Μεταφραστές:

Προγραμματιστική Άσκηση

Ομάδα:

Δεδούση Λεμονιά 4348 (cs04348@uoi.gr)
Σιδηρόπουλος Γιώργος 4488 (cs04488@uoi.gr)

Περιεχόμενα

- Εισαγωγή ..... Σελίδα 3
  - Ανάγνωση αρχείου από την γραμμή εντολών ..... Σελίδα 3
  - Σφάλματα που εκτυπώνει το πρόγραμμα ..... Σελίδα 3
- Λεκτικός Αναλυτής (Λεκτική Ανάλυση) ..... Σελίδα 7
- Συντακτικός Αναλυτής (Συντακτική Ανάλυση) ..... Σελίδα 14
- Ενδιάμεσος κώδικας (Παραγωγή ενδιάμεσου κώδικα) ..... Σελίδα 24
  - Καθολικές Μεταβλητές ..... Σελίδα 24
  - Βοηθητικές Συναρτήσεις ..... Σελίδα 24
  - Οι Τετράδες που θα παράγονται ..... Σελίδα 27
  - Αλλαγές στον Συντακτικό Αναλυτή ..... Σελίδα 28
- Πίνακας Συμβόλων ..... Σελίδα 38
  - Κλάσεις ..... Σελίδα 38
  - Βοηθητικές Συναρτήσεις και καθολικές μεταβλητές ..... Σελίδα 41
  - Αλλαγές στον Συντακτικό Αναλυτή ..... Σελίδα 43
- Τελικός κώδικας (Παραγωγή τελικού κώδικα) ..... Σελίδα 48
  - Αλλαγές στον κώδικα του Πίνακα Συμβόλων ..... Σελίδα 48
  - Βοηθητικές Συναρτήσεις και καθολικές μεταβλητές ..... Σελίδα 49
  - Συνάρτηση παραγωγής Τελικού κώδικα ..... Σελίδα 51
  - Αλλαγές στην συνάρτηση syntaxAnalyser για την εκτύπωση όλων των αρχείων ..... Σελίδα 54

Εισαγωγή

- Στην αναφορά αυτή θα δούμε αναλυτικά τις φάσεις για τον κώδικα του μεταφραστή που δημιουργήσαμε.

- Ο κώδικας που παραδόθηκε έχει σχόλια στα αγγλικά για διευκόλυνση του αναγνώστη. Καθώς και υπάρχουν σχόλια που επιδεικνύουν τις αλλαγές μέσα στον κώδικα κατά τις αλλαγές των φάσεων μέχρι να φτάσουμε στα τελικά στάδια του κώδικά μας. Φυσικά έχουμε αναφέρει και bug-fixes που βρήκαμε και διορθώσαμε κατά την μετάβαση μεταξύ των φάσεων αυτών.

Ανάγνωση αρχείου από την γραμμή εντολών

Όσο αφορά την αρχή του προγράμματος μας, για να μπορέσουμε να καλούμε το πρόγραμμα μαζί με ένα αρχείο εισόδου από τη γραμμή εντολών, κάναμε τα εξής:

1. Αρχικά, καλέσαμε συγκεκριμένες παραμέτρους και λειτουργίες του συστήματος με την εντολή “import sys” για να μπορέσουμε να πάρουμε είσοδο από την γραμμή εντολών.

```
import sys
```

2. Μετά πήραμε το αλφαριθμητικό που δώσαμε στην είσοδο ως αρχείο (ή ως μονοπάτι του αρχείου) και το χωρίσαμε σε μία λίστα ως [όνομα, κατάληξη - τύπος αρχείου].

```
filePath = sys.argv[1]
fileNameAndType = filePath.split('.')
```

3. Τέλος, παίρνουμε την κατάληξη του αρχείου και την συγκρίνουμε με τις καταλήξεις που έχουν τα αρχεία της Cimple γλώσσας. Αν το αρχείο εν τέλη δεν είναι γραμμένο σε Cimple τότε θα μας δίνεται μήνυμα λάθους και το πρόγραμμα θα σταματά. Εδώ πρέπει να πούμε πως κατά τον τελικό κώδικα διορθώσαμε ένα bug το οποίο αν το αρχείο δεν είχε κατάληξη μας έβγαζε “IndexOutOfBounds”. Η διόρθωση έγινε με την προσθήκη ενός try-expect όπως φαίνεται παρακάτω:

```
try:
    if(fileNameAndType[1] != "ci"):
        error(-1,None)
except IndexError:
    error(-1,None)
```

Σφάλματα που εκτυπώνει το πρόγραμμα

Εδώ θα δούμε τα σφάλματα που το πρόγραμμα θα εκτυπώνει σε περίπτωση που ο χρήστης κάνει κάποιο λάθος κατά την εκτέλεση των εντολών, σε περίπτωση που γίνει κάποιο λάθος κατά την εκτέλεση του λεκτικού αναλυτή, συντακτικού αναλυτή ή κατά την εκτέλεση του πίνακα συμβόλων. Μετά την εκτύπωση κάποιου σφάλματος, το πρόγραμμα θα τερματίζει.

ERROR (-1): Wrong file types. Files must by of type '.ci' (Cimple).	Το λάθος που θα πάρουμε κατά την φόρτωση ενός πηγαίου αρχείου που δεν είναι γραμμένο σε γλώσσα Cimple. (Αρχή προγράμματος)
ERROR (0): Illegal token detected ‘...’	Το λάθος που θα πάρουμε αν βρούμε κάποιο illegalTOKEN κατά την λεκτική ανάλυση.

	(Λεκτική Ανάλυση)
ERROR (1): Keyword “program” expected in line 1. All programs should start with the keyword “program”. Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν το πηγαίο πρόγραμμα δεν ξεκινά με την λέξη “program”. (programTOKEN) (Συντακτική Ανάλυση)
ERROR (2): The name of the program expected after the keyword “program” in line 1. The illegal program name ‘...’ appeared.	Το λάθος που θα λάβουμε αν συναντήσουμε κάτι άλλο εκτός από identifierTOKEN μετά την λέξη “program” (programTOKEN) (Συντακτική Ανάλυση)
ERROR (3): Every program should end with a fullstop, fullstop at the end is missing.	Το λάθος που θα λάβουμε αν λείπει η τελεία τερματισμού που δηλώνει το τέλος του πηγαίου προγράμματος. (Συντακτική Ανάλυση)
ERROR (4): No characters are allowed after the fullstop indicating the end of the program.	Το λάθος που θα λάβουμε αν συναντήσουμε χαρακτήρες μετά την τελεία τερματισμού του πηγαίου προγράμματος. (Συντακτική Ανάλυση)
ERROR (5): Curly left bracket '{' missing at the start of the block.	Το λάθος που θα λάβουμε αν στην αρχή ενός block προγράμματος δεν βρούμε { (Συντακτική Ανάλυση)
ERROR (6): Forgot to close '{' . Instead the token ‘...’ appeared.	Το λάθος που θα λάβουμε αν δεν έχει κλείσει μία παρένθεση τύπου { (Συντακτική Ανάλυση)
ERROR (7): Missing ';' . Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν μας λείπει ; (Συντακτική Ανάλυση)
ERROR (8): Missing variable in declaration. Instead ‘...’ appeared.	Το λάθος που θα πάρουμε αν βρούμε κάτι άλλο εκτός από identifierTOKEN μέσα σε μία variable list. (Συντακτική Ανάλυση)
ERROR (9): The name of the subprogram expected. The illegal subprogram name ‘...’ appeared.	Το λάθος που θα πάρουμε αν δεν έχουμε identifierTOKEN ως όνομα του subprogram που εντοπίσαμε. (Συντακτική Ανάλυση)
ERROR (10): Round left bracket '(' missing Instead ‘...’ appeared.	Το λάθος που θα πάρουμε αν δεν έχει μπει σωστά η παρένθεση (στο πηγαίο αρχείο. (Συντακτική Ανάλυση)
ERROR (11): Forgot to close '(' . Instead the token ‘...’ appeared.	Το λάθος που θα λάβουμε αν δεν έχει κλείσει κάποια παρένθεση ((Συντακτική Ανάλυση)
ERROR(12) was deleted.	
ERROR (13): Missing relational operator between expressions. Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν μεταξύ δύο μεταβλητών ή αριθμών δεν έχει μπει κάποιο σύμβολο που υποδηλώνει τη σχέση μεταξύ τους. (Συντακτική Ανάλυση)
ERROR (14): Missing argument after 'in' or 'inout' keyword. Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν μας λείπει κάποια λέξη μετά το “in” ή το “inout”. (Συντακτική Ανάλυση)
ERROR (15): Missing assignment token ':='. Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν μας λείπει το := (Συντακτική Ανάλυση)
ERROR (16): Missing keyword inside a case statement. Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν μας λείπει κάποια λέξη μέσα στο case (Συντακτική Ανάλυση)
ERROR (17): The name of the call statement expected. The illegal subprogram name ‘...’ appeared.	ο λάθος που θα πάρουμε αν δεν έχουμε identifierTOKEN ως όνομα του call που εντοπίσαμε. (Συντακτική Ανάλυση)
ERROR (18): Missing identifier inside input expression. Instead ‘...’ appeared.	Το λάθος που θα λάβουμε αν μας λείπει κάποια λέξη μέσα στο input (Συντακτική Ανάλυση)
ERROR (19): Forgot to close '[' . Instead	Το λάθος που θα λάβουμε αν δεν έχει κλείσει

the token '...' appeared.	κάποια παρένθεση [(Συντακτική Ανάλυση)
ERROR (20): There are no more scopes to delete.	Το λάθος που θα πάρουμε αν προσπαθούμε να διαγράψουμε επίπεδο ενώ δεν έχουμε αλλά επίπεδα να διαγράψουμε. (Πίνακας συμβόλων)
ERROR (21): The entity you are trying to create is invalid.	Το λάθος που θα πάρουμε αν προσπαθήσουμε να δημιουργήσουμε κάποια νέα οντότητα από κάποια κλάση που δεν υπάρχει. (Πίνακας συμβόλων)
ERROR (22): No entity with name '...' was found in the produced symbol table.	Το λάθος που θα πάρουμε εάν κατά την αναζήτηση μίας οντότητας δεν βρεθεί η οντότητα αυτή.

Όπου "..." στον πίνακα θα είναι το token που παίρνουμε από τα arguments της συνάρτησης που δημιουργήσαμε για την εκτύπωση των σφαλμάτων αυτών.

Παρακάτω παραθέτουμε και την συνάρτηση που δημιουργήσαμε για να επιστρέφουμε τα αντίστοιχα μηνύματα σφαλμάτων και να τερματίζουμε το πρόγραμμα.

```
def error(number,token):
    if(number == -1):
        print("ERROR (-1): Wrong file types. Files must by of type '.ci' (cimple).")
    elif(number == 0):
        print("ERROR (0): Illegal token detected: '",token,"'")
    elif(number == 1):
        print("ERROR (1): Keyword “program” expected in line 1. All programs should start with
the keyword “program”. Instead, '"+token+"' appeared.")
    elif(number == 2):
        print("ERROR (2): The name of the program expected after the keyword “program” in line
1. The illegal program name '"+token+"' appeared.")
    elif(number == 3):
        print("ERROR (3): Every program should end with a fullstop, fullstop at the end is
missing.")
    elif(number == 4):
        print("ERROR (4): No characters are allowed after the fullstop indicating the end of
the program.")
    elif(number == 5):
        print("ERROR (5): Curly left bracket '{' missing at the start of the block. Instead
the token '"+token+"' appeared.")
    elif(number == 6):
        print("ERROR (6): Forgot to close '{' . Instead the token '"+token+"' appeared.")
    elif(number == 7):
        print("ERROR (7): Missing ';' . Instead '"+token+"' appeared.")
    elif(number == 8):
        print("ERROR (8): Missing variable in declaration. Instead '"+token+"' appeared.")
    elif(number == 9):
        print("ERROR (9): The name of the subprogram expected. The illegal subprogram name
'"+token+"' appeared.")
    elif(number == 10):
        print("ERROR (10): Round left bracket '(' missing Instead '"+token+"' appeared.")
    elif(number == 11):
        print("ERROR (11): Forgot to close '(' . Instead the token '"+token+"' appeared.")
    #elif(number == 12):
    elif(number == 13):
        print("ERROR (13): Missing relational operator between expressions. Instead
'"+token+"' appeared.")
    elif(number == 14):
        print("ERROR (14): Missing argument after 'in' or 'inout' keyword. Instead '"+token+"'
appeared.")
```

```
    elif(number == 15):
        print("ERROR (15): Missing assignment token ':='. Instead '"+token+"' appeared.")
    elif(number == 16):
        print("ERROR (16): Missing keyword inside a case statement. Instead '"+token+"'
appeared.")
    elif(number == 17):
        print("ERROR (17): The name of the call statement expected. The illegal subprogram
name '"+token+"' appeared.")
    elif(number == 18):
        print("ERROR (18): Missing identifier inside input expression. Instead '"+token+"'
appeared.")
    elif(number == 19):
        print("ERROR (19): Forgot to close '[' . Instead the token '"+token+"' appeared.")
    elif(number == 20):
        print("ERROR (20): There are no more scopes to delete.")
    elif(number == 21):
        print("ERROR (21): The entity you are trying to create is invalid.")
    elif(number == 22):
        print("ERROR (22): No entity with name '"+token+"' was found in the produced symbol
table.")
    exit()
```

Λεκτική Ανάλυση - Λεκτικός Αναλυτής

Η δημιουργία ενός Λεκτικού Αναλυτή είναι η πρώτη φάση της δημιουργίας ενός Μεταφραστή. Εδώ θα διαβάζουμε το πηγαίο πρόγραμμα μας χαρακτήρα - χαρακτήρα και όταν βρίσκουμε κάποια λέξη θα παράγονται κάποιες λεκτικές μονάδες μέσω ενός αυτόματου. Οι λεκτικές μονάδες αυτές θα χρησιμοποιηθούν αργότερα στην Συντακτική Ανάλυση και επομένως στον Συντακτικό Αναλυτή. (περισσότερα για αυτή τη διαδικασία θα δούμε πιο μετά στο κομμάτι της Συντακτικής Ανάλυσης)

