope

    current_scope = globalScope

    while current_scope.list_entities:
        entity = current_scope.list_entities.pop()
        del entity

    globalScope = current_scope.outerScope
```

Η συνάρτηση delete_scope:

- Δηλώνει το global **globalScope** για να διασφαλίσει ότι η συνάρτηση μπορεί να έχει πρόσβαση και να τροποποιήσει το global scope.
- Εκχωρεί την τιμή του globalScope σε μια τοπική μεταβλητή που ονομάζεται **current_scope**. Αυτό επιτρέπει στη συνάρτηση να επαναλαμβάνει και να τροποποιεί το scope χωρίς να επηρεάζει άμεσα το global scope.
- Εισάγει έναν βρόχο που συνεχίζεται όσο το **current_scope** έχει οντότητες στο χαρακτηριστικό **list_entities**. Μέσα σε κάθε επανάληψη του βρόχου, αφαιρεί το τελευταίο entity από τη **list_entities,** χρησιμοποιώντας τη μέθοδο ***pop*** και την εκχωρεί στη μεταβλητή entity.
- Μετά την αφαίρεση της οντότητας, η συνάρτηση χρησιμοποιεί τη λέξη-κλειδί ***del*** για να τη διαγράψει από τη μνήμη.
- Μόλις διαγραφούν όλες οι οντότητες εντός του τρέχοντος πεδίου, η συνάρτηση ενημερώνει το **current_scope** εκχωρώντας το current_scope.outerScope στο globalScope. Αυτό αντικαθιστά αποτελεσματικά το **current_scope** με το outer scope.

❖ **Προσθήκη νέου Entity**: στον πίνακα συμβόλων προσθέτουμε νέο Entity όταν:

➢ συναντάμε δήλωση μεταβλητής
➢ δημιουργείται νέα προσωρινή μεταβλητή
➢ συναντάμε δήλωση νέας συνάρτησης
➢ συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης

```
def new_entity(obj):  # Dhmiourgoume ena neo entity
    global globalScope

    globalScope.list_entities.append(obj)
```

Η συνάρτηση new_entity:

- Δηλώνει το global **globalScope** για να διασφαλίσει ότι η συνάρτηση μπορεί να έχει πρόσβαση και να τροποποιήσει το global scope.
- Παίρνει ένα αντικείμενο ***obj*** ως είσοδο, το οποίο αντιπροσωπεύει την οντότητα που θα δημιουργηθεί.
- Στη συνέχεια, προσαρτά το ***obj*** στο χαρακτηριστικό list_entities του global scope . Αυτό προσθέτει τη νέα οντότητα στη list_entities με την εντολή append(obj).

❖ **Προσθήκη νέου Argument**: όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης.

```
def new_argument(obj):  # Dhmiourgoume ena neo argument
    global globalScope

    x = globalScope.list_entities[-1].Synartisi

    x.list_arguments.append(obj)
```

Η συνάρτηση new_argument:

- Δηλώνει το global **globalScope** για να διασφαλίσει ότι η συνάρτηση μπορεί να έχει πρόσβαση και να τροποποιήσει το global scope.
- Παίρνει ένα αντικείμενο ***obj*** ως είσοδο, το οποίο αντιπροσωπεύει την οντότητα που θα δημιουργηθεί.
- Εκχωρεί σε μια προσωρινή μεταβλητή x το globalScope.list_entities[-1].Synartisi.
- Στη συνέχεια, προσαρτά το ***obj*** στο χαρακτηριστικό list_arguments του συγκεκριμένου εύρους της συνάρτησης. Αυτό προσθέτει τη νέα οντότητα στη list_arguments με την εντολή append(obj).

❖ **Αναζήτηση**: μπορεί να αναζητηθεί κάποιο entity με βάση το όνομά του.
Η αναζήτηση ενός entity γίνεται ξεκινώντας από την αρχή του πίνακα και την πρώτη του γραμμή. Αν δε βρεθεί *πηγαίνουμε στην επόμενη γραμμή,* έως ότου *βρεθεί το entity* ή *τελειώσουν όλα τα entities,* οπότε επιστρέφουμε και μήνυμα λάθους. Αν με το ζητούμενο όνομα υπάρχει πάνω από ένα entity τότε επιστρέφουμε το πρώτο που θα συναντήσουμε.

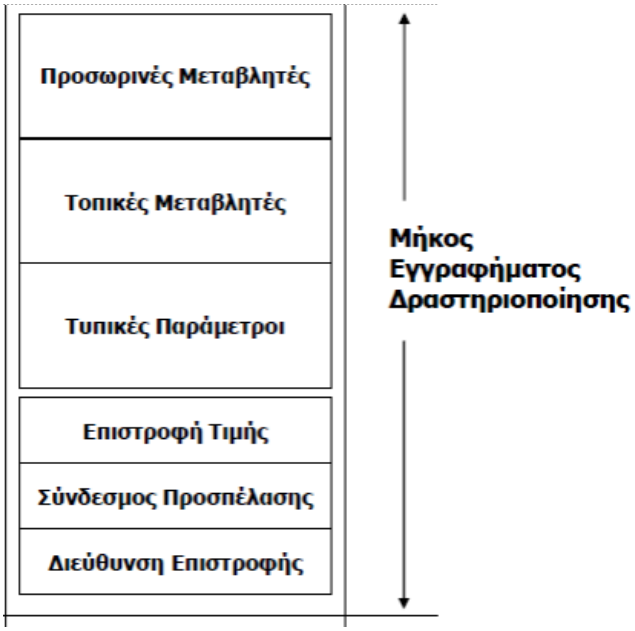
```
def search_entity(n):  # Anazhtoume ena entity ston pinaka simvolwn
    global globalScope

    current_scope = globalScope
    while current_scope != None:
        for entity in current_scope.list_entities:
            if (entity.name == n):
                return (current_scope, entity)
        current_scope = current_scope.outerScope

    print("Error: Den yparxei ston pinaka simbolon entity pou legetai " + str(n))
    exit()
```

Η συνάρτηση search_entity:

- Δηλώνει το global **globalScope** για να διασφαλίσει ότι η συνάρτηση μπορεί να έχει πρόσβαση και να τροποποιήσει το global scope.
- Παίρνει ένα όνομα **n** ως είσοδο, που αντιπροσωπεύει το όνομα της οντότητας προς αναζήτηση.
- Αρχικοποιεί μια τοπική μεταβλητή **current_scope** με την τιμή του globalScope.
- Εισέρχεται σε ένα βρόχο που συνεχίζεται όσο το **current_scope** δεν είναι **None**. Επαναλαμβάνεται μέσω της λίστας των οντοτήτων σε κάθε πεδίο.
- Μέσα σε κάθε επανάληψη του βρόχου, συγκρίνει το όνομα κάθε οντότητας με το όνομα **n**. Εάν βρεθεί μια αντιστοίχιση, επιστρέφει το **current_scope** και την αντίστοιχη οντότητα(entity).
- Εάν δεν βρεθεί αντιστοίχιση μετά τη διέλευση όλων των περιοχών, εκτυπώνει ένα μήνυμα σφάλματος που υποδεικνύει ότι η οντότητα με το δεδομένο όνομα *δεν υπάρχει στον πίνακα συμβόλων*. Στη συνέχεια, βγαίνει από το πρόγραμμα

Εγγράφημα Δραστηριοποίησης:

Κάθε πληροφορία στο εγγράφημα δραστηριοποίησής καταλαμβάνει 4 bytes.
Το εγγράφημα δραστηριοποίησης περιέχει πληροφορίες για την εκτέλεση και τον τερματισμό κάθε υποπρογράμματος καθώς και πληροφορίες που σχετίζονται με τις μεταβλητές που χρησιμοποιεί. Κάθε μεταβλητή (είτε προσωρινή είτε τοπική) και παράμετρος διατηρεί πληροφορία (offset) ώστε να τοποθετηθεί στην κατάλληλη θέση από την αρχή του εγγραφήματος δραστηριοποίησης. Όλα τα παραπάνω αποθηκεύονται 12 bytes μετά την αρχή του εγγραφήματος καθώς οι τρεις πρώτες θέσεις είναι δεσμευμένες από τα εξής : στην πρώτη θέση του εγγραφήματος δραστηριοποίησης αποθηκεύεται η διεύθυνση επιστροφής της συνάρτησης. Στη δεύτερη θέση του εγγραφήματος δραστηριοποίησης τοποθετείται ο σύνδεσμος προσπέλασης ο οποίος είναι ένα δείκτης που δείχνει στο εγγράφημα δραστηριοποίησης στο οποίο πρέπει να αναζητήσει η συνάρτηση μία μεταβλητή ή παράμετρο που δεν της ανήκει. Στην τρίτη θέση του εγγραφήματος δραστηριοποίησης δεσμεύουμε χώρο για την επιστροφή τιμής της συνάρτησης. Εκεί θα αποθηκευτεί η διεύθυνση της μεταβλητής στην οποία θα επιστραφεί η τιμή της συνάρτησης. Τέλος το μήκος εγγραφήματος δραστηριοποίησης αποτελεί το



framelength της συνάρτησης που αναφέραμε παραπάνω το οποίο υπολογίζεται προσθέτοντας στην τιμή του τελευταίου offset 4 bytes.

Επιπρόσθετες συναρτήσεις που χρησιμοποιήσαμε:

- **compute_offset()**: Η συνάρτηση είναι υπεύθυνη για τον υπολογισμό του αριθμού των byte στο τρέχον εύρος.
  - Δηλώνει το global globalScope για να διασφαλίσει ότι η συνάρτηση μπορεί να έχει πρόσβαση στο global scope.
  - Η συνάρτηση αρχικοποιεί τη μεταβλητή **offset** με τιμή 12.