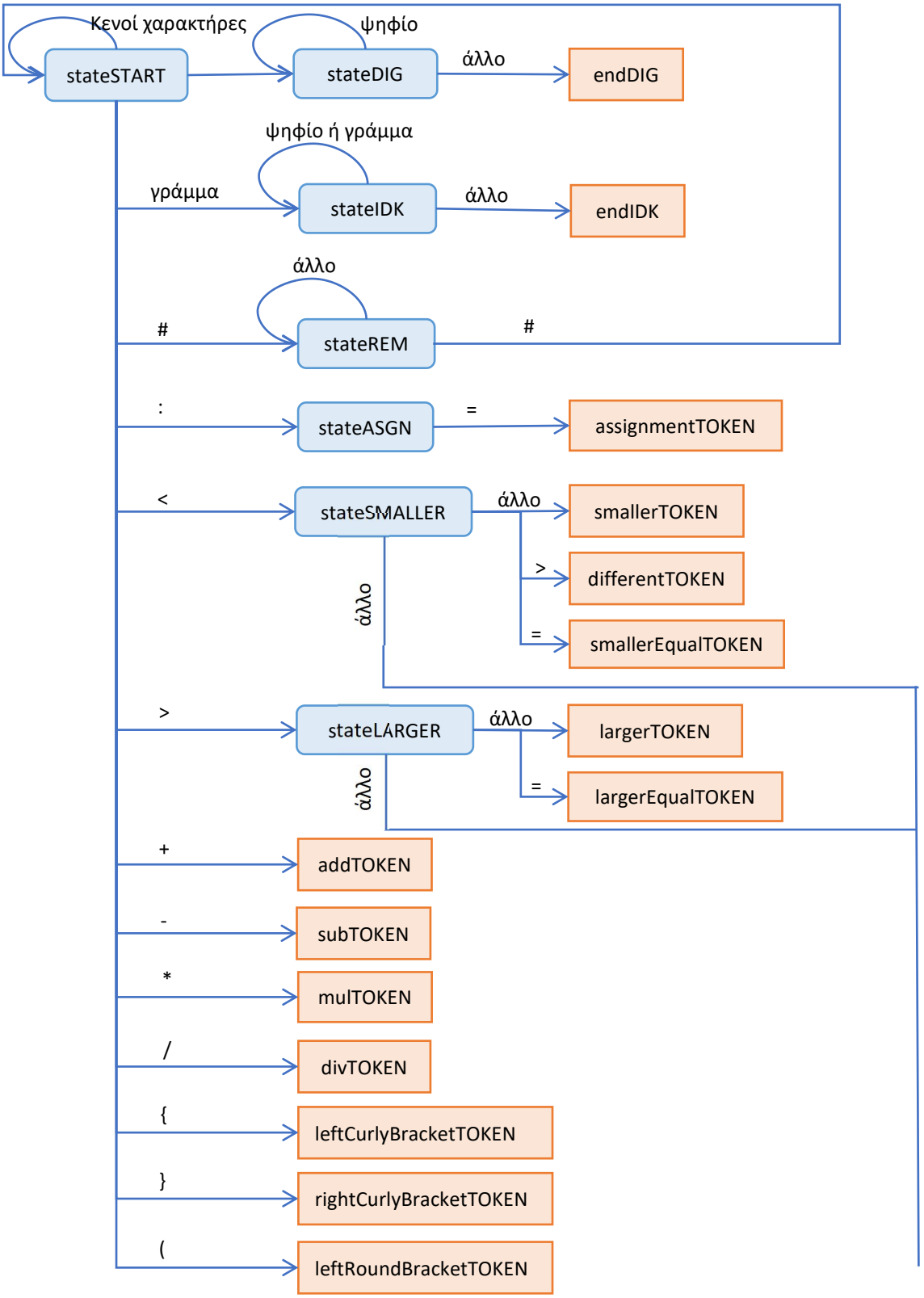
Πριν αναφέρουμε τι έγινε σε αυτό το κομμάτι από την άποψη κώδικα σε γλώσσα python, θα μιλήσουμε αρχικά για τα πιθανά αποτελέσματα (λεκτικές μονάδες) που μπορούμε να λάβουμε από τον Λεκτικό μας Αναλυτή καθώς και για το αυτόματο που χρησιμοποιήσαμε για την υλοποίηση του.

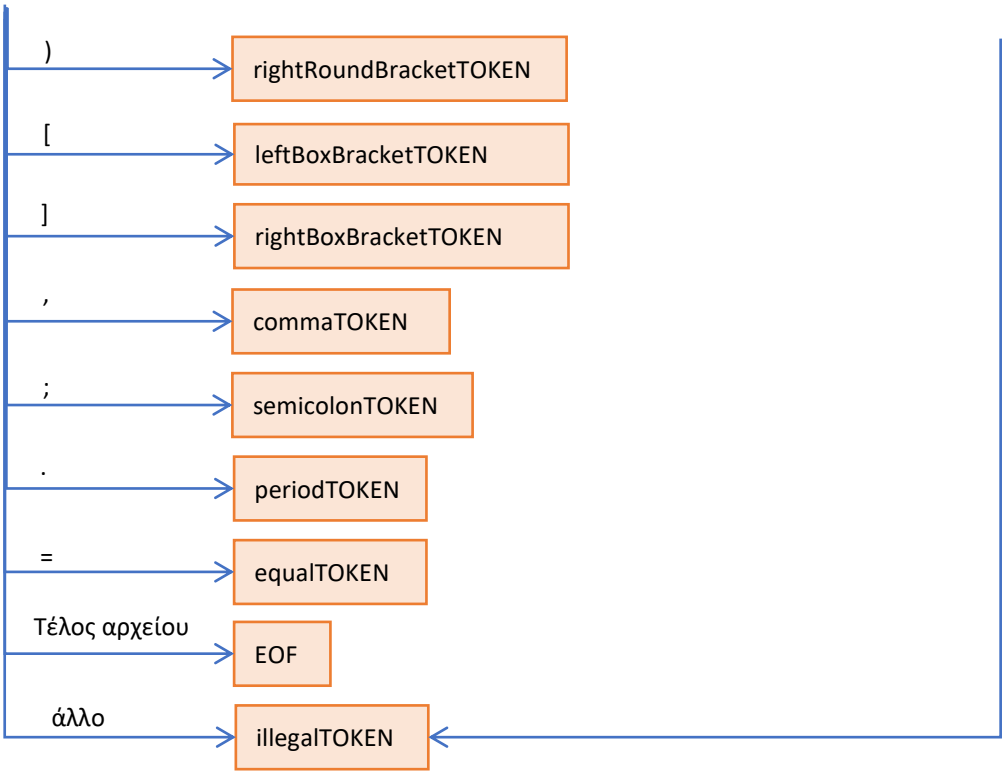
Οι λεκτικές μονάδες που θα έχει ως αποτέλεσμα ο Λεκτικός Αναλυτής μας είναι οι εξής:

illegalTOKEN	Είναι η λεκτική μονάδα που θα αντιπροσωπεύει όλες τις λέξεις που είναι “παράνομες”, δηλαδή δεν ανήκουν στην γλώσσα Cimple.
identifierTOKEN	Είναι η λεκτική μονάδα που θα αντιπροσωπεύει τα αλφαριθμητικά που θα βρίσκουμε στο πηγαίο πρόγραμμα.
numberTOKEN	Είναι η λεκτική μονάδα για τους αριθμούς που θα βρίσκουμε.
addTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το +
subTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το -
mulTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το *
divTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το /
leftCurlyBracketTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το {
rightCurlyBracketTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το }
leftRoundBracketTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το (
rightRoundBracketTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το)
leftBoxBracketTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το [
rightBoxBracketTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το]
commaTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το ,
semicolonTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το ;
periodTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το .
equalTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το =
smallerTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το <
smallerEqualTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το <=
differentTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το <>
largerTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το >
largerEqualTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το >=
assignmentTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει το :=
programTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη program
declareTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη declare

ifTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη if
elseTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη else
whileTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη while
switchcaseTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη switchcase
forcaseTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη forcase
incaseTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη incase
caseTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη case
defaultTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη default
notTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη not
andTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη and
orTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη or
functionTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη function
procedureTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη procedure
callTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη call
returnTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη return
inTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη in
inoutTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη inout
inputTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη input
printTOKEN	Η λεκτική μονάδα που θα αντιπροσωπεύει την δεσμευμένη λέξη print
EOF	Η λεκτική μονάδα που θα σημαίνει το τέλος του πηγαίου προγράμματος, δηλαδή ότι δεν υπάρχει κάτι άλλο να διαβάσουμε. Αυτή η λεκτική μονάδα θα βρίσκεται όταν φτάσει το read του αρχείου μας να διαβάζει το κενό.

Πρέπει να τονίσουμε ότι η λειτουργία του Λεκτικού Αναλυτή βασίζεται σε ένα αυτόματο, στο οποίο και θα αναφερθούμε. Το αυτόματο το οποίο δημιουργήσαμε με σκοπό να φτάσουμε στις παραπάνω λεκτικές μονάδες, ήταν βασισμένο σε αυτό που μας δόθηκε στο υλικό του μαθήματος. Αντί όμως να εξηγούμε με λόγια, ζωγραφήσαμε παρακάτω το αυτόματο με τα ονόματα των μεταβατικών καταστάσεων και των λεκτικών μονάδων να είναι αυτά που έχουμε χρησιμοποιήσει και μέσα στο πρόγραμμά μας.





Τώρα θα προχωρήσουμε στην επεξήγηση της υλοποίησης του Λεκτικού Αναλυτή μας σε γλώσσα προγραμματισμού python.
Αρχικά χρειάστηκε να ορίσουμε καθολικά (globally) δύο λίστες και μία μεταβλητή-μετρητή. Οι λίστες που δημιουργήσαμε ήταν η possibleEncounters και η keywords που περιέχουν τα αποδεκτά σύμβολα και τις δεσμευμένες λέξεις που έχει η γλώσσα Cimple αντίστοιχα. Όσο αφορά την μεταβλητή characterCounter, θα λειτουργεί ως ένας μετρητής χαρακτήρων ο οποίος θα μας βοηθά να βρούμε σε ποιον χαρακτήρα είχαμε σταματήσει κατά το διάβασμα του πηγαίου προγράμματός μας, βρίσκοντας δηλαδή κάθε φορά το από που ξεκινάμε να διαβάζουμε τη επόμενη λέξη. Αυτό σημαίνει ότι σε κάθε του κλήση, ο λεκτικός αναλυτής σημειώνει το σημείο στο οποίο σταματάει την ανάγνωση του αρχικού προγράμματος και την επόμενη φορά που καλείται συνεχίζει την ανάγνωση από το σημείο αυτό.

```
# a global counter that will help us return to where we left off at reading the file
characterCounter = 0

# an array of possible (and acceptable) encounters except from letters and numbers
possibleEncounters = ['+','-','*','/','{','}','(',')','[',']',',',';','.',':','<','>','=','#']

# an array of possible (and acceptable) preoccupied words
keywords = ["program","declare","if","else","while","switchcase","forcase","incase","case","default","not","and","or","function","procedure",
            "call","return","in","inout","input","print"]
```

Στη συνέχεια, δημιουργήσαμε και ένα λεξικό, το states, που ορίσαμε ως καθολικό (global) και το οποίο έχει ως κλειδιά τις μεταβατικές καταστάσεις του αυτόματου μας. Κάθε κατάσταση-κλειδί έχει 23 διαφορετικές τιμές που αντιπροσωπεύουν την επόμενη κίνηση του αυτόματου για κάθε πιθανή περίπτωση. Οι τιμές είναι ταξινομημένες με τέτοιο τρόπο ώστε οι θέσεις τους μέσα στην λίστα τιμών να αντιστοιχούν η κάθε μία σε μία συνάντηση με κάποιο χαρακτήρα. Πιο αναλυτικά εξηγεί ο παρακάτω πίνακας τις περιπτώσεις για κάθε θέση της λίστας τιμών των καταστάσεων - κλειδιών:

Θέση 0	Η περίπτωση που συναντάμε κενό χαρακτήρα
Θέση 1	Η περίπτωση που συναντάμε αριθμό
Θέση 2	Η περίπτωση που συναντάμε γράμμα
Θέση 3	Η περίπτωση που συναντάμε το +
Θέση 4	Η περίπτωση που συναντάμε το -

Θέση 5	Η περίπτωση που συναντάμε το *
Θέση 6	Η περίπτωση που συναντάμε το /
Θέση 7	Η περίπτωση που συναντάμε το {
Θέση 8	Η περίπτωση που συναντάμε το }
Θέση 9	Η περίπτωση που συναντάμε το (
Θέση 10	Η περίπτωση που συναντάμε το)
Θέση 11	Η περίπτωση που συναντάμε το [
Θέση 12	Η περίπτωση που συναντάμε το]
Θέση 13	Η περίπτωση που συναντάμε το ,
Θέση 14	Η περίπτωση που συναντάμε το ;
Θέση 15	Η περίπτωση που συναντάμε το .
Θέση 16	Η περίπτωση που συναντάμε το :
Θέση 17	Η περίπτωση που συναντάμε το <
Θέση 18	Η περίπτωση που συναντάμε το >
Θέση 19	Η περίπτωση που συναντάμε το =
Θέση 20	Η περίπτωση που συναντάμε το #
Θέση 21	Η περίπτωση που φθάνει σε τέλος το αρχείο μας.
Θέση 22	Η περίπτωση που συναντάμε κάτι παράνομο για την γλώσσα Cimple.

Εδώ παραθέτουμε φυσικά και το λεξικό όπως είναι μέσα στον κώδικά μας.

```
# the states of Lexical Analyser's state machine
states = {
    "stateSTART" : ["stateSTART","stateDIG","stateIDK","addTOKEN","subTOKEN","mulTOKEN","divTOKEN","leftCurlyBracketTOKEN",
                    "rightCurlyBracketTOKEN","leftRoundBracketTOKEN","rightRoundBracketTOKEN","leftBoxBracketTOKEN","rightBoxBracketTOKEN",
                    "commaTOKEN","semicolonTOKEN","periodTOKEN","stateASGN","stateSMALLER","stateLARGER","equalTOKEN","stateREM","EOF","illegalTOKEN"],
    "stateDIG" : ["endDIG","stateDIG","illegalTOKEN","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG",
                  "endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG","endDIG"],
    "stateIDK" : ["endIDK","stateIDK","stateIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK",
                  "endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK","endIDK"],
    "stateASGN" : ["illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN",
                   "illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN",
                   "illegalTOKEN","illegalTOKEN","assignmentTOKEN","illegalTOKEN","illegalTOKEN","illegalTOKEN"],
    "stateSMALLER" : ["smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN",
                      "smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN",
                      "differentTOKEN","smallerEqualTOKEN","smallerTOKEN","smallerTOKEN","smallerTOKEN"],
    "stateLARGER" : ["largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN",
                     "largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","differentTOKEN","largerTOKEN",
                     "largerEqualTOKEN","largerTOKEN","largerTOKEN","largerTOKEN","largerTOKEN"],
    "stateREM" : ["stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM",
                  "stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateREM","stateSTART","illegalTOKEN","stateREM"]
}
```

Μετά από τις αναθέσεις των καθολικών μεταβλητών μας, προχωρήσαμε στην δημιουργία της μεθόδου του Λεκτικού Αναλυτή που ονομάσαμε lexicalAnalyser(). Εδώ κινηθήκαμε ως εξής:

1. Καλέσαμε τις καθολικές μας μεταβλητές.

```
def lexicalAnalyser():
    global filePath,characterCounter
```

2. Ανοίξαμε το πηγαίο αρχείο και αρχίσαμε να διαβάζουμε από τον χαρακτήρα στη θέση characterCounter, όπου characterCounter η καθολική μεταβλητή-μετρητής.

```
    file = open(filePath, "r")                                  # open the file.
    file.read(characterCounter)
```

3. Ορίσαμε ένα κενό αλφαριθμητικό (token), το οποίο θα κρατά τη λέξη που διαβάζουμε χαρακτήρα - χαρακτήρα. Ακόμα ορίσαμε ως αρχική κατάσταση (currentState) την stateSTART του αυτόματου και το αποτέλεσμα(result) του αυτόματου ως 0 αρχικά. Όπου 0 θα είναι η θέση στην οποία, αφού έχουμε currentState = stateSTART, θα είναι το stateSTART. Αυτό έγινε με σκοπό την αρχικοποίηση της μεταβλητής result (αν δεν χρειαζόταν αρχικοποίηση της μεταβλητής δεν θα το ορίζαμε και ως μηδέν).

```
    token = ""
```

```
    currentState = "stateSTART"
    result = 0
```

4. Αμέσως μετά προχωρήσαμε στην υλοποίηση του αυτόματου. Όπου ουσιαστικά μέχρι να βρούμε κάποια ολοκληρωμένη λέξη, εναλλάσσουμε states μέσω του λεξικού ανάλογα με τους χαρακτήρες που συναντάμε. Η διαδικασία αυτή θα είναι μέσα σε έναν βρόγχο και θα τερματίζει μόνο όταν βρίσκει illegalTOKEN ή κάποια τελική κατάσταση.

Εδώ πρέπει να πούμε πως η εντολή file.read(1) πραγματοποιεί ανάγνωση του επόμενου χαρακτήρα καθώς και ότι μέσα στην περίπτωση που έχουμε σχόλια, δεν κρατάμε στο token τα σχόλια αυτά και αντί να τα διαβάζουμε λέξη - λέξη, τα διαβάζουμε ολόκληρα. Τέλος, δεν κρατάμε τους λευκούς χαρακτήρες μέσα στο token μας.

```
    while(currentState != "illegalTOKEN") or (currentState not in states.keys()):

        # read character by character.
        currentCharacter = file.read(1)                         # current character.
        characterCounter = characterCounter + 1

        # state machine.
        if(currentCharacter.isspace()):                         # if we have empty char.
            result = 0
        elif(currentCharacter.isdigit()):                       # if we have number char.
            result = 1
        elif(currentCharacter.isalpha()):                       # if we have letter char.
            result = 2
        elif(currentCharacter in possibleEncounters):           # if we have a symbol char.
            result = 3 + possibleEncounters.index(currentCharacter)
            if(currentCharacter == '#') and (currentState == "stateREM"):   # for comments.
                token = ""
        elif(currentCharacter == ""):                           # if we have EOF.
            result = 21
        else:                                                   # if we have illegal char.
            result = 22
        currentState = states[currentState][result]

        # if we stay in stateSTART, we have whitespace and we don't want to save that.
        if(currentState != "stateSTART"):
            token = token + currentCharacter

        # if we have a state that is a finishing state then we get out of the state machine.
        if(currentState not in states.keys()): break
```

5. Μετά τον τερματισμό του αυτόματου, έχουμε πλέον μία τελική κατάσταση την οποία και θα διερευνήσουμε για να καταλήξουμε στο τελικό αποτέλεσμα μας. Πιο συγκεκριμένα αν έχουμε καταλήξει σε τελική κατάσταση endDIG, endIDK, smallerTOKEN ή largerTOKEN τότε πρέπει να μεταφερθούμε ένα χαρακτήρα πίσω (αφού διαβάσαμε ένα χαρακτήρα μπροστά) και να αφαιρέσουμε και από το token μας το τελικό χαρακτήρα.

```
 if(currentState == "endDIG" or currentState == "endIDK" or currentState == "smallerTOKEN" or
currentState == "largerTOKEN"):

        token = token[:-1]
        characterCounter = characterCounter - 1
```

6. Μόλις κάνουμε αυτό, αν έχουμε endDIG, θα καταλήξουμε τότε στο numberTOKEN. Αν όμως έχουμε endIDK θα διερευνήσουμε αν η λέξη μας είναι δεσμευμένη ή όχι και θα καταλήξουμε στο ανάλογο TOKEN αν η λέξη μας είναι δεσμευμένη αλλιώς αν δεν είναι, στο identifierTOKEN.

```
if(currentState == "endDIG"):
        currentState = "numberTOKEN"
if(currentState == "endIDK"):
        if(token in keywords):
            str = token + "TOKEN"
            currentState = str
        else:
            currentState = "identifierTOKEN"
```

7. Τέλος, αν έχουμε καταλήξει σε illegalTOKEN τότε θα τερματίζουμε την λειτουργία του προγράμματος και θα εκτυπώνουμε το αντίστοιχο μήνυμα λάθους.

```
 if(currentState == "illegalTOKEN"): error(0,token)
```

8. Ο λεκτικός μας αναλυτής θα επιστρέφει το αποτέλεσμα [current state, λεκτική μονάδα].

```
    finalResult = [currentState,token]
    return finalResult
```

Συντακτική Ανάλυση - Συντακτικός Αναλυτής

Τη φάση της Λεκτικής Ανάλυσης ακολουθεί η φάση της Συντακτικής Ανάλυσης. Εδώ θα ελέγχουμε αν η ακολουθία των λεκτικών μονάδων που λαμβάνουμε από την χρήση του Λεκτικού Αναλυτή, αποτελεί μία αποδεκτή ακολουθία με βάση τη γραμματική της γλώσσας Cimple. Αν η ακολουθία που λαμβάνουμε δεν αναγνωρίζεται από τη γραμματική, τότε θα αποτελεί μη νόμιμο κώδικα και θα οδηγεί στο τερματισμό του προγράμματος και στην εκτύπωση ενός μηνύματος σφάλματος για το λάθος που έγινε. Σημαντικό δε είναι να πούμε ότι πέρα από την αναγνώριση σφαλμάτων, δίνει το περιβάλλον πάνω στο οποίο θα βασιστεί η επόμενη φάση, η παραγωγή ενδιάμεσου κώδικα, αλλά και όλη η υπόλοιπη μεταγλώττιση.
Τώρα θα προχωρήσουμε στην επεξήγηση του κώδικά μας (***μετά την πρώτη παράδοση διορθώσαμε 3 λάθη τα οποία και θα αναλύσουμε***). Στο κομμάτι αυτό θα παραθέσουμε δίπλα - δίπλα κάθε γραμματική που δημιουργήσαμε για να γίνει με επιτυχία η Συντακτική Ανάλυση μαζί με τον κώδικα που της αναλογεί.
Στον Συντακτικό Αναλυτή λοιπόν, κινηθήκαμε ως εξής:

1. Αρχικά ορίσαμε μία καθολική μεταβλητή (currentToken) ως την μεταβλητή που θα κρατά το αποτέλεσμα του Λεκτικού μας Αναλυτή κάθε φορά. Είναι σημαντικό δε να πούμε πως η μεταβλητή ορίστηκε καθολική καθώς όπως και θα δούμε στη συνέχεια μέσα στον Συντακτικό Αναλυτή θα εναλλασσόμαστε μεταξύ συναρτήσεων.

```
# a global variable for the token we are currently in.
currentToken = []
```

2. Αμέσως μετά, ορίσαμε την κεντρική (ίσως κάποιος να έλεγε και main) συνάρτηση του Συντακτικού Αναλυτή μας (**syntaxAnalyser**). Μέσα στην οποία καλούμε την currentToken μεταβλητή να λάβει την πρώτη λεκτική μονάδα από τον Λεκτικό μας Αναλυτή και να καλέσει την συνάρτηση program. Όταν η program ολοκληρωθεί δίχως κάποιο λάθος (αλλιώς θα τερμάτιζε το πρόγραμμα), θα τυπωθεί ένα μήνυμα ότι η Συντακτική Ανάλυση έχει επιτυχώς ολοκληρωθεί.

```
# Syntax Analyser
def syntaxAnalyser():
    global currentToken
    currentToken = lexicalAnalyser()
    program()
    print("Compilation successfully completed.\n")
```

3. Όσο αφορά την συνάρτηση **program**, αρχικά συγκρίναμε αν το currentToken είναι η δεσμευμένη λέξη “program” (programTOKEN). Στη συνέχεια πήραμε την αμέσως επόμενη λεκτική μονάδα η οποία θα πρέπει να είναι ID δηλαδή identifierTOKEN και καλέσαμε την συνάρτηση programBlock. Όταν η programBlock καταφέρει να ολοκληρωθεί, θα είμαστε ήδη στην αμέσως επόμενη λεκτική μονάδα την οποία και θα ελέγξουμε για το αν θα είναι “.” (periodTOKEN). Αν αυτό ισχύει, τότε θα δούμε αν το πρόγραμμα τερματίζει (EOF). Τέλος, μετά από τον τερματισμό του προγράμματος η program θα τερματίζει και αυτή. Στην περίπτωση που δεν ισχύει κάτι από τα παραπάνω το πρόγραμμα τερματίζει και δίνει μήνυμα σφάλματος καθώς θα έχει παραβιαστεί η γραμματική της γλώσσας Cimple.

```
program → program ID
          block
          .
```

```
def program():
    global currentToken,currentScope,symbolTable
    if(currentToken[0] == "programTOKEN"):
        currentToken = lexicalAnalyser()                    # move on to the next token.
        if(currentToken[0] == "identifierTOKEN"):
            currentToken = lexicalAnalyser()
            programBlock()
            if(currentToken[0] == "periodTOKEN"):
                currentToken = lexicalAnalyser()
                if(currentToken[0] == "EOF"):
                    currentToken = lexicalAnalyser()
```

```
                else:
                    error(4,currentToken[1])
            else:
                error(3,currentToken[1])
        else:
            error(2,currentToken[1])
    else:
        error(1,currentToken[1])
```

Έτσι κινηθήκαμε και στις υπόλοιπες συναρτήσεις. Γι'αυτόν τον λόγο θα αναφέρουμε τον κώδικα δίπλα από τη γραμματική που υλοποιεί χωρίς περιττά λόγια για την τετριμμένη υλοποίηση του. Σε περίπτωση που χρειαστεί παραπάνω επεξήγηση, ο κώδικας μέσα στο python αρχείο έχει κατάλληλα σχόλια που αναλυτικά εξηγούν τι γίνεται σε κάθε γραμμή.

4. **programBlock**: από την φάση 2 άλλαξε λόγο σφάλματος που έβγαζε και δεν θα καλεί πλέον την programStatements (η οποία άλλαξε όνομα σε statements κατά την δεύτερη παράδοση) αλλά την programStatementBlock. Η αλλαγή αυτή αν και όχι εμφανή στην πρώτη φάση, ήταν σημαντική για την σωστή ροή του προγράμματος μας. Η αλλαγμένη programBlock είναι η εξής:

```
block → {
            declarations
            subprograms
            blockstatements
        }
```

```
def programBlock():
    global currentToken
    if(currentToken[0] == "leftCurlyBracketTOKEN"):
        currentToken = lexicalAnalyser()
        programDeclarations()
        programSubprograms()
        programStatementBlock()
        if(currentToken[0] == "rightCurlyBracketTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(6,currentToken[1])
    else:
        error(5,currentToken[1])
```

5. **programDeclarations**: Εδώ κάνουμε αναδρομή για να αντικαταστήσουμε ουσιαστικά την Kleene Star που έχουμε παρακάτω στο σχήμα της συνάρτησης. Το ίδιο θα εφαρμόσουμε για την Kleene Star και σε παρακάτω μεθόδους.

```
declarations → ( declare varlist ; )*
```

```
def programDeclarations():
    global currentToken
    if(currentToken[0] == "declareTOKEN"):                  # if we have "declare".
        currentToken = lexicalAnalyser()
        variableList()
        if(currentToken[0] == "semicolonTOKEN"):            # if we have ; at the end.
            currentToken = lexicalAnalyser()
            programDeclarations()                           # recursion
        else:
            error(7,currentToken[1])
```

6. **variableList**:

```
varlist → ID
          ( , ID )*
        | ε
```

```
def variableList():
    global currentToken
    if(currentToken[0] == "identifierTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "commaTOKEN"):
            currentToken = lexicalAnalyser()
            variableList()                                  # recursion
    else:
```

```
        error(8,currentToken[1])
```

7. **programSubprograms**: Το συγκεκριμένο πρόγραμμά-συνάρτηση σέβεται και την περίπτωση που δεν υπάρχουν subprograms και για αυτόν τον λόγο δεν έχει κάποιο μήνυμα σφάλματος για την περίπτωση αυτή. Επίσης σέβεται την Kleene Star κάνοντας αναδρομή.

```
# zero or more subprograms
subprograms   →   ( subprogram )*


# a subprogram is a function or a procedure,
# followed by parameters and block
subprogram    →   function ID ( formalparlist )
                  block
              |   procedure ID ( formalparlist )
                  block
```

```
global currentToken,currentScope,symbolTable
    # if we have a function or a procedure:
    if(currentToken[0] in ["functionTOKEN","procedureTOKEN"]):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "leftRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                formalParameterList()
                if(currentToken[0] == "rightRoundBracketTOKEN"):    # if the next token is ')'
                    currentToken = lexicalAnalyser()
                    programBlock()
                    programSubprograms()                        # recursion
                else:
                    error(11,currentToken[1])
            else:
                error(10,currentToken[1])
        else:
            error(9,currentToken[1])
```

8. **formalParameterList**: Εδώ πρέπει να πούμε πως κατά την φάση 2 διορθώσαμε την λειτουργία της συνάρτησης (έτσι όπως φαίνεται στην εικόνα) καθώς κατά την πρώτη παράδοση υπήρχε σφάλμα για parameter list με παραπάνω από 2 statements. Η διόρθωση έγινε με την χρήση while-loop. Η αλλαγμένη συνάρτηση είναι η εξής:

```
formalparlist →   formalparitem
                  ( , formalparitem )*
              |   ε
```

```
def formalParameterList():
    global currentToken
    formalParameterItem()
    while(currentToken[0] == "commaTOKEN"):
        currentToken = lexicalAnalyser()
        formalParameterItem()
```

9. **formalParameterItem**:

```
formalparitem →   in ID
              |   inout ID
```

```
def formalParameterItem(name):
    global currentToken
    if(currentToken[0] in ["inTOKEN","inoutTOKEN"]):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(14,currentToken[1])
```

10. **Statements (*ονομαζόμενη και ως programStatements κατά την πρώτη φάση*)**: Η συνάρτηση αυτή κατά την δεύτερη φάση άλλαξε ριζικά. Αρχικά δεν καλείται πια από την programBlock αλλά μόνο από τις συναρτήσεις (statement). Ακόμα δεν θα εκτελεί αναδρομή όπως έκανε αρχικά καθώς αυτό προκαλούσε πρόβλημα για την ροή του προγράμματος. Η αναδρομή της αντικαταστάθηκε από μία while-loop. Η αλλαγμένη και πλέον περισσότερο σωστή προσέγγιση της είναι η εξής:

```
def statements():
    global currentToken
```

```
    if(currentToken[0] == "leftCurlyBracketTOKEN"):             # if we have '{'
        currentToken = lexicalAnalyser()
        statement()
        while(currentToken[0] == "semicolonTOKEN") or (currentToken[1] in keywords):
            if(currentToken[0]=="semicolonTOKEN"):
                currentToken = lexicalAnalyser()
            statement()
        if(currentToken[0] == "rightCurlyBracketTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(6,currentToken[1])
    else:
        statement()
        if(currentToken[0] == "semicolonTOKEN"):
            currentToken = lexicalAnalyser()
```

```
statements  →  statement ;
            |  {
                   statement
                   ( ; statement )*
               }
```

11. **programStatementBlock**:

```
blockstatements→  statement
                  ( ; statement )*
```

```
def programStatementBlock():
    global currentToken
    statement()
    if(currentToken[0] == "semicolonTOKEN"):
        currentToken = lexicalAnalyser()
        programStatementBlock()
```