```
def compute_offset():   # Ypologizoume ton arithmo twn bytes
    global globalScope

    offset = 12
    if globalScope.list_entities:
        count = 0
        for entity in globalScope.list_entities:
            if (entity.type == 'METAVLHTH' or entity.type == 'PARAMETROS'
                    or entity.type == 'PROSWRINH_METAVLHTH'):
                count = count + 1
        offset = offset + (count * 4)  # auksisi kata 4

    return offset
```

  - Εάν το global scope έχει οντότητες στο χαρακτηριστικό list_entities, η συνάρτηση εισάγει έναν βρόχο για επανάληψη μέσω κάθε οντότητας, τον count και τον αρχικοποιεί στο 0. Ελέγχει τον τύπο κάθε οντότητας και εάν ο τύπος της οντότητας ταιριάζει με έναν από τους καθορισμένους τύπους:
    - "METAVLHTH",
    - "PARAMETROS"
    - "PROSWRINH_METAVLHTH"

    τότε αυξάνει το count κατά 1.
  - Μετά την καταμέτρηση των σχετικών οντοτήτων, ενημερώνεται το offset προσθέτοντας count* 4 (κάθε οντότητα συνεισφέρει 4 byte).
  - Τέλος, επιστρέφει το offset.

- **compute_startQuad()**: Η συνάρτηση είναι υπεύθυνη για τον υπολογισμό του χαρακτηριστικού startQuad.

```
def compute_startQuad():  # Ypologizoyme to startQuad
    global globalScope

    x = globalScope.outerScope.list_entities[-1]

    x.Synartisi.startQuad = nextquad()
```

➢ **compute_framelength()**: Η συνάρτηση είναι υπεύθυνη για τον υπολογισμό του χαρακτηριστικού framelength

```
def compute_framelength():  # Ypologizoume to framelength
    global globalScope

    x = globalScope.outerScope.list_entities[-1]

    x.Synartisi.framelength = compute_offset()
```

➢ **add_Parameters()**: Η συνάρτηση δημιουργεί entities από τα arguments. Έχει πρόσβαση στο *outer scope* χρησιμοποιώντας το globalScope.outerScope και εκχωρεί τις αντίστοιχες τιμές στις οντότητες που δημιουργήθηκαν πρόσφατα, όπως

- το όνομα,
- ο τύπος ('PARAMETROS'),
- η λειτουργία ('CV')
- το offset που λαμβάνεται από τη συνάρτηση **compute_offset()**.
- αυτές οι οντότητες προστίθενται στη λίστα των οντοτήτων στο *global scope* χρησιμοποιώντας τη συνάρτηση new_entity.

```
def add_Parameters():  # Dhmioyrgoume Entities apo ta Arguments
    global globalScope

    x = globalScope.outerScope.list_entities[-1].Synartisi

    for argument in x.list_arguments:
        entity = Entity()
        entity.name = argument.name
        entity.type = 'PARAMETROS'
        entity.Parametros.mode = 'CV'
        entity.Parametros.offset = compute_offset()
        new_entity(entity)
```

➢ **print_Symbol_table(**OutputFile**)**: Η συνάρτηση είναι υπεύθυνη για την εκτύπωση του πίνακα συμβόλων σε ένα αρχείο εξόδου **“OutputFile”**. Καλείται μέσα στη def_main_function, στη def_function και στην call_main_part . Αυτή η συνάρτηση παρέχει έναν τρόπο οπτικοποίησης και ανάλυσης των περιεχομένων του πίνακα συμβόλων.

```python
def print_Symbol_table(OutputFile):  # Ektypwnoume ton pinaka symbolwn
    global globalScope

    with open(OutputFile, "a") as f:
        current_scope = globalScope
        while current_scope != None:
            f.write("----------------------------------------------------------------------------------------------------\n")
            f.write("*Scopes*\n")
            f.write("\tScope: \t" + "  Name: " + current_scope.name + "\t\tNestingLevel: " + str(current_scope.nestingLevel) + "\n")

            f.write("*Entities*\n")
            for entity in current_scope.list_entities:
                if (entity.type == 'METAVLHTH'):
                    f.write("\tEntity: " + "  Name: " + entity.name + "\t\t\tOffset: " + str(entity.Metavliti.offset) + "\n")
                elif (entity.type == 'SYNARTHSH'):
                    f.write("\tEntity: " + "  Name: " + entity.name + "\t\tStartQuad: " + str(entity.Synartisi.startQuad) +
                            "\t\tFramelength: " + str(entity.Synartisi.framelength) + "\n")
                elif (entity.type == 'PARAMETROS'):
                    f.write("\tEntity: " + "  Name: " + entity.name + "\t\t\tOffset: " + str(entity.Parametros.offset) + "\n")
                elif (entity.type == 'PROSWRINH_METAVLHTH'):
                    f.write("\tEntity: " + "  Name: " + entity.name + "\t\t\tOffset: " + str(entity.ProswriniMetavliti.offset) + "\n")

            f.write("*Arguments*\n")
            for argument in entity.Synartisi.list_arguments:
                f.write("\tArgument: " + "Name: " + argument.name + "\n")

            current_scope = current_scope.outerScope
        f.write("----------------------------------------------------------------------------------------------------\n")
        f.write("\n\n\n")
```

Ακολουθεί η ένα παράδειγμα εκτύπωσης του αρχείου **OutputFile.** Το αρχείο cpy έχει τη συνάρτηση *main_factorial()*.

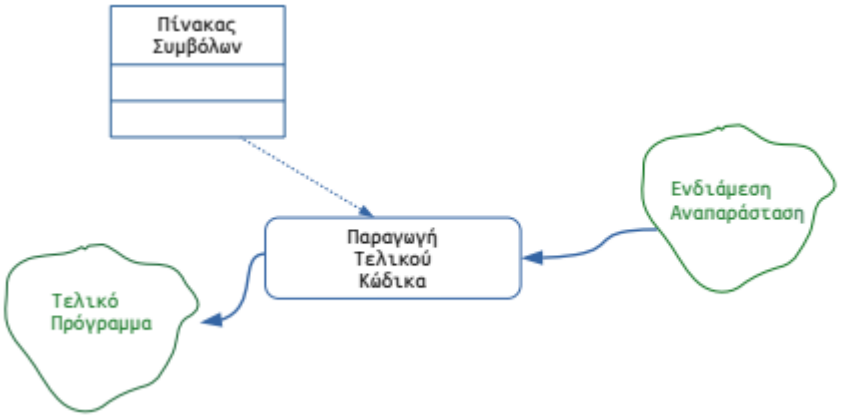
```
def main_factorial():
#{
     #$ declarations #$
     #declare x
     #declare i,fact

     #$ 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();
```

Στο αρχείο **OutputFile** έχει γραφτεί ο πίνακας συμβόλων ως εξής:

```
----------------------------------------------------------------------------------------------------
*Scopes*
        Scope:    Name: main_factorial          NestingLevel: 1
*Entities*
        Entity:   Name: x                       Offset: 12
        Entity:   Name: i                       Offset: 16
        Entity:   Name: fact                    Offset: 20
        Entity:   Name: %1                      Offset: 24
        Entity:   Name: %2                      Offset: 28
*Arguments*
----------------------------------------------------------------------------------------------------
*Scopes*
        Scope:    Name: main            NestingLevel: 0
*Entities*
        Entity:   Name: main_factorial          StartQuad: 1            Framelength: 32
*Arguments*
----------------------------------------------------------------------------------------------------




----------------------------------------------------------------------------------------------------
*Scopes*
        Scope:    Name: main            NestingLevel: 0
*Entities*
        Entity:   Name: main_factorial          StartQuad: 1            Framelength: 32
*Arguments*
----------------------------------------------------------------------------------------------------
```

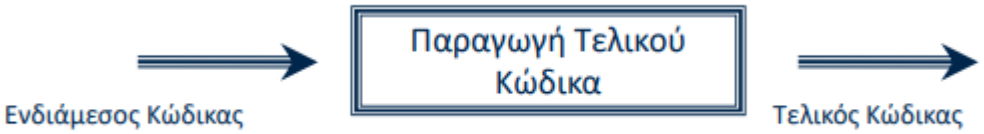
ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

Ο τελικός κώδικας δημιουργείται από τον ενδιάμεσο κώδικα, ο οποίος βασίζεται στον πίνακα συμβόλων για να βοηθήσει στη δημιουργία του. Κάθε εντολή ενδιάμεσου κώδικα δημιουργεί μια σειρά από εντολές τελικού κώδικα που αντλούν πληροφορίες από τον πίνακα συμβόλων.



Η παραγωγή τελικού κώδικα στη διαδικασία της μεταγλώττισης.

Ο τελικός κώδικας θα σχεδιαστεί βασισμένος σε όσο το δυνατόν λιγότερη εξάρτηση από το υλικό και τις ιδιαιτερότητες του **RISC-V.**



ΒΟΗΘΗΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ:

- **gnvlcode(v)**: Η συνάρτηση είναι υπεύθυνη για τη δημιουργία κώδικα για τη μεταφορά της τιμής μιας μη τοπικής μεταβλητής στον καταχωρητή **t0** (*προσωρινός καταχωρητής 0*). Λειτουργεί στο global scope που αντιπροσωπεύεται από τη μεταβλητή globalScope και εγγράφει τον κώδικα που δημιουργείται στο **finalFile**.

```python
def gnlvcode(v):  # pairnei san orisma mia metavlhth, ths opoias thn timh h th dieuthynsh theloume na prospelasoume (opou v to onoma ths metavlh
    # metaferei ston t0 th dieuthynsh mias mh topikhs metavlhths
    # apo ton pinaka symbolwn vriskei posa epipeda panw vrisketai h mh topikh metavlhth kai mesa apo to syndesmo prospelashs thn entopizei
    global globalScope
    global finalFile

    scope_pin, entity_pin = search_entity(v)  # kaloume th search_entity() gia na entopisoume th metavlhth v

    finalFile.write("\tlw t0,-4(sp)\n")  # stoiva tou gonea

    n = globalScope.nestingLevel - scope_pin.nestingLevel - 1;  # opou n ta epipeda pou prepei na anevei h gnlvcode() gia na entopisei to eggraf
    for i in range(n):
        finalFile.write("\tlw t0,-4(t0)\n")  # stoiva tou progonou pou exei th metavlhth

    # pairnw to offset apo ton pinaka simvolwn
    if entity_pin.type == 'METAVLHTH':
        finalFile.write("\taddi t0,t0,-%d\n" % (entity_pin.Metavliti.offset))  # dieuthinsi ths mh topikhs metavlhths
    elif entity_pin.type == 'PARAMETROS':
        finalFile.write("\taddi t0,t0,-%d\n" % (entity_pin.Parametros.offset))  # dieuthinsi ths mh topikhs metavlhths
```

Η συνάρτηση **gnvlcode()**:

- Η συνάρτηση λαμβάνει μια παράμετρο **v**, η οποία αντιπροσωπεύει το όνομα της μεταβλητής που θέλουμε να έχουμε πρόσβαση.
- Δηλώνει σαν global το globalScope και το finalFile για να εξασφαλίσει πρόσβαση στο global scope και στο τελικό αρχείο για τη δημιουργία κώδικα.
- Καλεί τη συνάρτηση **search_entity**, περνώντας το όνομα της μεταβλητής v, για να βρει την αντίστοιχη οντότητα στον πίνακα συμβόλων. Αυτό επιστρέφει το εύρος (scope_pin) και την οντότητα (entity_pin) όπου βρίσκεται η μεταβλητή.
- Υπολογίζει το **nesting level** μεταξύ του τρέχοντος εύρους (globalScope.nestingLevel) και του εύρους όπου ορίζεται η μεταβλητή (scope_pin.nestingLevel)-1. Αυτή η διαφορά, που συμβολίζεται ως n, αντιπροσωπεύει τον αριθμό των επιπέδων πάνω από την ιεραρχία εμβέλειας που πρέπει να διασχίσει ο κώδικας για να αποκτήσει πρόσβαση στη μη τοπική μεταβλητή.
- Γράφει κώδικα assembly για να *φορτώσει* την τιμή της μη τοπικής μεταβλητής στον καταχωρητή **t0**. Ξεκινά φορτώνοντας την τιμή του δείκτη γονικού πλαισίου από τη στοίβα **(lw t0,-4(sp))**.
- Σε έναν βρόχο που επαναλαμβάνει **n** φορές, φορτώνει την τιμή του γονέα από τη μνήμη χρησιμοποιώντας το **offset -4(t0)**. Αυτό διασχίζει αποτελεσματικά την αλυσίδα των γονικών πλαισίων μέχρι να φτάσει στο κατάλληλο επίπεδο ένθεσης όπου ορίζεται η μη τοπική μεταβλητή.
- Εάν ο τύπος οντότητας είναι 'METAVLHTH', η μετατόπιση λαμβάνεται από entity_pin.Metavliti.offset. Εάν ο τύπος οντότητας είναι "PARAMETROS", η μετατόπιση λαμβάνεται από το entity_pin.Parametros.offset.

- ➢ Γράφει κώδικα **assembly** για να αφαιρέσει τη μετατόπιση (**y**) από τον καταχωρητή **t0 (addi t0,t0,-%d),** υπολογίζοντας ουσιαστικά τη διεύθυνση της μη τοπικής μεταβλητής και αποθηκεύοντάς την στον καταχωρητή t0.
- ➢

❖ **loadvr(v,r):** Η συνάρτηση δημιουργεί κώδικα **assembly** για τη φόρτωση δεδομένων σε έναν καθορισμένο καταχωρητή (r). Λειτουργεί στο *global scope* που αντιπροσωπεύεται από τη μεταβλητή globalScope και εγγράφει τον κώδικα που δημιουργείται στο **finalFile**.

- ➢ Η συνάρτηση **loadvr(v,r)**:
- ➢ Παίρνει δύο παραμέτρους:
  - v, που αντιπροσωπεύει τη μεταβλητή που θα φορτωθεί και
  - r, που υποδεικνύει τον καταχωρητή όπου πρέπει να αποθηκευτούν τα δεδομένα.