12. Στην συνάρτηση **statement**, ανάλογα με το τι είναι η τωρινή μας λεκτική μονάδα, θα καλούμε και διαφορετική συνάρτηση, όπως φαίνεται και παρακάτω.

```
def statement():
    global currentToken

    if(currentToken[0] == "identifierTOKEN"):
        currentToken = lexicalAnalyser()
        assignStatement()
    elif(currentToken[0] == "ifTOKEN"):
        currentToken = lexicalAnalyser()
        ifStatement()
    elif(currentToken[0] == "whileTOKEN"):
        currentToken = lexicalAnalyser()
        whileStatement()
    elif(currentToken[0] == "switchcaseTOKEN"):
        currentToken = lexicalAnalyser()
        switchcaseStatement()
    elif(currentToken[0] == "forcaseTOKEN"):
        currentToken = lexicalAnalyser()
        forcaseStatement()
    elif(currentToken[0] == "incaseTOKEN"):
        currentToken = lexicalAnalyser()
        incaseStatement()
    elif(currentToken[0] == "callTOKEN"):
        currentToken = lexicalAnalyser()
        callStatement()
    elif(currentToken[0] == "returnTOKEN"):
        currentToken = lexicalAnalyser()
        returnPrintStatement()
    elif(currentToken[0] == "inputTOKEN"):
        currentToken = lexicalAnalyser()
        inputStatement()
    elif(currentToken[0] == "printTOKEN"):
```

```
statement  →  assignStat
           |  ifStat
           |  whileStat
           |  switchcaseStat
           |  forcaseStat
           |  incaseStat
           |  callStat
           |  returnStat
           |  inputStat
           |  printStat
           |  ε
```

```
        currentToken = lexicalAnalyser()
        returnPrintStatement()
```

13. **assisgnStatement**:

```
def assignStatement():
    global currentToken
    if(currentToken[0] == "assignmentTOKEN"):
        currentToken = lexicalAnalyser()
        expression()
    else:
        error(15,currentToken[1])
```

```
assignStat  →  ID := expression
```

14. **ifStatement**:

```
def ifStatement():
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        condition()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            statements()
            elsePart()
        else:
            error(11,currentToken[1])
    else:
        error(10,currentToken[1])
```

```
ifStat  →  if ( condition )
               statements
           elsepart
```

15. **elsePart**: στην γραμματική της γλώσσας Cimple δεν είναι απαραίτητο να υπάρχει το κομμάτι του “else” για να είναι σωστή μία if-statement οπότε και δεν θα έχουμε ανάλογο μήνυμα σφάλματος.

```
def elsePart():
    global currentToken
    if(currentToken[0] == "elseTOKEN"):
        currentToken = lexicalAnalyser()
        statements()
```

```
elsepart  →  else
                 statements
          |  ε
```

16. **whileStatement**:

```
def whileStatement():
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        condition()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            statements()
        else:
            error(11,currentToken[1])
    else:
        error(10,currentToken[1])
```

```
whileStat  →  while ( condition
                  statements
```

17. Όσο αφορά την συνάρτηση **caseStatement** που υπήρχε κατά την πρώτη παράδοση και ένωνε ουσιαστικά τις forcase και switchcase λόγο ομοιοτήτων στον κώδικα τους, αναγκαστήκαμε να την “σπάσουμε”, δηλαδή να τις διαχωρίσουμε σε forcase και switchcase. Αυτό έγινε λόγο διαφορών στον ενδιάμεσο κώδικα, κάτι που θα δούμε και αργότερα.

a) **forcaseStatement**:

```
# forcase statement
forcaseStat    →    forcase
                        ( case ( condition ) statements )*
                        default statements
```

```
def forcaseStatement():
    global currentToken
    if(currentToken[0] == "caseTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "leftRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            condition()
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                statements()
                if(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
                    forcaseStatement()
            else:
                error(11,currentToken[1])
        else:
            error(10,currentToken[1])
    elif(currentToken[0] == "defaultTOKEN"):
        currentToken = lexicalAnalyser()
        statements()
    else:
        error(16,currentToken[1])
```

b) **switchcaseStatement**:

```
# switch statement
switchcaseStat →   switchcase
                        ( case ( condition ) statements )*
                        default statements
```

```
def switchcaseStatement():
    global currentToken
    if(currentToken[0] == "caseTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "leftRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            condition()
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                statements()
                if(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
                    switchcaseStatement()
            else:
                error(11,currentToken[1])
        else:
            error(10,currentToken[1])
    elif(currentToken[0] == "defaultTOKEN"):
        currentToken = lexicalAnalyser()
        statements()
    else:
        error(16,currentToken[1])
```

18. **incaseStatement**:

```
incaseStat    →    incase
                        ( case ( condition ) statements )*
```

```
def incaseStatement():
    global currentToken
    while(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
```

```
        if(currentToken[0] == "caseTOKEN"):
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "leftRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                condition()
                if(currentToken[0] == "rightRoundBracketTOKEN"):
                    currentToken = lexicalAnalyser()
                    statements()
                else:
                    error(11,currentToken[1])
            else:
                error(10,currentToken[1])
        elif(currentToken[0] == "defaultTOKEN"):
            currentToken = lexicalAnalyser()
            statements()
        else:
            error(16,currentToken[1])
```

19. Όσο αφορά την συνάρτηση **returnPrintStatement**, εδώ ενώσαμε τις συναρτήσεις returnStat και printStat που ουσιαστικά έχουν την ίδια δομή.

```
def returnPrintStatement():
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        expression()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(11,currentToken[1])
    else:
        error(10,currentToken[1])
```

```
# return statement
returnStat    →    return( expression )
```

```
# print statement
printStat     →    print( expression )
```

20. **callStatement**:

```
def callStatement():
    global currentToken
    if(currentToken[0] == "identifierTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "leftRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            actualParameterList()
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
            else:
                error(11,currentToken[1])
        else:
            error(10,currentToken[1])
    else:
        error(17,currentToken[1])
```

```
# call statement
callStat      →    call ID( actualparlist )
```

21. **inputStatement**:

```
def inputStatement():
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
```

```
# input statement
inputStat     →    input( ID )
```

```
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
            else:
                error(11,currentToken[1])
        else:
            error(18,currentToken[1])
    else:
        error(10,currentToken[1])
```

22. **actualParameterList**: Εδώ πρέπει να πούμε πως κατά την φάση 2 διορθώσαμε την λειτουργία της συνάρτησης (έτσι όπως φαίνεται παρακάτω) καθώς κατά την πρώτη παράδοση υπήρχε σφάλμα για parameter list με παραπάνω από 2 statements. Η διόρθωση έγινε με την χρήση while-loop.

```
def actualParameterList():
    global currentToken
    actualParameterItem()
    while(currentToken[0] == "commaTOKEN"):
        currentToken = lexicalAnalyser()
        actualParameterItem()
```

```
# list of actual parameters
actualparlist →    actualparitem
                   ( , actualparitem )*
                |  ε
```

23. **actualParameterItem**:

```
def actualParameterItem():
    global currentToken
    if(currentToken[0] == "inoutTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(14,currentToken[1])
    elif(currentToken[0] == "inTOKEN"):
        currentToken = lexicalAnalyser()
        expression()
```

```
actualparitem →    in expression
               |   inout ID
```

24. **condition**: Η αναδρομή της πρώτης παράδοσης αντικαταστάθηκε από while-loop.

```
def condition():
    global currentToken
    boolTerm()
    while(currentToken[0] == "orTOKEN"):
        currentToken = lexicalAnalyser()
        boolTerm()
```

```
condition    →    boolterm
                  ( or boolterm )*
```

25. **boolterm**: Η αναδρομή της πρώτης παράδοσης αντικαταστάθηκε από while-loop.

```
def boolTerm():
    global currentToken
    boolFactor()
    while(currentToken[0] == "andTOKEN"):
        currentToken = lexicalAnalyser()
        boolFactor()
```

```
boolterm    →    boolfactor
                 ( and boolfactor )*
```

26. **boolfactor**:

```
def boolFactor(boolFactorCondition):
    global currentToken
    if(currentToken[0] == "notTOKEN"):
        currentToken = lexicalAnalyser()
```

```
boolfactor    →    not [ condition ]
              |    [ condition ]
              |    expression REL_OP expression
```

```
    if(currentToken[0] == "leftBoxBracketTOKEN"):
        currentToken = lexicalAnalyser()
        condition()
        if(currentToken[0] == "rightBoxBracketTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(19,currentToken[1])
    else:
        expression()
        if(currentToken[0] in
["equalTOKEN","smallerEqualTOKEN","largerEqualTOKEN","smallerTOKEN","largerTOKEN","differentTO
KEN"]):
            currentToken = lexicalAnalyser()
            expression()
        else:
            error(13,currentToken[1])
```

27. **expression**: Η αναδρομή της πρώτης παράδοσης αντικαταστάθηκε από while-loop.

```
expression  →  optionalSign term
               ( ADD_OP term )*
```

```
def expression():
    global currentToken,currentScope
    optionalSign()
    term()
    while(currentToken[0] in ["addTOKEN","subTOKEN"]):
        currentToken = lexicalAnalyser()
        term()
```

28. **term**: Η αναδρομή της πρώτης παράδοσης αντικαταστάθηκε από while-loop.

```
term  →  factor
         ( MUL_OP factor )*
```

```
def term():
    global currentToken
    factor()
    while(currentToken[0] in ["mulTOKEN","divTOKEN"]):
        currentToken = lexicalAnalyser()
        factor()
```

29. **factor**:

```
factor  →  INTEGER
        |  ( expression )
        |  ID idtail
```

```
def factor():
    global currentToken
    if(currentToken[0] == "numberTOKEN"):
        currentToken = lexicalAnalyser()
    elif(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        expression()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(11,currentToken[1])
    elif(currentToken[0] == "identifierTOKEN"):
        currentToken = lexicalAnalyser()
        idTail()
```

30. **idtail**:

```
idtail  →  ( actualparlist )
        |  ε
```

```
def idTail(name):
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        actualParameterList()
```

```
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
        else:
            error(11,currentToken[1])
```

31. **optionalSign**:

optionalSign → ADD_OP | ε

```
def optionalSign():
    global currentToken
    if(currentToken[0] == "addTOKEN") or (currentToken[0] == "subTOKEN"):
        currentToken = lexicalAnalyser()
```

Τέλος, για να τρέξει αυτή η φάση ως κώδικας απλά καλέσαμε τον Συντακτικό Αναλυτή στο τέλος του προγράμματός μας ως εξής:

```
syntaxAnalyser()
```

Ενδιάμεσος κώδικας (Παραγωγή ενδιάμεσου κώδικα)

Μετά την δημιουργία των αναλυτών μας, ακολουθεί η παραγωγή του ενδιάμεσου κώδικα του μεταφραστή μας. Η παραγωγή ενδιάμεσου κώδικα σαν ενότητα λογισμικού μπορούμε να θεωρήσουμε ότι παίρνει ως είσοδο το δέντρο της συντακτικής ανάλυσης και δημιουργεί σαν αποτέλεσμα το αρχικό πρόγραμμα μεταφρασμένο σε μία ενδιάμεση γλώσσα. Η ενδιάμεση γλώσσα αυτή θα αποτελείται από μία σειρά από τετράδες που θα αποθηκεύονται σε μία κατάλληλη δομή στη μνήμη. Κάθε τετράδα θα αποτελείται από έναν τελεστή και τρία τελούμενα. Αν μετρήσουμε και την ετικέτα που θα την χαρακτηρίζει, παρατηρούμε πως η κάθε εντολή στην ενδιάμεση γλώσσα αυτή θα αποτελείται από πέντε οντότητες.

Καθολικές Μεταβλητές

Έχοντας τα προαναφερθέντα στο μυαλό μας, η δομή που δημιουργήσαμε για να αποθηκεύουμε τις πέντε αυτές οντότητες ήταν ένας καθολικός (global) πίνακας με το όνομα “quadList”. Εδώ ο πίνακας αυτός θα έχει δομή **[ετικέτα,[τελεστής, τελούμενο1, τελούμενο2, τελούμενο3]]**.

```
quadList = []
```

Οι ετικέτες μας ακόμα θα είναι αριθμοί και θα μετριούνται με την σειρά τους από ένα καθολικό μετρητή επ’ ονόματι “quadCounter”. Η πρώτη ετικέτα θα είναι το μηδέν (0) και η αρχικοποίηση του μετρητή αυτού θα είναι στο -1.

```
quadCounter = -1
```

Τέλος, αφού θα έχουμε και παραγωγή προσωρινών μεταβλητών, θα έχουμε και έναν καθολικό μετρητή για αυτές, ο οποίος θα έχει αρχικοποίηση στο μηδέν (0). Η πρώτη δε προσωρινή μεταβλητή θα είναι η “temp_0” και οι υπόλοιπες θα ακολουθούν στην ίδια μορφή.

```
tempCounter = 0
```

Βοηθητικές Συναρτήσεις

Για να γίνει η σωστή παραγωγή του ενδιάμεσου κώδικα μας, θα χρειαστούμε να φτιάξουμε και κάποιες βοηθητικές συναρτήσεις. Οι βοηθητικές συναρτήσεις που δημιουργήσαμε ήταν οι εξής:

1. **genQuad**: είναι η συνάρτηση που θα δημιουργεί καινούργιες τετράδες και θα τις εισάγει με την αντίστοιχη τους ετικέτα στον πίνακα quadList. Ακόμα ως παραμέτρους θα έχει τον τελεστή και τα τελούμενα που θα αποτελούν την τετράδα που θέλουμε να εισάγουμε κάθε φορά. Ο αριθμός της τετράδας που δημιουργείται προκύπτει αυτόματα από τον αριθμό της τελευταίας τετράδας που δημιουργήθηκε, συν ένα.

```
def genQuad(operator, operand1, operand2, operand3):
    global quadList,quadCounter
    quadCounter = quadCounter + 1
    newQuad = [quadCounter,[operator,operand1,operand2,operand3]] # making a new quad entry.
    quadList.append(newQuad)
```

2. **nextQuad**: είναι μία συνάρτηση που θα μας επιστρέφει (μέσω return) την επόμενη ελεύθερη ετικέτα της “quadList” όταν κληθεί. Δεν θα επηρεάζει τον μετρητή ετικετών μας όμως.

```
def nextQuad():
    global quadCounter
    temporary = quadCounter + 1                                    # finding the next label.
    return temporary
```

3. **newTemp**: η συνάρτηση που θα επιστρέφει το όνομα της επόμενης προσωρινής μεταβλητής μέσω μίας return.

```
def newTemp():
    global tempCounter
    temporary = "temp_" + str(tempCounter)
    tempCounter = tempCounter + 1
    return temporary
```

4. **emptyList**: η συνάρτηση που θα δημιουργεί μία κενή λίστα και θα την επιστρέφει.

```
def emptyList():
    emptyList = []                                                 # making an empty list.
    return emptyList
```

5. **makeList**: η συνάρτηση που θα δημιουργεί μία λίστα με μόνο της περιεχόμενο ότι δώσουμε ως παράμετρο.

```
def makeList(label):
    newList = [label]
    return newList
```

6. **mergeList**: η συνάρτηση που θα δημιουργεί μία λίστα και συνενώσει τις λίστες που δόθηκαν ως παράμετροι σε αυτή.

```
def mergeList(list1,list2):
    newList = list1 + list2
    return newList
```

7. **backpatch**: Διαβάζει μία μία της τετράδες που σημειώνονται στη λίστα list και για την τετράδα που αντιστοιχεί στην ετικέτα αυτή, συμπληρώνουμε το τελευταίο πεδίο της με το label.

```
def backpatch(list,label):
    for i in list:
        quadList[i][1][3] = label
```

8. **printList**: η συνάρτηση που θα εκτυπώνει την quadList, δηλαδή τον ενδιάμεσο κώδικα μας σε δύο τύπους αρχείου (.init και .c). Εδώ πρέπει να πούμε πως για το αρχείο c, όταν θα έχουμε function ή procedure μέσα στο πρόγραμμά μας, δεν θα εκτυπώνουμε ακριβώς τις εντολές που θα έπρεπε να εκτυπωθούν αλλά θα εκτυπώσουμε εντολές goto και αρχικοποίηση μεταβλητών καθώς δεν μπορούμε να καλέσουμε τις procedures ή τις functions.

```
def printList(list):                                               # printing the results
in .init and .c files.
```

```
    try:                                                    # if files .init and .c don't
exist -> create them.
        f_init = open("endiamesos.init", "w")
        f_c = open("endiamesos.c", "w")
    except FileExistsError:                                 # if files .init and .c exist
-> overwrite them.
        f_init = open("endiamesos.init", "x")
        f_c = open("endiamesos.c", "x")

    # print to the .init file.
    f_init.write("The Middleware results are:\n")
    for i in range(0,len(list)):
        f_init.write(str(i)+" :
"+str(list[i][1][0])+","+str(list[i][1][1])+","+str(list[i][1][2])+","+str(list[i][1][3])+"\n")

    # print to the .c file.
    variables = []
    f_c.write("//The Middleware results are:\n\n")
    f_c.write("#include <stdio.h>\n\nint main(){\n")
    for i in range(0,len(list)):

        one = list[i][1][0]
        two = list[i][1][1]
        three = list[i][1][2]
        four = list[i][1][3]
        if("main_" in str(two)) and (one == "begin_block"):
            f_c.write("\tLine_"+str(i)+": goto Line_"+str(i+1)+";")
            f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
        elif("main_" in str(two)) and (one == "end_block"):
            f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            f_c.write("}\n")
        elif("main_" not in str(two)) and (one in ["begin_block","end_block"]):
            f_c.write("\tLine_"+str(i)+": goto Line_"+str(i+1)+";")
            f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
        else:
            if(one == "halt"):
                f_c.write("\tLine_"+str(i)+": return 0;")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "jump"):
                f_c.write("\tLine_"+str(i)+": goto Line_"+str(four)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "out"):
                f_c.write("\tLine_"+str(i)+": printf(\"%d\","+str(two)+");")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "in"):
                f_c.write("\tLine_"+str(i)+": scanf(\"%d\", &"+two+");")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "ret"):
                f_c.write("\tLine_"+str(i)+": goto Line_"+str(i+1)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one in ["+","-","*","/"]):
                f_c.write("\tLine_"+str(i)+": ")
                if(four not in variables):
```

```
                    f_c.write("int "+str(four)+" = "+str(two)+" "+str(one)+" "+str(three)+";")
                    variables.append(four)
                else:
                    f_c.write(str(four)+" = "+str(two)+" "+str(one)+" "+str(three)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one in [">","<","<=",">="]):
                f_c.write("\tLine_"+str(i)+": if("+str(two)+" "+str(one)+" "+str(three)+")
goto Line_"+str(four)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "="):
                f_c.write("\tLine_"+str(i)+": if("+str(two)+" == "+str(three)+") goto
Line_"+str(four)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "<>"):
                f_c.write("\tLine_"+str(i)+": if("+str(two)+" != "+str(three)+") goto
Line_"+str(four)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == ":="):
                f_c.write("\tLine_"+str(i)+": ")
                if(four not in variables):
                    f_c.write("int "+str(four)+" = "+str(two)+";")
                    variables.append(four)
                else:
                    f_c.write(str(four)+" = "+str(two)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(one == "par"):
                if(three == "cv"):
                    f_c.write("\tLine_"+str(i)+": goto Line_"+str(i+1)+";")
                elif(three == "ret"):
                    if(two not in variables):
                        f_c.write("\tLine_"+str(i)+": int "+str(two)+";")
                        variables.append(four)
                    else:
                        f_c.write("\tLine_"+str(i)+": "+str(two)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")
            elif(list[i][1][0] == "call"):
                f_c.write("\tLine_"+str(i)+": goto Line_"+str(i+1)+";")
                f_c.write("\t//"+str(i)+" :
"+str(one)+","+str(two)+","+str(three)+","+str(four)+"\n")

            else: f_c.write("\n")
```

Οι Τετράδες που παράγονται.

Πριν αναφερθούμε στην δημιουργία τους, πρέπει να αναφερθούμε στις διάφορες τετράδες που θα δημιουργούνται στον ενδιάμεσο κώδικα. Ο παρακάτω πίνακας τις αναφέρει αναλυτικά.

begin_block, <όνομα> , _ , _	Τετράδα για την αρχή κάποιου μπλοκ κώδικα.
end_block, <όνομα> , _ , _	Τετράδα για τη λήξη κάποιου μπλοκ κώδικα.
halt, _ , _ , _	Τετράδα για τη λήξη του προγράμματος.
:=, <ονομα2>, _, <ονομα1>	Τετράδα για την πράξη της ανάθεσης. (όνομα1 := όνομα2)

jump, _ , _ , <ετικέτα>	Τετράδα για τα jump.
ret, <όνομα>, _ , _	Τετράδα για τα return. [return(όνομα)]
out, <όνομα>, _ , _	Τετράδα για τα print. [print(όνομα)]
call,<όνομα>, _ , _	Τετράδα για τα call. [call(όνομα)]
in,<όνομα>, _ , _	Τετράδα για τα input. [input(όνομα)]
par,<όνομα>, ref, _	Τετράδα για τις παραμέτρους με inout.
par,<όνομα>, cv, _	Τετράδα για τις παραμέτρους με in.
par,<όνομα>,ret,_	Τετράδα για τα return μίας συνάρτησης που καλείται.
relOp, op1, op2, op3	Τετράδα για λογικές και αριθμητικές πράξεις. (op3 := op1 relOp op2)

Αλλαγές στον Συντακτικό Αναλυτή

Αφού δημιουργήσαμε τις βοηθητικές μεταβλητές και συναρτήσεις για την παραγωγή του ενδιάμεσου κώδικα, στην συνέχεια έπρεπε να κάνουμε κάποιες αλλαγές στον κώδικα της Συντακτικής Ανάλυσης. Οι αλλαγές αυτές γίνονται καθώς η παραγωγή ενδιάμεσου κώδικα σαν ενότητα λογισμικού παίρνει ως είσοδο το δέντρο της συντακτικής ανάλυσης, το οποίο δεν υπάρχει ουσιαστικά και άρα ο ενδιάμεσος κώδικας θα παράγεται όσο παίρνει μέρος η Συντακτική Ανάλυση. Οι αλλαγές μας ήταν οι παρακάτω (στον κώδικα που παρατίθεται θα είναι υπογραμμισμένες με μπλε χρώμα).

1. Στην συνάρτηση **program,** προσθέσαμε μία μεταβλητή ID που θα κρατά ένα όνομα ως “**main_<όνομα που βρήκαμε από την Λεκτική Ανάλυση για το πρόγραμμα>**”. Αυτό το όνομα αργότερα θα χρησιμοποιηθεί για να το περάσουμε ως παράμετρο στην programBlock.

```
def program():
    global currentToken,currentScope,symbolTable
    if(currentToken[0] == "programTOKEN"):
            currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            ID = "main_" + currentToken[1]
            currentToken = lexicalAnalyser()
            programBlock(ID)
            if(currentToken[0] == "periodTOKEN"):
                currentToken = lexicalAnalyser()
                if(currentToken[0] == "EOF"):
                    currentToken = lexicalAnalyser()
                else:
                    error(4,currentToken[1])
            else:
                error(3,currentToken[1])
        else:
            error(2,currentToken[1])
    else:
        error(1,currentToken[1])
```

2. Στην συνάρτηση **programBlock**, αρχικά προσθέσαμε μία παράμετρο που θα είναι ουσιαστικά το όνομα που θα αντιστοιχεί σε κάθε program-block. Ακόμα εισήγαμε την δημιουργία της τετράδας που αντιστοιχεί στο “ξεκίνημα” των προγραμματιστικών μπλοκ (begin_block) καθώς και στην “λήξη” τους (end_block). Τέλος, αν το όνομα που δόθηκε ήταν το όνομα του προγράμματος (main_<όνομα>), τότε μαζί με την λήξη του μπλοκ, θα λαμβάνουμε και την τετράδα λήξης του προγράμματος (halt).

```
def programBlock(name):
    global currentToken
    if(currentToken[0] == "leftCurlyBracketTOKEN"):             # if the next token is '{'
        currentToken = lexicalAnalyser()
        programDeclarations()
        programSubprograms()
        genQuad("begin_block",name,"_","_")
        programStatementBlock()
        if(currentToken[0] == "rightCurlyBracketTOKEN"):
            currentToken = lexicalAnalyser()
            if("main_" in name):
                genQuad("halt","_","_","_")
            genQuad("end_block",name,"_","_")
        else:
            error(6,currentToken[1])
    else:
        error(5,currentToken[1])
```

3. Στην συνάρτηση **programSumbprograms**, κινηθήκαμε όπως και στην program αφού μιλάμε για υποπρογράμματα. Δηλαδή, προσθέσαμε μία μεταβλητή ID που θα κρατά το όνομα του αντίστοιχου υποπρογράμματος, και την χρησιμοποιήσαμε για να καλέσουμε την συνάρτηση programBlock. Εδώ η μόνη διαφορά είναι ότι η μεταβλητή δεν θα έχει πρόθεμα “main_” αφού δεν μιλάμε για το κύριο πρόγραμμα.

```
def programSubprograms():
    global currentToken,currentScope,symbolTable
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            ID = currentToken[1]
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "leftRoundBracketTOKEN"):     # if the next token is '('
                currentToken = lexicalAnalyser()
                formalParameterList()
                if(currentToken[0] == "rightRoundBracketTOKEN"):    # if the next token is ')'
                    currentToken = lexicalAnalyser()
                    programBlock(ID)
                    programSubprograms()
                else:
                    error(11,currentToken[1])
            else:
                error(10,currentToken[1])
        else:
            error(9,currentToken[1])
```

4. Στην συνάρτηση **statement**, κάναμε κάποιες μικρές αλλαγές.

a) Αρχικά, στην περίπτωση που θέλουμε να μεταβούμε στην assignStatement, προσθέσαμε μία μεταβλητή (previousToken) που θα λαμβάνει το αλφαριθμητικό που υπάρχει πριν το σύμβολο ανάθεσης (:=) και θα την χρησιμοποιούμε για να καλέσουμε την συνάρτηση assignStatement. Αυτό γίνεται με σκοπό να μπορέσουμε να περάσουμε αργότερα σωστά τις τετράδες μας χωρίς να χάνουμε στοιχεία.

b) Στην περίπτωση που θέλουμε να μεταβούμε στην συνάρτηση returnPrintStatement, θα πρέπει από πριν να κρατάμε σε μία μεταβλητή (tokenDivider) τις ανάλογες λεκτικές μονάδες “return” ή “print” και να τις περάσουμε σαν παραμέτρους στην returnPrintStatement με σκοπό να

μπορέσουμε να περάσουμε αργότερα σωστά τις τετράδες μας χωρίς να χάνουμε στοιχεία.

```
def statement():
    global currentToken

    if(currentToken[0] == "identifierTOKEN"):                   # if we have ID token we can
        previousToken = currentToken
        currentToken = lexicalAnalyser()
        assignStatement(previousToken)
    elif(currentToken[0] == "ifTOKEN"):
        currentToken = lexicalAnalyser()
        ifStatement()
    elif(currentToken[0] == "whileTOKEN"):
        currentToken = lexicalAnalyser()
        whileStatement()
    elif(currentToken[0] == "switchcaseTOKEN"):
        currentToken = lexicalAnalyser()
        switchcaseStatement()
    elif(currentToken[0] == "forcaseTOKEN"):
        currentToken = lexicalAnalyser()
        forcaseStatement()
    elif(currentToken[0] == "incaseTOKEN"):
        currentToken = lexicalAnalyser()
        incaseStatement()
    elif(currentToken[0] == "callTOKEN"):
        currentToken = lexicalAnalyser()
        callStatement()
    elif(currentToken[0] == "returnTOKEN"):
        tokenDivider = currentToken[0]
        currentToken = lexicalAnalyser()
        returnPrintStatement(tokenDivider)
    elif(currentToken[0] == "inputTOKEN"):
        currentToken = lexicalAnalyser()
        inputStatement()
    elif(currentToken[0] == "printTOKEN"):
        tokenDivider = currentToken[0]
        currentToken = lexicalAnalyser()
        returnPrintStatement(tokenDivider)
```

5. Στην **assignStatement**, αρχικά προσθέσαμε μία παράμετρο που θα αναπαριστά το αλφαριθμητικό πριν το σύμβολο ανάθεσης (:=). Στην συνέχεια, πήραμε αποτέλεσμα (expr) από την συνάρτηση expression, και δημιουργήσαμε την τετράδα που αντιστοιχεί στην πράξη της ανάθεσης.

```
def assignStatement(previousToken):
    global currentToken
    if(currentToken[0] == "assignmentTOKEN"):
        currentToken = lexicalAnalyser()
        expr = expression()
        genQuad(":=",expr,"_",previousToken[1])
    else:
        error(15,currentToken[1])
```

6. Στην **ifStatement**, κάναμε τα εξής. Αρχικά θέσαμε έναν πίνακα (cond) ο οποίος θα είναι της μορφής **[[cond.true][cond.false]]**. Μετά, τον περάσαμε σαν παράμετρο στην condition, η οποία θα τον διαμορφώσει. Στην συνέχεια, κάναμε τα εικονιζόμενα στην φωτογραφία.

```
def ifStatement():
    global currentToken
    cond = [[],[]]
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        condition(cond)
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            backpatch(cond[0],nextQuad())
            statements()
            ifList = makeList(nextQuad())
            genQuad("jump","_","_","_")
            backpatch(cond[1],nextQuad())
            elsePart()
            backpatch(ifList,nextQuad())
        else:
            error(11,currentToken[1])
    else:
        error(10,currentToken[1])
```

```
ifStat    → if ( condition ) {p1} statements(1)  {p2}
            elsePart {p3}
elsePart  → else statements(2)
          | ε

{p1} : backpatch(condition.true,nextquad())
{p2} : ifList = makeList(nextQuad())
       genQuad('jump','_','_','_')
       backpatch(condition.false,nextquad())
{p3} : backpatch(ifList,nextquad())
```