```
def loadvr(v, r):  # metafora dedomenwn aton kataxwrhth r
    # h metafora mporei na ginei apo th mnhmh (stoiva) h na ekxvrhthei sto r mia stathera
    global globalScope
    global finalFile
```

- ➢ Δηλώνει ως global μεταβλητές το globalScope και το finalFile για να εξασφαλίσει πρόσβαση στο global scope και στο τελικό αρχείο για τη δημιουργία κώδικα.
- ➢ Ελέγχει εάν η τιμή (**v**) είναι ψηφίο χρησιμοποιώντας τη συνάρτηση **isdigit()**. Εάν το v είναι ένα ψηφίο, αντιμετωπίζεται ως σταθερή τιμή και η συνάρτηση γράφει κώδικα assembly για να φορτώσει την άμεση τιμή στον καταχωρητή **t[r]** χρησιμοποιώντας την εντολή **li**.
- ➢

```
if v.isdigit():  # an v einai stathera
    finalFile.write('li t%d,%s\n' % (r, v))
else:  # an v einai metavlhth
    scope_pin, entity_pin = search_entity(v)
```

- ➢ Εάν το **v** δεν είναι ψηφίο, αντιμετωπίζεται ως μεταβλητή. Η συνάρτηση καλεί τη συνάρτηση **search_entity**, περνώντας το όνομα της μεταβλητής (**v**), για να βρει την αντίστοιχη οντότητα στον πίνακα συμβόλων. Αυτό επιστρέφει το εύρος (**scope_pin**) και την οντότητα (**entity_pin**) όπου βρίσκεται η μεταβλητή.
- ➢ Ελέγχει το **nesting level** του εύρους όπου ορίζεται η μεταβλητή (scope_pin.nestingLevel). Εάν το **nesting level** είναι 0, δηλαδή η μεταβλητή βρίσκεται στο κύριο πρόγραμμα, η συνάρτηση ελέγχει τον τύπο της οντότητας (entity_pin.type). Εάν ο τύπος οντότητας είναι 'METAVLHTH', η συνάρτηση γράφει κώδικα **assembly** για να φορτώσει την τιμή από τη θέση μνήμης global pointer (**gp**) με βάση τη μετατόπιση (entity_pin.Metavliti.offset). Εάν ο τύπος οντότητας είναι

"PROSWRINH_METAVLHTH", η συνάρτηση φορτώνει την τιμή από τη θέση μνήμης global pointer (**gp**) με βάση τη μετατόπιση (entity_pin.ProswriniMetavliti.offset).

- Εάν το **nesting level** είναι το ίδιο με το global scope **nesting level** (scope_pin.nestingLevel == globalScope.nestingLevel), δηλαδή η μεταβλητή είναι μια τοπική μεταβλητή ή μια παράμετρος στο τρέχον εύρος, η συνάρτηση ελέγχει τον τύπο της οντότητας (entity_pin .type). Εάν ο τύπος οντότητας είναι¨:
  - 'METAVLHTH',
  - 'PARAMETROS'
  - ή 'PROSWRINH_METAVLHTH',
- η συνάρτηση γράφει κώδικα **assembly** για να φορτώσει την τιμή από τη θέση μνήμης του global pointer (**gp**) με βάση τη μετατόπιση (entity_pin.Metavliti.offset, entity_pin.Parametros .offset ή entity_pin.ProswriniMetavliti.offset).

```
# an h v einai katholikh metavlhth, dhladh anhkei sto kyriws programma
if scope_pin.nestingLevel == 0:  # h v anhkei sto kyriws programma ara to nestinglevel isoutai me 0
    if entity_pin.type == 'METAVLHTH':
        finalFile.write('lw t%d,-%d(gp)\n' % (r, entity_pin.Metavliti.offset))
    elif entity_pin.type == 'PROSWRINH_METAVLHTH':
        finalFile.write('lw t%d,-%d(gp)\n' % (r, entity_pin.ProswriniMetavliti.offset))

# an h v exei dhlwthei sth synarthsh pou auth th stigmh ekteleitai kai einai topikh metavlhth, h typi
elif scope_pin.nestingLevel == globalScope.nestingLevel:  # nestinglevel iso me to trexon
    if entity_pin.type == 'METAVLHTH':
        finalFile.write('lw t%d,-%d(sp)\n' % (r, entity_pin.Metavliti.offset))
    elif entity_pin.type == 'PARAMETROS':
        finalFile.write('lw t%d,-%d(sp)\n' % (r, entity_pin.Parametros.offset))
    elif entity_pin.type == 'PROSWRINH_METAVLHTH':
        finalFile.write('lw t%d,-%d(sp)\n' % (r, entity_pin.ProswriniMetavliti.offset))
```

- Εάν το **nesting level** είναι χαμηλότερο από το global scope **nesting level** (scope_pin.nestingLevel<globalScope.nestingLevel), δηλαδή η μεταβλητή είναι μια μη τοπική μεταβλητή σε ένα ancestor scope, η συνάρτηση δημιουργεί κώδικα για πρόσβαση στη μεταβλητή χρησιμοποιώντας τη συνάρτηση **gnvlcode**. Η συνάρτηση **gnvlcode** υπολογίζει τη διεύθυνση της μη τοπικής μεταβλητής και την αποθηκεύει στον καταχωρητή **t0**. Στη συνέχεια, η συνάρτηση **loadvr** γράφει κώδικα **assembly** για να φορτώσει την τιμή από τη μνήμη στη διεύθυνση του καταχωρητή t0 και την αποθηκεύει στον καταχωρητή **t[r]**.

```
# an h v exei dhlwthei se kapoio progono kai ekei einai topikh metavlhth, h
elif scope_pin.nestingLevel < globalScope.nestingLevel:  # nestinglevel mikr
    if entity_pin.type == 'METAVLHTH' or entity_pin.type == 'PARAMETROS':
        gnvlcode(v)
        finalFile.write('lw t%d,(t0)\n' % (r))
```

- **storerv(r,v):** Η συνάρτηση είναι υπεύθυνη για τη δημιουργία κώδικα για την αποθήκευση της τιμής από έναν καταχωρητή (r) σε μια καθορισμένη μεταβλητή (v). Λειτουργεί στο *global scope* που αντιπροσωπεύεται από τη μεταβλητή globalScope και εγγράφει τον κώδικα που δημιουργείται στο **finalFile**.

    - Η συνάρτηση **storerv(r,v):**
    - Δηλώνει ως global το globalScope και το finalFile για να εξασφαλίσει πρόσβαση στο global scope και στο τελικό αρχείο για τη δημιουργία κώδικα.

```
def storerv(r, v):  # metafora dedomenwn apo ton kataxwrhth r sth mnhmh (metavlhth v)

    global globalScope
    global finalFile
```

    - Στη συνέχεια, καλεί τη συνάρτηση **search_entity**, περνώντας το όνομα της μεταβλητής (**v**), για να βρει την αντίστοιχη οντότητα στον πίνακα συμβόλων. Αυτό επιστρέφει το εύρος (**scope_pin**) και την οντότητα (**entity_pin**) όπου βρίσκεται η μεταβλητή.

```
scope_pin, entity_pin = search_entity(v)
```

    - Ελέγχει το **nesting level** του πεδίου όπου ορίζεται η μεταβλητή (scope_pin.nestingLevel) . Εάν το **nesting level** είναι 0, υποδεικνύοντας ότι η μεταβλητή βρίσκεται στο κύριο πρόγραμμα, η συνάρτηση ελέγχει τον τύπο της οντότητας (entity_pin.type). Εάν ο τύπος οντότητας είναι 'METAVLHTH', η συνάρτηση γράφει κώδικα **assembly** για να αποθηκεύσει την τιμή από τον καταχωρητή **t[r]** στη θέση μνήμης global pointer (**gp**) με βάση τη μετατόπιση (entity_pin.Metavliti.offset). Εάν ο τύπος οντότητας είναι "PROSWRINH_METAVLHTH", η συνάρτηση αποθηκεύει την τιμή από τον καταχωρητή **t[r]** στη θέση μνήμης global pointer (**gp**) με βάση τη μετατόπιση (entity_pin.ProswriniMetavliti.offset).

```
# an h v einai katholikh metavlhth, dhladh anhkei sto kyriws programma
if scope_pin.nestingLevel == 0:  # h v anhkei sto kyriws programma ara to nestinglevel isoutai me 0
    if entity_pin.type == 'METAVLHTH':
        finalFile.write("\tsw t%d,-%d(gp)\n" % (r, entity_pin.Metavliti.offset))
    elif entity_pin.type == 'PROSWRINH_METAVLHTH':
        finalFile.write("\tsw t%d,-%d(gp)\n" % (r, entity_pin.ProswriniMetavliti.offset))
```

    - Εάν το **nesting level** είναι το ίδιο με το global scope **nesting level** (scope_pin.nestingLevel == globalScope.nestingLevel), υποδεικνύοντας ότι η μεταβλητή είναι μια τοπική μεταβλητή ή μια παράμετρος στο τρέχον εύρος, η συνάρτηση ελέγχει τον τύπο της οντότητας (entity_pin .type). Εάν ο τύπος οντότητας είναι¨:

- 'METAVLHTH',
- 'PARAMETROS'
- ή 'PROSWRINH_METAVLHTH',
  η συνάρτηση γράφει κώδικα **assembly** για να αποθηκεύσει την τιμή από τον καταχωρητή **t[r]** στη θέση μνήμης του global pointer (**gp**) με βάση τη μετατόπιση (entity_pin.Metavliti.offset, entity_pin.Parametros .offset ή entity_pin.ProswriniMetavliti.offset).