7. Στην **whileStatement**, θέσαμε έναν πίνακα (cond) ο οποίος θα είναι της μορφής **[[cond.true][cond.false]]**. Μετά, τον περάσαμε σαν παράμετρο στην condition, η οποία θα τον διαμορφώσει. Ακόμα, ορίσαμε μία μεταβλητή (conditionQuad) η οποία θα κρατά την ετικέτα της συνθήκης του while. Στην συνέχεια, κάναμε τα εικονιζόμενα στην φωτογραφία.

```
def whileStatement():
    global currentToken
    cond = [[],[]]
    conditionQuad = nextQuad()
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        condition(cond)
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            backpatch(cond[0],nextQuad())
            statements()
            genQuad("jump","_","_",conditionQuad)
            backpatch(cond[1],nextQuad())
        else:
            error(11,currentToken[1])
    else:
        error(10,currentToken[1])
```

```
whileStat    →    while {p0} ( condition ) {p1}
                  statements {p2}

{p0} : condQuad = nextQuad()
{p1} : backpatch(condition.true,nextQuad())
{p2} : genQuad('jump','_','_',condQuad)
       backpatch(condition.false,nextQuad())
```

8. Στην **forcaseStatement**, θέσαμε έναν πίνακα (cond) ο οποίος θα είναι της μορφής **[[cond.true][cond.false]]**. Μετά, τον περάσαμε σαν παράμετρο στην condition, η οποία θα τον διαμορφώσει. Ακόμα, ορίσαμε μία μεταβλητή (firstConditionQuad) η οποία θα κρατά την ετικέτα της πρώτης συνθήκης του forcase. Στην συνέχεια, κάναμε τα εικονιζόμενα στην φωτογραφία.

```
def forcaseStatement():
    global currentToken
    cond = [[],[]]
    if(currentToken[0] == "caseTOKEN"):
        currentToken = lexicalAnalyser()
        firstCondQuad = nextQuad()
        if(currentToken[0] == "leftRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            condition(cond)
```

```
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                backpatch(cond[0],nextQuad())
                statements()
                genQuad("jump","_","_",firstCondQuad)
                backpatch(cond[1],nextQuad())
                if(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
                    forcaseStatement()
            else:
                error(11,currentToken[1])
        else:
            error(10,currentToken[1])
    elif(currentToken[0] == "defaultTOKEN"):
        currentToken = lexicalAnalyser()
        statements()
    else:
        error(16,currentToken[1])
```

```
forcaseStat  →   forcase  {p1}
                 ( case ( condition ) {p2}
                    statements(1)  {p3} )*
                 default statements(2)

{p1} :  firstCondQuad = nextQuad()
{p2} :  backpatch(condition.true,nextQuad())
{p3} :  genQuad('jump','_','_','_')
        backpatch(condition.false,nextQuad())
```

9. Στην **switchcaseStatement**, αρχικά ορίσαμε μία καθολική (global) μεταβλητή exitList η οποία στην αρχή θα είναι μία κενή λίστα. Στην συνέχεια κινηθήκαμε όπως και στην while, if και forcase. Στην συνέχεια, κάναμε τα εικονιζόμενα στην φωτογραφία. Πέρα όμως από αυτό, στο τέλος της switchcase θα ξανά αδιάζουμε την exitList καθώς είναι καθολική μεταβλητή και θα χρειαστεί να ξαναχρησιμοποιηθεί όπως είχε οριστεί αρχικά.

```
exitList = emptyList()
def switchcaseStatement():
    global currentToken,exitList
    cond = [[],[]]
    if(currentToken[0] == "caseTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "leftRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            condition(cond)
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                backpatch(cond[0],nextQuad())
                statements()
                temp = makeList(nextQuad())
                genQuad("jump","_","_","_")
                exitList = mergeList(exitList,temp)
                backpatch(cond[1],nextQuad())
                if(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
                    switchcaseStatement()
            else:
                error(11,currentToken[1])
        else:
            error(10,currentToken[1])
    elif(currentToken[0] == "defaultTOKEN"):
        currentToken = lexicalAnalyser()
        statements()
        backpatch(exitList,nextQuad())
        exitList = emptyList()
    else:
        error(16,currentToken[1])
```

```
switchcaseStat  →   switchcase {p0}
                    ( case ( condition ) {p1}
                       statements(1)  {p2} ))*
                    default statements(2)
                    {p3}

{p0} :  exitList = emptyList()
{p1} :  backpatch(condition.true,nextQuad())
{p2} :  t = makeList(nextQuad)
        genQuad('jump','_','_','_')
        exitList = mergeList(exitList,t)
        backpatch(condition.false,nextQuad())
{p3} :  backpatch(exitList,nextQuad())
```

10. Στην **incaseStatement**, πέρα από τις κινήσεις που είναι όπως οι προαναφερόμενες, εδώ ορίσαμε μία μεταβλητή (defaultDetector) η οποία θα γίνεται 1 όταν θα εντοπίζει default συνθήκη. Αυτό το κάναμε για να μπορέσουμε να διαχειριστούμε το flag μας αργότερα. Στην συνέχεια κινηθήκαμε όπως και στην while, if, switchcase και forcase. Στην συνέχεια,

κάναμε τα εικονιζόμενα στην φωτογραφία. Αφού τελειώσουμε, το flag θα γίνεται reset σε περίπτωση που γίνει αναδρομή.

```
def incaseStatement():
    global currentToken
    cond = [[],[]]
    defaultDetector = 0
    flag = newTemp()
    firstCondQuad = nextQuad()
    while(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
        genQuad(":=",0,"_",flag)
        if(currentToken[0] == "caseTOKEN"):
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "leftRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                condition(cond)
                if(currentToken[0] == "rightRoundBracketTOKEN"):
                    currentToken = lexicalAnalyser()
                    backpatch(cond[0],nextQuad())
                    statements()
                    backpatch(cond[1],nextQuad()+1)
                else:
                    error(11,currentToken[1])
            else:
                error(10,currentToken[1])
        elif(currentToken[0] == "defaultTOKEN"):
            defaultDetector = 1
            genQuad(":=",1,flag,firstCondQuad)
            currentToken = lexicalAnalyser()
            statements()
            flag = ""
        else:
            error(16,currentToken[1])
    if(defaultDetector == 0):
            genQuad(":=",1,flag,firstCondQuad)
```

incase → incase {p1}
(case (condition) {p2}
statements(1) {p3})*
default statements(2)

{p1} : flag = newTemp()
firstCondQuad = nextQuad()
genQuad(':=',0,_,flag)
{p2} : backpatch(condition.true,nextQuad())
{p3} : genQuad(':=',1,_,flag)
backpatch(condition.false,nextQuad())
{p4} : genQuad('=',1,_,flag,firstQuad)

11. Στην **returnPrintStatement**, θέσαμε μία παράμετρο η οποία κρατά αν έχουμε return ή print. Αν αυτή η μεταβλητή είναι returnTOKEN τότε θα δημιουργήσουμε τετράδα για return, αλλιώς θα δημιουργήσουμε τετράδα για print.

```
def returnPrintStatement(tokenDivider):
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        expr = expression()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            if(tokenDivider == "returnTOKEN"):
                genQuad("ret",expr,"_","_")
            if(tokenDivider == "printTOKEN"):
                genQuad("out",expr,"_","_")
            currentToken = lexicalAnalyser()
        else:
            error(11,currentToken[1])
    else:
        error(10,currentToken[1])
```

12. Στην **callStatement**, απλά πήραμε το όνομα της συνάρτησης που θέλουμε να καλέσουμε και το αποθηκεύσαμε σε μία μεταβλητή ID. Μετά, το χρησιμοποιήσαμε για να δημιουργήσουμε τετράδα για τα call.

```
def callStatement():
    global currentToken
    if(currentToken[0] == "identifierTOKEN"):
        ID = currentToken[1]
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "leftRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            actualParameterList()
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                genQuad("call",ID,"_","_")
            else:
                error(11,currentToken[1])
        else:
            error(10,currentToken[1])
    else:
        error(17,currentToken[1])
```

13. Στην **inputStatement**, κάναμε το ίδιο με την callStatement.

```
def inputStatement():
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            ID = currentToken[1]
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "rightRoundBracketTOKEN"):
                genQuad("in",ID,"_","_")
                currentToken = lexicalAnalyser()
            else:
                error(11,currentToken[1])
        else:
            error(18,currentToken[1])
    else:
        error(10,currentToken[1])
```

14. Στην actualParameterItem, ανάλογα με το αν είχαμε inout ή in, δημιουργήσαμε και τις ανάλογες τετράδες.

```
def actualParameterItem():
    global currentToken
    if(currentToken[0] == "inoutTOKEN"):
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            genQuad("par",currentToken[1],"ref","_")
            currentToken = lexicalAnalyser()
        else:
            error(14,currentToken[1])
    elif(currentToken[0] == "inTOKEN"):
        currentToken = lexicalAnalyser()
        expr = expression()
        genQuad("par",expr,"cv","_")
```

15. Στην **condition** κάναμε ότι και στην φωτογραφία. Όπου Q θα είναι η conditionCondition, R1 η cond1 και R2 η cond2. Σημαντικό είναι να πούμε ότι η παράμετρος που λαμβάνει η συνάρτηση, θα διαμορφωθεί ανάλογα μέσα από αυτή.

```
def condition(conditionCondition):
    global currentToken
    cond1 = cond2 = [[],[]]
    boolTerm(cond1)
    conditionCondition[0] = cond1[0]
    conditionCondition[1] = cond1[1]
    while(currentToken[0] == "orTOKEN"):
        currentToken = lexicalAnalyser()
        backpatch(conditionCondition[1],nextQuad())
        boolTerm(cond2)
        conditionCondition[0] = mergeList(conditionCondition[0],cond2[0])
        conditionCondition[1] = cond2[1]
```

```
Q → R(1) {p1} ( or {p2} R(2) {p3} )*
{p1} : Q.true = R(1).true
       Q.false = R(1).false
{p2} : backpatch(Q.true, nextquad())
{p3} : Q.false = mergeList(Q.false,R(2).false)
       Q.true = R(2).true
```

16. Στην **boolTerm**, κάναμε ότι και στην φωτογραφία. Όπου Q θα είναι η boolTermCondition, R1 η cond1 και R2 η cond2. Σημαντικό είναι να πούμε ότι η παράμετρος που λαμβάνει η συνάρτηση, θα διαμορφωθεί ανάλογα μέσα από αυτή.

```
def boolTerm(boolTermCondition):
    global currentToken
    cond1 = cond2 = [[],[]]
    boolFactor(cond1)
    boolTermCondition[0] = cond1[0]
    boolTermCondition[1] = cond1[1]
    while(currentToken[0] == "andTOKEN"):
        currentToken = lexicalAnalyser()
        backpatch(boolTermCondition[0],nextQuad())
        boolFactor(cond2)
        boolTermCondition[0] = cond2[0]
        boolTermCondition[1] = mergeList(boolTermCondition[1],cond2[1])
```

```
Q → R(1) {p1} ( and {p2} R(2) {p3} )*
```

```
{p1} : Q.true = R(1).true
       Q.false = R(1).false
{p2} : backpatch(Q.true, nextquad())
{p3} : Q.false = mergeList(Q.false,R(2).false)
       Q.true = R(2).true
```

17. Στην **boolFactor**, κάναμε ότι και στην φωτογραφία. Όπου R θα είναι η boolFactorCondition, E1 το expr1, E2 το expr2 και B η cond. Σημαντικό είναι να πούμε ότι η παράμετρος που λαμβάνει η συνάρτηση, θα διαμορφωθεί ανάλογα μέσα από αυτή. Ακόμα βάλαμε και μία μεταβλητή που αν βρίσκει not θα γίνεται 1. Αυτό μας βοηθά στο να αποφασίσουμε την υλοποίηση που θα γίνει.

```
def boolFactor(boolFactorCondition):
    global currentToken
    cond = [[],[]]
    notDetector = 0                                    # not detection
    if(currentToken[0] == "notTOKEN"):
        currentToken = lexicalAnalyser()
        notDetector = 1
    # box brackets
    if(currentToken[0] == "leftBoxBracketTOKEN"):
        currentToken = lexicalAnalyser()
        condition(cond)
        if(currentToken[0] == "rightBoxBracketTOKEN"):
            currentToken = lexicalAnalyser()
            if(notDetector == 0):
                boolFactorCondition[0] = cond[0]
                boolFactorCondition[1] = cond[1]
            else:
                boolFactorCondition[0] = cond[1]
```

```
R → [ B ] {p1}
{p1} : R.true = B.true
       R.false = B.false
```

```
R → not [ B ] {p1}
{p1} : R.true = B.false
       R.false = B.true
```

```
                boolFactorCondition[1] = cond[0]
        else:
            error(19,currentToken[1])
    else:
        expr1 = expression()
        if(currentToken[0] in
["equalTOKEN","smallerEqualTOKEN","largerEqualTOKEN","smallerTOKEN","largerTOKEN","differentTO
KEN"]):
            relationalOperator = currentToken[1]
            currentToken = lexicalAnalyser()
            expr2 = expression()
            boolFactorCondition[0] = makeList(nextQuad())
            genQuad(relationalOperator,expr1,expr2,"_")
            boolFactorCondition[1] = makeList(nextQuad())
            genQuad("jump","_","_","_")
        else:
            error(13,currentToken[1])
```

```
R → E(1) rel_op E(2) {p1}
{p1} : R.true = makeList(nextQuad())
       genQuad(rel_op, E(1).place, E(2).place}, '_')
       R.false = makeList(nextQuad())
       genQuad('jump', '_', '_', '_')
```

18. Στην **expression**, κάναμε ότι και στην φωτογραφία και κάναμε return το Eplace που για εμάς είναι το expressionPlace. Ακόμα όπου T1, T2 θα είναι τα term1 και term2 αντίστοιχα και η temp θα είναι η w.

```
def expression():
    global currentToken,currentScope
    optionalSign()
    term1 = term()
    while(currentToken[0] in ["addTOKEN","subTOKEN"]):
        relationalOperator = currentToken[1]
        currentToken = lexicalAnalyser()
        term2 = term()
        temp = newTemp()
        genQuad(relationalOperator,term1,term2,temp)
        term1 = temp
        expressionPlace = term1
    return expressionPlace
```

```
E → T(1) ( + T(2) {p1} )* {p2}
{p1} : w = newTemp()
       genQuad('+', T(1).place, T(2).place, w)
       T(1).place = w
{p2} : E.place = T(1).place
```

19. Στην **term**, κάναμε ότι και στην φωτογραφία και κάναμε return το Tplace που για εμάς είναι το termPlace. Ακόμα όπου F1, F2 θα είναι τα factor1 και factor2 αντίστοιχα και η temp θα είναι η w.

```
def term():
    global currentToken
    factor1 = factor()
    while(currentToken[0] in ["mulTOKEN","divTOKEN"]):
        relationalOperator = currentToken[1]
        currentToken = lexicalAnalyser()
        factor2 = factor()
        temp = newTemp()
        genQuad(relationalOperator,factor1,factor2,temp)
        factor1 = temp
        termPlace = factor1
    return termPlace
```

```
T → F(1) ( * F(2) {p1} )* {p2}
{p1} : w = newTemp()
       genQuad('+', F(1).place, F(2).place, w)
       F(1).place = w
{p2} : T.place = F(1).place
```

20. Στην **factor**, κάναμε ότι και στην φωτογραφία και κάναμε return το Fplace που για εμάς είναι το factorPlace.

```
def factor():

    global currentToken
```

```
    if(currentToken[0] == "numberTOKEN"):
        factorPlace = currentToken[1]
        currentToken = lexicalAnalyser()
    elif(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        expr = expression()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            factorPlace = expr
            currentToken = lexicalAnalyser()
        else:
            error(11,currentToken[1])
    elif(currentToken[0] == "identifierTOKEN"):
        ID = currentToken[1]
        currentToken = lexicalAnalyser()
        factorPlace = idTail(ID)
    return factorPlace
```

```
F  →  ID  {p1}
   {p1} : F.place = ID.place
```

```
F  →  ( E ) {p1}
   {p1} : F.place = E.place
```

21. Στην **idTail**, βάλαμε μία παράμετρο ID για να μπορούμε να παίρνουμε το αλφαριθμητικό που είχαμε βρει πριν κληθεί. Ακόμα προσθέσαμε δημιουργία τετράδων για τις προσωρινές μεταβλητές και τα calls τους. Στο τέλος επιστέφουμε το ID που είχαμε λάβει στην αρχή.

```
def idTail(name):
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        actualParameterList()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            temp = newTemp()
            genQuad("par",temp,"ret","_")
            genQuad("call",name,"_","_")
            return temp
        else:
            error(11,currentToken[1])
    return name
```

Τέλος καλέσαμε εκτύπωση του ενδιάμεσου κώδικα μας στο τέλος της εκτέλεσης της συντακτική ανάλυσης.

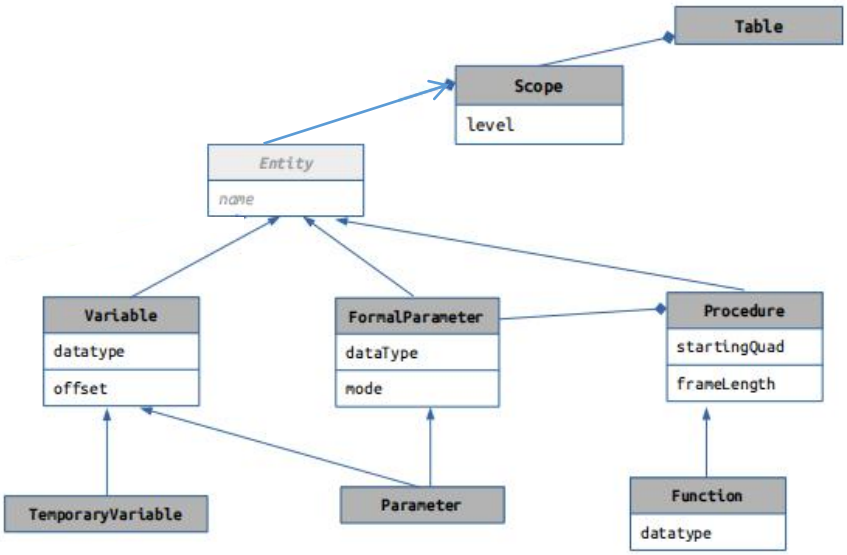
```
# Syntax Analyser
def syntaxAnalyser():
    global currentToken,file
    currentToken = lexicalAnalyser()
    try:
        file = open("pinakas.symb","w")
    except FileExistsError:
        file = open("pinakas.symb","x")
    program()
    file.close()
    print("Compilation successfully completed.\n")
    printList(quadList)
```

Πίνακας Συμβόλων

Ο πίνακας συμβόλων είναι μία δυναμική δομή στην οποία αποθηκεύεται πληροφορία που σχετίζεται με τις μεταβλητές του προγράμματος, τις διαδικασίες και 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. Είναι λοιπόν προφανές ότι στην δημιουργία του μεταφραστή μας, θα είναι το επόμενο αντικείμενο σχεδίασης μας.

Κλάσεις

Για την δημιουργία του πίνακα συμβόλων, ήταν αναγκαία η σωστή δημιουργία κλάσεων με σωστές συνδέσεις. Εμείς προσπαθήσαμε να πετύχουμε τις εξής συνδέσεις μεταξύ των κλάσεων:



Πετύχαμε το σχηματικό αυτό δημιουργώντας τις προαναφερόμενες κλάσεις ως εξής.

1. Δημιουργήσαμε την κλάση **Entity** που μόνο χαρακτηριστικό της θα είναι το name.

```
class Entity:                                          # class for entity.
    def __init__(self,name):                           # initialiser.
        self.name = name
```

2. Μετά δημιουργήσαμε την κλάση **Variable** που θα κληρονομεί την Entity και θα έχει ως χαρακτηριστικά το name, data type και το offset. Μέσα της θα ορίσουμε και μία συνάρτηση printFields που θα την επιστρέφει με σκοπό να την πάρουμε αργότερα για εκτύπωση.

```
class Variable(Entity):
    def __init__(self,name,dataType,offset):           # initialiser.
        Entity.__init__(self,name)
        self.dataType = dataType
        self.offset = offset

    def printFields(self):
        return "|| name = "+str(self.name)+"|| data type = "+str(self.dataType)+"|| offset =
"+str(self.offset)
```

3. Δημιουργήσαμε και την κλάση **TemporaryVariable** για τις προσωρινές μεταβλητές, η οποία θα κληρονομεί όπως είναι την κλάση Variable μαζί και με την printFields.

```
class TemporaryVariable(Variable): pass
```

4. Ακόμα, δημιουργήσαμε την κλάση **FormalParameter** για τις παραμέτρους, η οποία θα κληρονομεί την Entity και θα έχει ως χαρακτηριστικά το name, data type, mode και το offset. Μέσα της θα ορίσουμε και μία συνάρτηση printFields που θα την επιστρέφει με σκοπό να την πάρουμε αργότερα για εκτύπωση.

```
class FormalParameter(Entity):
    def __init__(self,name,dataType,mode,offset):              # initialiser.
        Entity.__init__(self,name)
        self.dataType = dataType
        self.mode = mode
        self.offset = offset

    def printFields(self):
        return "|| name = "+str(self.name)+"|| data type = "+str(self.dataType)+"|| mode =
"+str(self.mode)+"|| offset = "+str(self.offset)
```

5. Δημιουργήσαμε και την **Parameter** κλάση που θα είναι για τις παραμέτρους και θα κληρονομεί την Variable αλλά και την FormalParameter κλάση. Θα έχει ως χαρακτηριστικά το name,data type, mode και το offset.

```
class Parameter(FormalParameter,Variable):                     # class for parameters that
implement the classes entity and formalparameters.
    def __init__(self,name,dataType,mode,offset):              # initialiser.
        FormalParameter.__init__(self,name,dataType,mode)
        Variable.__init__(self,name,dataType,offset)
```

6. Φτιάξαμε και μία κλάση για τις procedures, την **Procedure** που θα κληρονομεί την Entity και θα έχει ως χαρακτηριστικά το name, τη λίστα με τα formal parameters της, το starting quad (αριθμό που θα λέει που ξεκινά στον ενδιάμεσο κώδικα) και το frame length. Μέσα της θα ορίσουμε και μία συνάρτηση printFields που θα την επιστρέφει με σκοπό να την πάρουμε αργότερα για εκτύπωση. Ακόμα θα ορίσουμε και μία συνάρτηση που θα μας επιτρέπει να προσθέτουμε αντικείμενα στην λίστα formalParameters. Τέλος ορίσαμε και δύο συναρτήσεις που μας επιτρέπουν να ενημερώνουμε τα startingQuad και frameLength.

```
class Procedure(Entity):
    def __init__(self,name,formalParameters,startingQuad,frameLength): # initialiser.
        Entity.__init__(self,name)
        self.startingQuad = startingQuad
        self.frameLength = frameLength
        self.formalParameters = formalParameters

    def printFields(self):
        return "|| name = "+str(self.name)+"|| frame length = "+str(self.frameLength)+"||
starting quad = "+str(self.startingQuad)+"|| formal parameters ="+str(self.formalParameters)

    def addFormalParameters(self,update):
        self.formalParameters.append(update)

    def updateStartingQuad(self,update):
        self.startingQuad = update

    def updateFrameLength(self,update):
        self.frameLength = update
```

7. Ακόμα φτιάξαμε μία κλάση για τα functions, την **Function**. Η κλάση αυτή θα κληρονομεί την Procedure και όλες της τις συναρτήσεις και θα έχει ως χαρακτηριστικά το name, τη λίστα με τα formal parameters της, το starting quad (αριθμό που θα λέει που ξεκινά στον ενδιάμεσο κώδικα), το frame length και το data type. Μέσα της θα ορίσουμε και μία συνάρτηση printFields που θα την επιστρέφει με σκοπό να την πάρουμε αργότερα για εκτύπωση.

```
class Function(Procedure):
    def __init__(self,name,formalParameters,startingQuad,frameLength,dataType): # initialiser.
        Procedure.__init__(self,name,formalParameters,startingQuad,frameLength)
        self.dataType = dataType

    def printFields(self):
        return "|| name = "+str(self.name)+"|| data type = "+str(self.dataType)+"|| frame
length = "+str(self.frameLength)+"|| starting quad = "+str(self.startingQuad)+"|| formal
parameters ="+str(self.formalParameters)
```

8. Μετά δημιουργήσαμε μία κλάση για τα επίπεδα μας, την **Scope**. Εδώ θα έχουμε χαρακτηριστικά της κλάσης τα name, entityList (μία λίστα με τα αντικείμενα που θα έχει το επίπεδο) και nextScope, δηλαδή το επόμενο επίπεδο που θα πάμε αν αυτό εδώ διαγραφεί. Ακόμα έχει μία συνάρτηση getOffset η οποία θα μας δίνει το επόμενο στην σειρά offset που θα έχουμε με σκοπό να το ορίσουμε στις οντότητες της entityList. Ακόμα, θα έχουμε και μία συνάρτηση που θα ψάχνει το entityList για να βρει κάποια συγκεκριμένη μεταβλητή. Αν την βρει, θα την επιστρέφει αλλιώς θα μας δίνει None (τιμή αντίστοιχη του null στην python).

```
class Scope:                                                    # class for the scopes.
    def __init__(self,name,entityList,nextScope):               # initialiser.
        self.name = name
        self.entityList = entityList
        self.nextScope = nextScope

    def getOffset(self):
        if(self.entityList == None) or (self.entityList == []):
            newOffset = 12
        else:
            count = -1
            for i in range(0,len(self.entityList)):
                offset = 0
                if(isinstance (self.entityList[count],Variable)) or (isinstance
(self.entityList[count],FormalParameter)) or (isinstance
(self.entityList[count],TemporaryVariable)) or (isinstance (self.entityList[count],Parameter)):
                    offset = self.entityList[count].offset
                    break
                else:
                    count = count - 1
            newOffset = offset + 4
        return newOffset

    def searchEntity(self,name):
        entity = None
        for i in self.entityList:
            if (i.name == name):
                entity = i
                break
        return entity
```

9. Τέλος δημιουργήσαμε και μία κλάση για τον πίνακα μας, την Table. Αυτή η κλάση θα έχει ως χαρακτηριστικά μόνο το scopeList, μία λίστα που θα κρατά τα επίπεδα που φτιάχνονται. Τέλος, θα

έχει και μία συνάρτηση printScope που θα την επιστρέφει με σκοπό να την πάρουμε αργότερα για εκτύπωση.

```
class Table():                                          # class for the symbol table.
    def __init__(self,scopeList):
        self.scopeList = scopeList

    def printTable(self):
        for i in self.scopeList:
            i.printScope()
```

Βοηθητικές Συναρτήσεις και καθολικές μεταβλητές

Για την επίτευξη του πίνακα συμβόλων χρειαστήκαμε και κάποιες μεταβλητές και βοηθητικές συναρτήσεις. Αρχικά, χρειάστηκαν δύο καθολικές μεταβλητές που η μία (currentScope) θα κρατά το επίπεδο που βρισκόμαστε και η άλλη την οντότητα του πίνακα συμβόλων μας (symbolTable). Για το αρχείο .symb όμως χρειαστήκαμε και την καθολική μεταβλητή file που θα είναι το αρχείο που θα γράφουμε τα αποτελέσματα μας.

```
currentScope = None
symbolTable = None
file = None
```

Τώρα για βοηθητικές συναρτήσεις φτιάξαμε τις παρακάτω.

1. Μία συνάρτηση που θα δημιουργεί νέο επίπεδο, θα το προσθέτει στην λίστα του πίνακα μας και θα το υιοθετεί ως currentScope. Η συνάρτηση θα έχει το όνομα **createScope** και θα παίρνει ως παράμετρο το όνομα που θέλουμε να δώσουμε στο νέο επίπεδο.

```
def createScope(name):
    global currentScope,symbolTable

    if(symbolTable == None):                            # if we dont have a table yet.
        symbolTable = Table([])
    if(currentScope == None):                           # if we have our first scope
        currentScope = Scope(name,[],None)              # the new scope.
        symbolTable.scopeList.append(currentScope)      # adding it to the scope list
    else:
        newScope = Scope(name,[],currentScope)
        symbolTable.scopeList.append(newScope)
        currentScope = newScope
```

2. Μία συνάρτηση που θα διαγράφει το τωρινό επίπεδο και θα μας πηγαίνει στο προηγούμενο από αυτό. Η συνάρτηση θα έχει όνομα **removeScope** και σε περίπτωση που δεν έχουμε άλλα επίπεδα για διαγραφή, θα εκτυπώνει μήνυμα λάθους αν κληθεί.

```
def removeScope():
    global currentScope,symbolTable
    if(currentScope == None):                   # if we don't have any more scopes to remove.
        error(20,None)
    else:
        symbolTable.scopeList.remove(symbolTable.scopeList[-1]) # removing it from the table.
        currentScope = currentScope.nextScope                   # moving on to the next scope.
```

3. Μία συνάρτηση που θα δημιουργεί καινούργιες οντότητες. Θα έχει όνομα **insertEntity** και ορίσματα θα είναι ο τύπος της οντότητας που θέλουμε να δημιουργήσουμε (variable,temporary,parameter,procedure,function), το όνομα που θέλουμε να της δώσουμε και το mode της αν έχει. Η οντότητα αυτή θα εντάσσεται στην entityList του τωρινού μας επιπέδου.

```
def insertEntity(entityType,name,mode):
    global currentScope
    newOffset = currentScope.getOffset()
    if(entityType == "variable"):
        newEntity = Variable(name,"VAR",newOffset)
    elif(entityType == "temporary"):
        newEntity = TemporaryVariable(name,"TEMP",newOffset)
    elif(entityType == "parameter"):
        newEntity = Parameter(name,"PARAM",mode,newOffset)
    elif(entityType == "formal"):
        newEntity = FormalParameter(name,"FORMAL",mode,newOffset)
    elif(entityType == "procedure"):
        newEntity = Procedure(name,[],None,None)
    elif(entityType == "function"):
        newEntity = Function(name,[],None,None,"FUNC")
    else:
        error(21,None)
    currentScope.entityList.append(newEntity)                    # inserting the new Entity
```

4. Μία συνάρτηση που θα ψάχνει να βρει μία οντότητα μέσα σε όλο τον πίνακα συμβόλων. Θα έχει το όνομα **searchEntity** και όρισμα το όνομα της συνάρτησης. Αν την βρει, θα την επιστρέφει αλλιώς θα επιστρέφει None.

```
def searchEntity(name):
    global symbolTable
    entity = None
    for i in symbolTable.scopeList:
        entity = i.searchEntity(name)
        if(entity != None):
            break                                                # if we find it -> stop.
    return entity
```