```
# an h v einai topikh metavlhth, h typikh parametros poy pernaei me timh kai vathos fwliasmatos iso me to trexon, h proswrinh metavlhth
elif scope_pin.nestingLevel == globalScope.nestingLevel:  # nestinglevel iso me to trexon
    if entity_pin.type == 'METAVLHTH':
        finalFile.write("\tsw t%d,-%d(sp)\n" % (r, entity_pin.Metavliti.offset))
    elif entity_pin.type == 'PARAMETROS':
        finalFile.write("\tsw t%d,-%d(sp)\n" % (r, entity_pin.Parametros.offset))
    elif entity_pin.type == 'PROSWRINH_METAVLHTH':
        finalFile.write("\tsw t%d,-%d(sp)\n" % (r, entity_pin.ProswriniMetavliti.offset))
```

- Εάν το **nesting level** είναι χαμηλότερο από το global scope **nesting level** (scope_pin.nestingLevel<globalScope.nestingLevel), υποδεικνύοντας ότι η μεταβλητή είναι μια μη τοπική μεταβλητή σε ένα ancestor scope, η συνάρτηση δημιουργεί κώδικα για πρόσβαση στη μεταβλητή χρησιμοποιώντας τη συνάρτηση **gnvlcode**. Η συνάρτηση **gnvlcode** υπολογίζει τη διεύθυνση της μη τοπικής μεταβλητής και την αποθηκεύει στον καταχωρητή **t0**. Στη συνέχεια, η συνάρτηση **loadvr** γράφει κώδικα **assembly** για να φορτώσει την τιμή από τη μνήμη στη διεύθυνση του καταχωρητή t0 και την αποθηκεύει στον καταχωρητή **t[r]**.

```
# an h v einai topikh metavlhth, h typikh parametros poy pernaei me timh
elif scope_pin.nestingLevel < globalScope.nestingLevel:  # nestinglevel mi
    if entity_pin.type == 'METAVLHTH' or entity_pin.type == 'PARAMETROS':
        gnvlcode(v)
        finalFile.write('sw t%d,(t0)\n' % (r))
```

ΜΕΤΑΓΛΩΤΙΣΣΗ ΤΕΤΡΑΔΩΝ

Στην συνάρτηση **finalCode()** δημιουργεί κώδικα assembly μεταφράζοντας τετράδες σε αντίστοιχες οδηγίες assembly. Χειρίζεται διαφορετικές λειτουργίες και χρησιμοποιεί βοηθητικές λειτουργίες για να διευκολύνει τη δημιουργία κώδικα.

```
def finalCode():

    global globalScope
    global all_Quads
    global finalFile
    global i
```

- Αυτό το τμήμα του κώδικα εκχωρεί μια ετικέτα σε κάθε τετράδα στη λίστα **all_Quads** και το γράφει στο finalFile.

```
for q in range(len(all_Quads)):
    finalFile.write("\nL" + str(all_Quads[q][0]) + ": \n")  # L (label)
```

- Στην περίπτωση που διαβάσουμε την λέξη "**inp**" πρέπει να φορτώσουμε στον καταχωρητή a7 την τιμή 5 και εκτελέσουμε την εντολή "**ecall**" ώστε να φορτωθεί στον καταχωρητή a0 η τιμή που θέλει ο χρήστης.

```
# eisodos dedomenwn
if (all_Quads[i][1] == 'inp'):
    finalFile.write("\tli a7,5\n")
    finalFile.write("\tecall\n")
    finalFile.write("\tmv t1,a0\n")
```

- Στην περίπτωση που διαβάσουμε την λέξη "**out**" πρέπει να φορτώσουμε στον καταχωρητή a0 την τιμή 44 και στον καταχωρητή a7 την τιμή 1. Μετά με την κλήση της **ecall** εμφανίζονται στην οθόνη τα περιεχόμενα του a0.

```
# eksodos dedomenwn
elif (all_Quads[i][1] == 'out'):
    finalFile.write("\tli a0,44\n")
    finalFile.write("\tli a7,1\n")
    finalFile.write("\tecall\n")
```

- Στην περίπτωση που διαβάσουμε την λέξη "**halt**" πρέπει να φορτώσουμε στον καταχωρητή a0 το 0 και στον καταχωρητή a7 την τιμή 93. Μετά με την κλήση της **ecall** εμφανίζονται στην οθόνη τα περιεχόμενα του a0.

```
# termatismos programmatos
elif (all_Quads[i][1] == 'halt'):
    finalFile.write("\tli a0,0\n")
    finalFile.write("\tli a7,93\n")
    finalFile.write("\tecall\n")
```

- Στην περίπτωση που διαβάσουμε την λέξη **jump** χρειάζεται να κάνουμε άλμα σε label οπότε μεταγλωττίζεται σε **b label (ισοδύναμα j)**.

```
# jump
elif all_Quads[q][1] == 'jump':
    finalFile.write("\tj L" + str(all_Quads[q][4]) + "\n")
```

- Στην περίπτωση που διαβάσουμε κάποιον relop τελεστή με x,y όρους αρχικά χρειάζεται να φορτώσουμε σε καταχωρητή τους όρους με τις εντολές loadvr(x,t1) και loadvr(y,t2). Στην συνέχεια διακρίνουμε τις εξής περιπτώσεις :
    - o **beq t1,t2,z** για το **==**
    - o **bne t1,t2,z** για το **!=**
    - o **bgt t1,t2,z** για το **>**
    - o **blt t1,t2,z** για το **<**
    - o **bge t1,t2,z** για το **>=**
    - o **ble t1,t2,z** για το **<=**

```
# relop
elif all_Quads[q][1] in ['==', '!=', '>', '<', '>=', '<=']:
    loadvr(all_Quads[q][2], 1)
    loadvr(all_Quads[q][3], 2)

    if all_Quads[q][1] == '==':
        finalFile.write("\tbeq,t1,t2,L" + str(all_Quads[q][4]) + "\n")
    elif all_Quads[q][1] == '!=':
        finalFile.write("\tbne,t1,t2,L" + str(all_Quads[q][4]) + "\n")
    elif all_Quads[q][1] == '>':
        finalFile.write("\tbgt,t1,t2,L" + str(all_Quads[q][4]) + "\n")
    elif all_Quads[q][1] == '<':
        finalFile.write("\tblt,t1,t2,L" + str(all_Quads[q][4]) + "\n")
    elif all_Quads[q][1] == '>=':
        finalFile.write("\tbge,t1,t2,L" + str(all_Quads[q][4]) + "\n")
    elif all_Quads[q][1] == '<=':
        finalFile.write("\tble,t1,t2,L" + str(all_Quads[q][4]) + "\n")
```

- Αν εντοπίσουμε τετράδα εκχώρησης (=,x,"_",z) τότε χρειάζεται να φορτώσουμε στον t1 την τιμή του x (loadvr(x,t1)) και να την αποθηκεύσουμε στον z (store(t1,z)).

```
# ekxwrhsh
elif all_Quads[q][1] == '=':
    loadvr(all_Quads[q][2], 1)
    storerv(1, all_Quads[q][4])
```

- Αν εντοπίσουμε τετράδα αριθμητικών πράξεων της μορφής op,x,y,z χρειάζεται να φορτώσουμε στους καταχωρητές t1 και t2 τις τιμές x,y αντίστοιχα (loadvr(x,t1), loadvr(y,t2)). Στην συνέχεια εκτελούμε την εντολή op t1,t1,t2 για να εκτελεστεί η αριθμητική πράξη και αποθηκεύουμε το αποτέλεσμα με την **storerv(t1,z)**. (όπου op οι αριθμητικοί τελεστές *add(πρόσθεση)*, *sub(αφαίρεση)*, *mul(πολλαπλασιασμός)*, *div(διαίρεση)*).

```
# entoles arithmhtikwn praksewn
elif all_Quads[q][1] in ['+', '-', '*', '//']:
    loadvr(all_Quads[q][2], 1)
    loadvr(all_Quads[q][3], 2)

    if all_Quads[q][1] == '+':
        finalFile.write("\tadd,t1,t1,t2\n")
    elif all_Quads[q][1] == '-':
        finalFile.write("\tsub,t1,t1,t2\n")
    elif all_Quads[q][1] == '*':
        finalFile.write("\tmul,t1,t1,t2\n")
    elif all_Quads[q][1] == '//':
        finalFile.write("\tdiv,t1,t1,t2\n")

    storerv(1, all_Quads[q][4])
```

❖ Αν διαβάσουμε την λέξη “**retv**” τότε φορτώνουμε στην μεταβλητή τα περιεχόμενα του καταχωρητή t1. Στην συνέχεια φορτώνουμε στο t0 την 3η θέση του εγγραφήματος δραστηριοποίησης με την **lw to,-8(sp)** και αποθηκεύουμε εκεί την τιμή του **t1 (sw t1,(t0))**.

```
# epistrofh timhs synarthshs
elif all_Quads[q][1] == 'retv':
    loadvr(all_Quads[q][2], 1)
    finalFile.write("\tlw t0,-8(sp)\n")
    finalFile.write("\tsw t1,(t0)\n")
    finalFile.write("\tlw ra,(sp)\n")
    finalFile.write("\tjr ra\n")
```

❖ Αν διαβάσουμε την λέξη “**par**”, διακρίνουμε τις περιπτώσεις :

➢ Αν η μεταβλητή περνάει με τιμή (**par,x,CV,_**) : αρχικά φορτώνουμε την τιμή της στον t0 με την εντολή loadvr(x,t0). Στην συνέχεια υπολογίζουμε την θέση της μεταβλητής μέσα στην στοίβα με τον τύπο 12+4i όπου i ένας αύξων αριθμός για κάθε παράμετρο και την αποθηκεύουμε με την εντολή sw t0,-(12+4i)(fp).

```
elif all_Quads[q][1] == 'par':
    if all_Quads[q][3] == 'CV':
        loadvr(all_Quads[q][2], 0)
        finalFile.write("\tsw t0,-%d(fp)\n" % (12 + 4 * i))
        i = i + 1
```

➢ Αν έχουμε τετράδα για μεταβλητή στην οποία επιστρέφεται τιμή (par,x,RET,_) τότε γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της
προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή :
addit0,sp,-offset
swt0,-8(fp)

```
elif all_Quads[q][3] == 'RET':
    scope_pin, entity_pin = search_entity(all_Quads[q][2])
    finalFile.write("\taddi t0,sp,-%d\n" % (entity_pin.ProswriniMetavliti.offset))
    finalFile.write("\tsw t0,-8(fp)\n")
```

❖ Αν διαβάσουμε την λέξη “**call**” διακρίνουμε τις περιπτώσεις :

➢ αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα:
lw t0, 4(sp)
sw t0, 4(fp)

➢ αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας:
sw sp,-4(fp)

```
# klhsh synarthshs
elif all_Quads[q][1] == 'call':
    i = -1
    scope_pin, entity_pin = search_entity(all_Quads[q][2])

    # an h kalousa kai h klhtheisa synarthsh exoun to idio nestinglevel, tote exoun ton idio gonea
    if globalScope.nestingLevel == entity_pin.Synartisi.nestingLevel:
        finalFile.write("\tlw t0,-4(sp)\n")
        finalFile.write("\tsw t0,-4(fp)\n")
    # an h kalousa kai h klhtheisa synarthsh exoun diaforetiko nestinglevel, tote h kalousa einai o goneas ths klhtheisas
    elif globalScope.nestingLevel < entity_pin.Synartisi.nestingLevel:
        finalFile.write("\tsw sp,-4(fp)\n")
```

Στην συνέχεια για όλες τις περιπτώσεις ακολουθούμε την εξής διαδικασία:

- μεταφέρουμε τον δείκτη στοίβας στην κληθείσα (addi sp,sp,framelength) ,
- καλούμε την συνάρτηση (jal f) και όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα (addi sp,sp,-framelength).

```
finalFile.write("\taddi sp,sp,%d\n" % (entity_pin.Synartisi.framelength))  # metaferoume to deikth stoivas sthn klhtheisa
finalFile.write("\tjal L%d\n" % (entity_pin.Synartisi.startQuad))  # kaloume th synarthsh
finalFile.write("\taddi sp,sp,-%d\n" % (entity_pin.Synartisi.framelength))  # otan epistrepsoume pairnoume pisw to deikth
```

❖ Αν διαβάσουμε την λέξη “**begin_block**” διακρίνουμε τις περιπτώσεις:

➢ Μέσα στην κληθείσα συνάρτηση, αρχικά στην αρχή της, αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον ra η jal (sw ra,(sp))

```
elif all_Quads[q][1] == 'begin_block':
    # mesa sthn klhtheisa sthn arxh kathe synarthshs
    if globalScope.nestingLevel != 0:
        finalFile.write("\tsw ra,(sp)\n")
```

➢ Για το κυρίως πρόγραμμα, πρέπει να δημιουργήσουμε στην αρχή του αρχείου assembly ένα άλμα (**j Lmain**), καθώς δεν είναι το πρώτο πράγμα που μεταφράζεται. Στην συνέχεια κατεβάζουμε τον καταχωρητή sp κατά το framelength της main (**addi sp,sp,framelength**) και σημειώνουμε στον gp το εγγράφημα δραστηριοποίησης της main (**move gp,sp**).

```
else:
    # arxh programmatos kai kyriws programma
    finalFile.seek(0,0)  # sthn arxh tou programmatos
    finalFile.write("\tj L%d\n" % (all_Quads[q][0]))  # xreiazetai ena alma pou na odhgei sthn prwth etiketa toy kyriws
    finalFile.seek(0,2)
    finalFile.write("\taddi sp,sp,%d\n" % (compute_offset()))  # katevazoume ton sp kata framelength ths main
    finalFile.write("\tmv gp,sp\n")  # na shmeiwsoume ston gp to eggrafhma drasthriopoihshs wste na exoume eukolh prosv
```

❖ Αν διαβάσουμε την λέξη “**end_block**”:

➢ παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της συνάρτησης και την βάζουμε πάλι στον ra. Μέσω του ra επιστρέφουμε στην καλούσα (lw ra,(sp) και jr ra).

```
# mesa sthn klhtheisa sto telos kathe synarthshs
elif all_Quads[q][1] == 'end_block':
    if globalScope.nestingLevel != 0:
        finalFile.write("\tlw ra,(sp)\n")
        finalFile.write("\tjr ra\n")
```

Στο τέλος της finalCode(), θα πρέπει να αδειάσουμε τη λίστα με τα quads, καθώς την επόμενη φορά που θα καλέσουμε τη finalCode() πρέπει να είναι άδεια από τετράδες έτσι ώστε να δημιουργηθούν νέες.

```
all_Quads = []
```

Τέλος, στο τέλος του ολοκληρωμένου μας πια κώδικα, θα προσθέσουμε τις παρακάτω εντολές:

```
file_with_quads = open('file_with_quads.int', 'w')
syntax_analitis()

print("Compilation completed successfully!!")
intCode(file_with_quads)
file_with_quads.close()
finalFile.close()
```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΟΡΘΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ ΜΕΤΑΓΛΩΤΙΣΤΗ

Έστω ότι έχουμε το παρακάτω αρχείο εισόδου 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__":
    #$ call of main functions #$
    main_fibonacci();
```

Αν δεν υπάρχουν συντακτικά και λεκτικά λάθη κατά την μεταγλώττιση του αρχείου τότε θα παραχθεί ο ενδιάμεσος κώδικας και ο πίνακας συμβόλων. Αυτές οι δύο φάσεις θα παράγουν τα αρχεία **file_with_quads.int** και **OutputFile** (όπως προαναφέραμε).

```
1:  begin_block  fibonacci  _  _
2:  <=  x  1  4
3:  jump  _  _  6
4:  retv  x  _  _
5:  jump  _  _  16
6:  -  x  1  %1
7:  par  %1  CV  _
8:  par  %2  RET  _
9:  call  fibonacci  _  _
10:  -  x  2  %3
11:  par  %3  CV  _
12:  par  %4  RET  _
13:  call  fibonacci  _  _
14:  +  %2  %4  %5
15:  retv  %5  _  _
16:  end_block  fibonacci  _  _
17:  begin_block  main_fibonacci  _  _
18:  inp  x  _  _
19:  par  x  CV  _
20:  par  %6  RET  _
21:  call  fibonacci  _  _
22:  out  %6  _  _
23:  end_block  main_fibonacci  _  _
24:  begin_block  main  _  _
25:  call  main_fibonacci  _  _
26:  halt  _  _  _
27:  end_block  main  _  _
```

file with quads.int

```
-----------------------------------------------------------------------------------------------
*Scopes*
    Scope:    Name: fibonacci       NestingLevel: 2
*Entities*
    Entity:   Name: x           Offset: 12
    Entity:   Name: %1          Offset: 16
    Entity:   Name: %2          Offset: 20
    Entity:   Name: %3          Offset: 24
    Entity:   Name: %4          Offset: 28
    Entity:   Name: %5          Offset: 32
*Arguments*
-----------------------------------------------------------------------------------------------
*Scopes*
    Scope:    Name: main_fibonacci      NestingLevel: 1
*Entities*
    Entity:   Name: x           Offset: 12
    Entity:   Name: fibonacci       StartQuad: 1        Framelength: 36
*Arguments*
    Argument: Name: x
-----------------------------------------------------------------------------------------------
*Scopes*
    Scope:    Name: main        NestingLevel: 0
*Entities*
    Entity:   Name: main_fibonacci      StartQuad: 0        Framelength: 0
*Arguments*
-----------------------------------------------------------------------------------------------


-----------------------------------------------------------------------------------------------
*Scopes*
    Scope:    Name: main_fibonacci      NestingLevel: 1
*Entities*
    Entity:   Name: x           Offset: 12
    Entity:   Name: fibonacci       StartQuad: 1        Framelength: 36
    Entity:   Name: %6          Offset: 16
*Arguments*
-----------------------------------------------------------------------------------------------
*Scopes*
    Scope:    Name: main        NestingLevel: 0
*Entities*
    Entity:   Name: main_fibonacci      StartQuad: 17       Framelength: 20
*Arguments*
-----------------------------------------------------------------------------------------------


-----------------------------------------------------------------------------------------------
*Scopes*
    Scope:    Name: main        NestingLevel: 0
*Entities*
    Entity:   Name: main_fibonacci      StartQuad: 17       Framelength: 20
*Arguments*
-----------------------------------------------------------------------------------------------
```

OutputFile

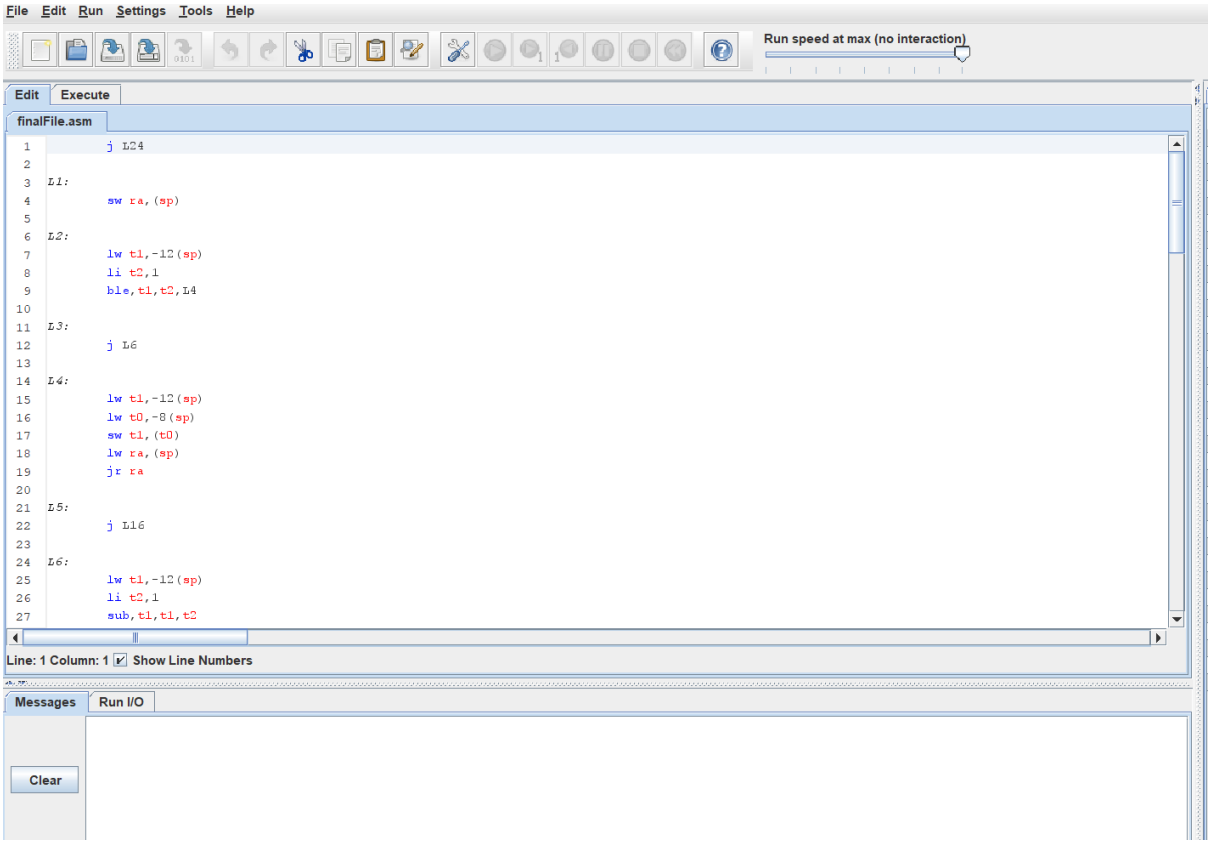
Και τέλος, στον Τελικό Κώδικα δημιουργείται ο παραγόμενος κώδικας σε assembly στο **finalFile.asm**.

```
        j L24

L1:
        sw ra,(sp)

L2:
        lw t1,-12(sp)
        li t2,1
        ble,t1,t2,L4

L3:
        j L6

L4:
        lw t1,-12(sp)
        lw t0,-8(sp)
        sw t1,(t0)
        lw ra,(sp)
        jr ra

L5:
        j L16

L6:
        lw t1,-12(sp)
        li t2,1
        sub,t1,t1,t2
        sw t1,-16(sp)

L7:
        lw t0,-16(sp)
        sw t0,-12(fp)

L8:
        addi t0,sp,-20
        sw t0,-8(fp)

L9:
        lw t0,-4(sp)
        sw t0,-4(fp)
        addi sp,sp,36
        jal L1
        addi sp,sp,-36

L10:
        lw t1,-12(sp)
        li t2,2
        sub,t1,t1,t2
        sw t1,-24(sp)

L11:
        lw t0,-24(sp)
        sw t0,-8(fp)

L12:
        addi t0,sp,-28
        sw t0,-8(fp)

L13:
        lw t0,-4(sp)
        sw t0,-4(fp)
        addi sp,sp,36
        jal L1
        addi sp,sp,-36

L14:
        lw t1,-20(sp)
        lw t2,-28(sp)
        add,t1,t1,t2
        sw t1,-32(sp)

L15:
        lw t1,-32(sp)
        lw t0,-8(sp)
        sw t1,(t0)
        lw ra,(sp)
        jr ra

L16:
        lw ra,(sp)
        jr ra

L17:
        sw ra,(sp)

L18:

L19:
        lw t0,-12(sp)
        sw t0,-8(fp)

L20:
        addi t0,sp,-16
        sw t0,-8(fp)

L21:
        sw sp,-4(fp)
        addi sp,sp,36
        jal L1
        addi sp,sp,-36

L22:

L23:
        lw ra,(sp)
        jr ra

L24:
        addi sp,sp,12
        mv gp,sp

L25:
        sw sp,-4(fp)
        addi sp,sp,20
        jal L17
        addi sp,sp,-20

L26:

L27:
```

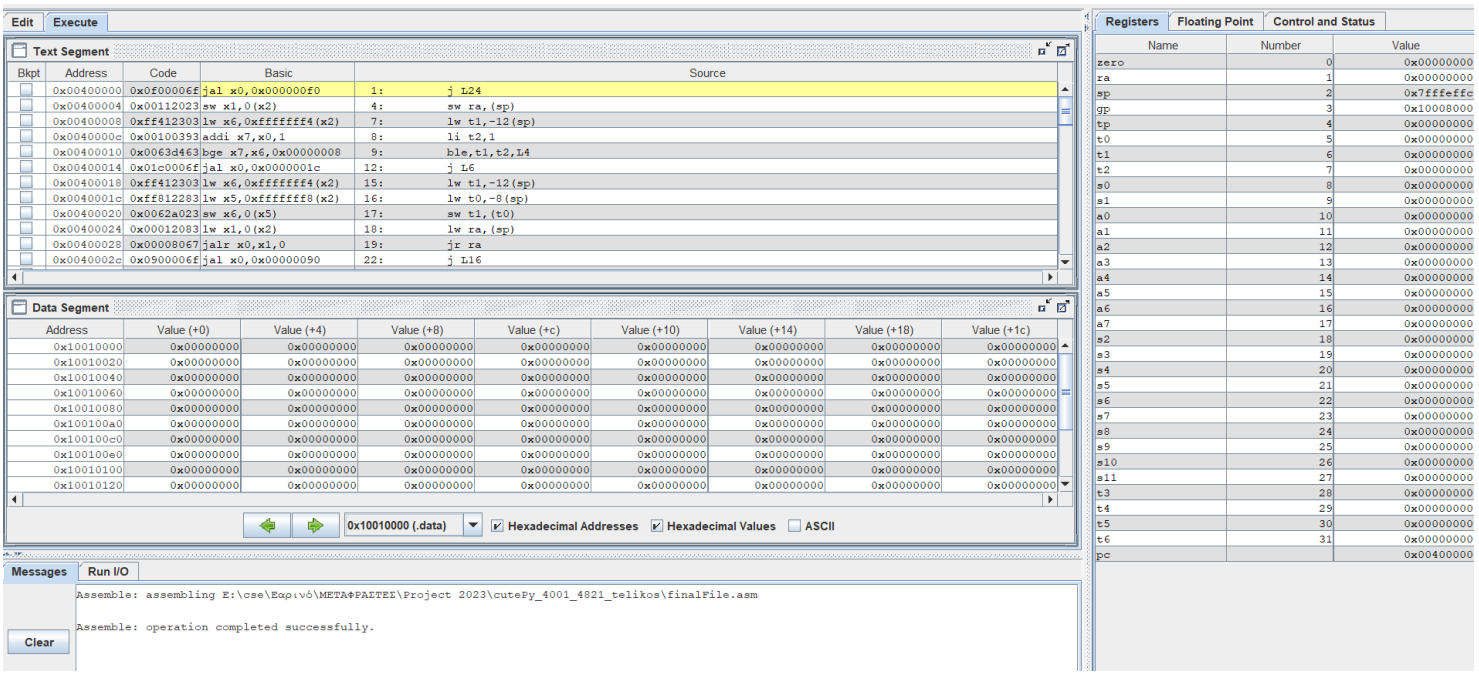
finalFile.asm

Μετά την παραγωγή του test.asm, δοκιμάζουμε να εκτελέσουμε το αρχείο στον μεταγλωττιστή του RISC-V (rars).

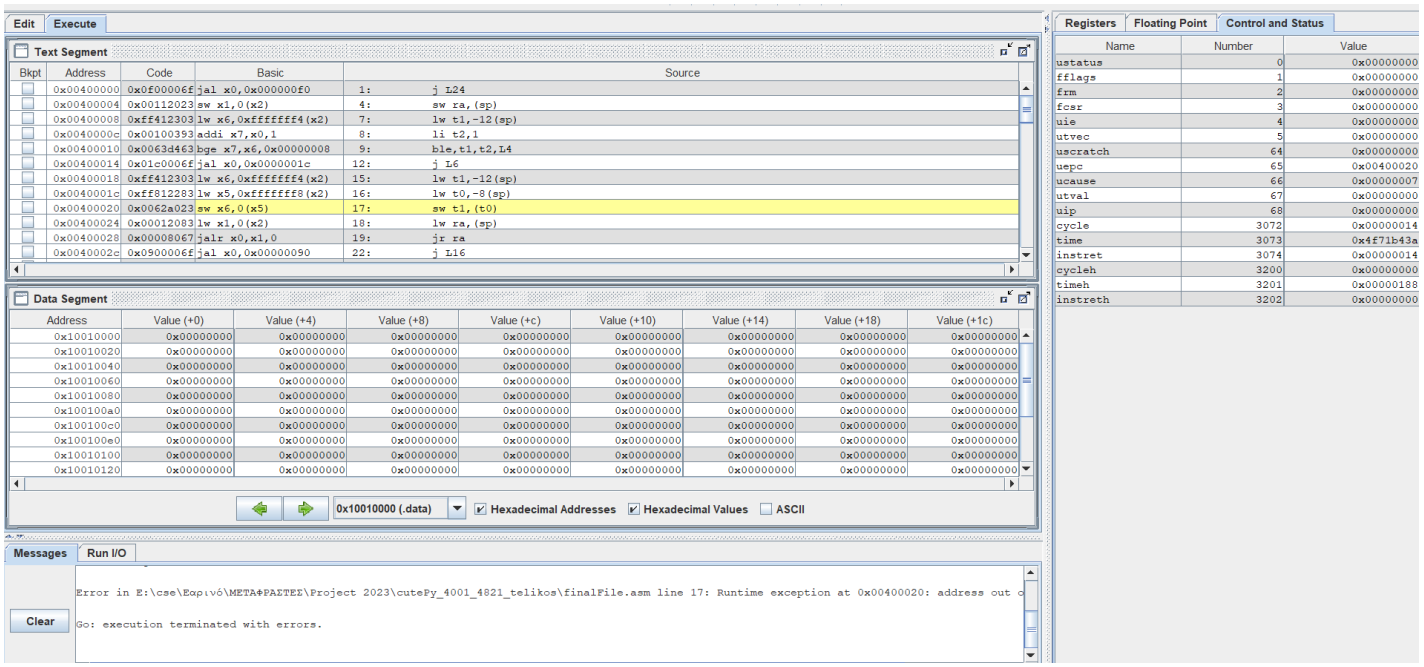
Αρχικά εκτελούμε File-> Open και ανοίγουμε το finalFile.asm.



Έπειτα, εκτελούμε Run -> Assemble.

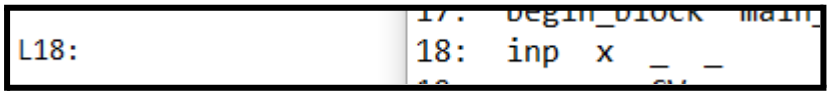


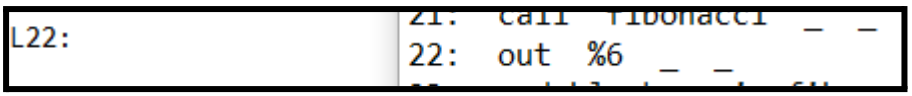
Και τέλος, εκτελούμε Run ->Go.



Σημείωση

Εάν κοιτάξουμε τα αρχεία **file_with_quads** και **finalFile** παρατηρούμε ότι κάθε label σου αρχείου assembly αντιστοιχίζεται με μία τετράδα. Για παράδειγμα το L1 αντιστοιχίζεται στην τετράδα 1. Παρόλα αυτά είναι φανερό από τις παρακάτω εικόνες οτι το L18 και το L22 δεν λειτουργούν με ορθό τρόπο.





Αυτό προφανώς έχει να κάνει με κάποια αστοχία μας πάνω στην υλοποίηση του κώδικα μας.

Ευχαριστούμε για τον χρόνο σας.