5. Μία συνάρτηση που θα ενημερώνει ή το startingQuad ή το frameLength μίας οντότητας. Θα έχει το όνομα **updateEntity** και πεδία της θα είναι το όνομα της οντότητας που θα ενημερώσουμε, το τι θα ενημερώσουμε (fieldName) και το σε τι θα ενημερωθεί (updateValue).

```
def updateEntity(name,fieldName,updateValue):
    global currentScope,symbolTable
    entity = searchEntity(name)                                  # find the entity.
    if(fieldName == "startingQuad"):
        entity.updateStartingQuad(updateValue)
    elif(fieldName == "frameLength"):
        entity.updateFrameLength(updateValue)
    else:
        error(22,name)
```

6. Τέλος, θα έχουμε και μία συνάρτηση που θα προσθέτει παραμέτρους στην λίστα formalParameters μίας οντότητας. Θα έχει όνομα **addFormalParameter** και ορίσματα το όνομα της οντότητας και ότι θέλουμε να βάλουμε στην λίστα της.

```
def addFormalParameter(name,formalParameter):
    global currentScope
    entity = searchEntity(name)                                    # find the entity.
    entity.addFormalParameters(formalParameter)
```

7. Για την εκτύπωση σε αρχείο .symb θα χρειαστούμε την παρακάτω συνάρτηση:

```
def printTable(f):
    global symbolTable
    symbolTable.printTable(f)
```

Αλλαγές στον Συντακτικό Αναλυτή

Για την επίτευξη του πίνακα συμβόλων, κάναμε και κάποιες αλλαγές και στον Συντακτικό Αναλυτή μας. Πιο συγκεκριμένα:

1. Στην συνάρτηση **syntaxAnalyser**, προσθέσαμε το άνοιγμα και το κλείσιμο του αρχείου .symb

```
def syntaxAnalyser():
    global currentToken,file
    currentToken = lexicalAnalyser()
    try:
        file = open("pinakas.symb","w")
    except FileExistsError:
        file = open("pinakas.symb","x")
    program()
    file.close()
    print("Compilation successfully completed.\n")
```

2. Στην συνάρτηση **program**, χρειαστήκαμε να δημιουργήσουμε το νέο μας επίπεδο, να εισάγουμε κάποια prints στο αρχείο .symb για να βεβαιωθούμε πως η λειτουργία του πίνακα είναι σωστή αλλά και όταν η συνάρτηση πάει να τελειώσει, να διαγράφουμε το επίπεδο που είχαμε δημιουργήσει.

```
def program():
    global currentToken,currentScope,file
    if(currentToken[0] == "programTOKEN"):
        currentToken = lexicalAnalyser()                           # move on to the next token.
        if(currentToken[0] == "identifierTOKEN"):
            ID = "main_" + currentToken[1]
            currentToken = lexicalAnalyser()
            createScope(ID)                                        # *Change for symbol table*
            programBlock(ID)                                       # *Change for middleware*
            printTable(file)
            removeScope()                                          # *Change for symbol table*
            if(currentToken[0] == "periodTOKEN"):
                currentToken = lexicalAnalyser()
                if(currentToken[0] == "EOF"):
                    currentToken = lexicalAnalyser()
                else:
                    error(4,currentToken[1])
            else:
                error(3,currentToken[1])
        else:
```

```
            error(2,currentToken[1])
    else:
        error(1,currentToken[1])
```

3. Στην **programBlock**, φτιάξαμε μία μεταβλητή που θα κρατά το startingQuad των οντοτήτων μας. Προσθέσαμε στην συνάρτηση ένα return για να το επιστρέφει με σκοπό να το χρησιμοποιήσουμε για να κάνουμε ενημέρωση.

```
def programBlock(name):
    global currentToken
    if(currentToken[0] == "leftCurlyBracketTOKEN"):             # if the next token is '{'
        currentToken = lexicalAnalyser()
        programDeclarations()
        programSubprograms()
        genQuad("begin_block",name,"_","_")
        startingQuad = nextQuad()
        programStatementBlock()
        if(currentToken[0] == "rightCurlyBracketTOKEN"):
            currentToken = lexicalAnalyser()
            if("main_" in name):
                genQuad("halt","_","_","_")
            genQuad("end_block",name,"_","_")
        else:
            error(6,currentToken[1])
    else:
        error(5,currentToken[1])
    return startingQuad
```

4. Στην variable, θα εισάγουμε τις μεταβλητές που βρίσκουμε στο entityList του επιπέδου που είμαστε.

```
def variableList():
    global currentToken
    if(currentToken[0] == "identifierTOKEN"):                   # if we have an ID (identifier)
        insertEntity("variable",currentToken[1],None)
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "commaTOKEN"):
            currentToken = lexicalAnalyser()
            variableList()                                      # recursion
    else:
        error(8,currentToken[1])
```

5. Στην **programSubprograms**, ίσως κάναμε και τις περισσότερες αλλαγές. Αρχικά ορίσαμε μία μεταβλητή type ως function ή procedure ανάλογα με το τι έχουμε. Η μεταβλητή θα χρησιμοποιηθεί αμέσως μετά για να εισάγουμε ως οντότητα το subprogram μας. Μετά με μία μεταβλητή temp πήραμε από την programBlock το startingQuad και το κάναμε ενημέρωση στην οντότητα μας.Τέλος κάναμε ενημέρωση και το frameLength και βάλαμε κάποια prints για να δούμε αποτελέσματα στο αρχείο .symb.

```
def programSubprograms():
    global currentToken,currentScope,file
    # if we have a function or a procedure:
    if(currentToken[0] in ["functionTOKEN","procedureTOKEN"]):
        if(currentToken[0] == "functionTOKEN"):
            type = "function"
        else:
            type = "procedure"
        currentToken = lexicalAnalyser()
```

```
        if(currentToken[0] == "identifierTOKEN"):
            ID = currentToken[1]
            insertEntity(type,ID,None)
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "leftRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                createScope(ID)
                formalParameterList(ID)
                if(currentToken[0] == "rightRoundBracketTOKEN"):
                    currentToken = lexicalAnalyser()
                    temp = programBlock(ID)
                    updateEntity(ID,"startingQuad",temp)
                    updateEntity(ID,"frameLength",currentScope.getOffset())
                    printTable(file)
                    removeScope()
                    programSubprograms()
                else:
                    error(11,currentToken[1])
            else:
                error(10,currentToken[1])
        else:
            error(9,currentToken[1])
```

6. Στην **formalParameterList**, απλά προσθέσαμε ένα όρισμα για να το περάσουμε στη formalParameterItem.

```
def formalParameterList(name)
```

7. Στην **formalParameterItem**, εισάγουμε τα formal parameters στην λίστα της οντότητας της οποίας το όνομα είναι δοσμένο ως όρισμα. Ακόμα κάνουμε τις ανάλογες εισαγωγές formal παραμέτρων για in και inout.

```
def formalParameterItem(name):
    global currentToken
    if(currentToken[0] in ["inTOKEN","inoutTOKEN"]):
        in_inout = currentToken[1]
        addFormalParameter(name,in_inout)
        currentToken = lexicalAnalyser()
        if(currentToken[0] == "identifierTOKEN"):
            if(in_inout == "in"):
                insertEntity("formal",currentToken[1],"CV")
            else:
                insertEntity("formal",currentToken[1],"REF")
            currentToken = lexicalAnalyser()
        else:
            error(14,currentToken[1])
```

8. Στην **assignStatement**, εισήγαμε μία μεταβλητή found που θα ψάχνει αν έχει οριστεί ξανά η μεταβλητή που είναι για να της ανατεθεί κάτι. Αν δεν έχει θα την εισάγουμε στο entityList του επιπέδου μας.

```
def assignStatement(previousToken):
    global currentToken
    if(currentToken[0] == "assignmentTOKEN"):
        currentToken = lexicalAnalyser()
        expr = expression()
        found = searchEntity(previousToken[1])
        if(found == None):
```

```
            insertEntity("variable",previousToken[1],None)
        genQuad(":=",expr,"_",previousToken[1])
    else:
        error(15,currentToken[1])
```

9. Στην **term** και **expression** κάναμε ακριβώς την ίδια αλλαγή. Μέσα στην while, εισήγαμε μία μεταβλητή found που θα ψάχνει αν έχει οριστεί ξανά η προσωρινή μας μεταβλητή. Αν δεν έχει θα την εισάγουμε στο entityList του επιπέδου μας.

```
        found = searchEntity(temp)
        if(found == None):
            insertEntity("temporary",temp,None)
```

10. Τέλος, στην **idTail**, εισήγαμε τις προσωρινές μεταβλητές που δημιουργούνται εκεί στο entityList του επιπέδου μας.

```
def idTail(name):
    global currentToken
    if(currentToken[0] == "leftRoundBracketTOKEN"):
        currentToken = lexicalAnalyser()
        actualParameterList()
        if(currentToken[0] == "rightRoundBracketTOKEN"):
            currentToken = lexicalAnalyser()
            temp = newTemp()
            insertEntity("temporary",temp,None)
            genQuad("par",temp,"ret","_")
            genQuad("call",name,"_","_")
            return temp
        else:
            error(11,currentToken[1])
    return name
```

11. Πρέπει όμως και να μην ξεχάσουμε και την μεταβλητή flag του incaseStatement οπότε στην συνάρτηση αυτή κάναμε τα εξής:

```
def incaseStatement():
    global currentToken
    cond = [[],[]]
    defaultDetector = 0
    flag = newTemp()
    firstCondQuad = nextQuad()
    while(currentToken[0] in ["caseTOKEN","defaultTOKEN"]):
        insertEntity("temporary",flag,None)
        genQuad(":=",0,"_",flag)
        if(currentToken[0] == "caseTOKEN"):
            currentToken = lexicalAnalyser()
            if(currentToken[0] == "leftRoundBracketTOKEN"):
                currentToken = lexicalAnalyser()
                condition(cond)
                if(currentToken[0] == "rightRoundBracketTOKEN"):
                    currentToken = lexicalAnalyser()
                    backpatch(cond[0],nextQuad())
                    statements()
                    backpatch(cond[1],nextQuad()+1)
                else:
                    error(11,currentToken[1])
            else:
                error(10,currentToken[1])
```

```
        elif(currentToken[0] == "defaultTOKEN"):
            defaultDetector = 1
            genQuad(":=",1,flag,firstCondQuad)
            currentToken = lexicalAnalyser()
            statements()
            flag = ""
        else:
            error(16,currentToken[1])
    if(defaultDetector == 0):
        genQuad(":=",1,flag,firstCondQuad)
```

Τελικός κώδικας (Παραγωγή τελικού κώδικα)

Τελικό στάδιο στην δημιουργία ενός πλήρους λειτουργικού μεταφραστή είναι η παραγωγή του τελικού μας κώδικα. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μία σειρά εντολών τελικού κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων. Σε αυτό το στάδιο λοιπόν, θα κάνουμε την μετάφραση της γλώσσας Cimple σε RISC-V.

Αλλαγές στον κώδικα του Πίνακα Συμβόλων.

Για την επίτευξη της δημιουργίας του τελικού κώδικα, κάναμε αρχικά κάποιες αλλαγές στις βοηθητικές συναρτήσεις που είχαμε φτιάξει για τον Πίνακα Συμβόλων. Αυτό έγινε καθώς εμείς θελήσαμε να πάρουμε τις πληροφορίες του πίνακα που παράγουμε μέσα στην συντακτική ανάλυση χωρίς όμως να αλλάξουμε πάλι τον Συντακτικό Αναλυτή μας. Πιο συγκεκριμένα πράξαμε τα εξής:

1. Αρχικά, δημιουργήσαμε ένα νέο καθολικό αντικείμενο τύπου Table που θα χρησιμοποιήσουμε για να παράξουμε ένα deep copy ή αλλιώς βαθύ αντίγραφο του πίνακα συμβόλων μας (symbolTable) που θα περιέχει όλα τα επίπεδα ολοκληρωμένα χωρίς να τα διαγράφει όπως κάνει ο «κανονικός» μας πίνακας συμβόλων μόλις ολοκληρωθούν.

```
symbolTable = deepCopySymbolTable = None
```

2. Μετά, προχωρήσαμε στο να αρχικοποιούμε μαζί πλέον τις δύο αυτές οντότητες (symbolTable και deepCopySymbolTable) μέσα στην βοηθητική συνάρτηση createScope.

```
def createScope(name):
    global currentScope,symbolTable,deepCopySymbolTable
    if(symbolTable == None):                                    # if we don't have a table yet.
        symbolTable = Table([])
        deepCopySymbolTable = Table([])                         # *Change for final code*
    if(currentScope == None):
        currentScope = Scope(name,[],None)                      # the new scope.
        symbolTable.scopeList.append(currentScope)
    else:
        newScope = Scope(name,[],currentScope)
        symbolTable.scopeList.append(newScope)
        currentScope = newScope
```

3. Τέλος, εκμεταλλευτήκαμε το γεγονός ότι τα επίπεδα μόλις ολοκληρωθούν, όσο αφορά τις πληροφορίες τους, διαγράφονται από τον symbolTable μέσω της removeScope και έτσι αλλάξαμε την συνάρτηση removeScope με σκοπό λίγο πριν να διαγράφει ένα επίπεδο να το προσθέτει πρώτα στο deepCopySymbolTable.

```
def removeScope():
    global currentScope,symbolTable,deepCopySymbolTable
    if(currentScope == None):
        error(20,None)
    else:
        deepCopySymbolTable.scopeList.append(symbolTable.scopeList[-1])
        symbolTable.scopeList.remove(symbolTable.scopeList[-1]) # removing it from the table.
        currentScope = currentScope.nextScope
```

Έτσι λοιπόν μας μένει ένας πίνακας μετά το τέλος της διαδικασίας της συντακτικής ανάλυσης με τα επίπεδα ανάποδα σε σειρά και ολοκληρωμένα, όσο αφορά τις πληροφορίες που έχουν, για να χρησιμοποιηθεί για την παραγωγή τελικού κώδικα.

Βοηθητικές Συναρτήσεις και καθολικές μεταβλητές

Μετά από τις μικρές αλλαγές στον Πίνακα Συμβόλων προχωρήσαμε στην δημιουργία καθολικών μεταβλητών και βοηθητικών συναρτήσεων που θα διευκολύνουν και θα κάνουν εν μέρη την παραγωγή του τελικού κώδικα.

1. Αρχικά δημιουργήσαμε μία καθολική μεταβλητή - μετρητή που θα κρατά πληροφορία για την μορφοποίηση του αρχείου .asm και θα μετρά σε ποια εντολή είμαστε. Ακόμα δημιουργήσαμε και μία καθολική μεταβλητή που θα αντιπροσωπεύει το αρχείο μας. Στην αρχή, πριν γίνει η δημιουργία του αρχείο η αρχικοποίηση του θα είναι None δηλαδή κενό.

```
# *Change for final code* global variables.
lineCounter = 0                             # a line counter for the file we will write.
final_file = None                           # the file we will write our final code in.
```

2. Μετά δημιουργήσαμε μία βοηθητική μέθοδο αναζήτησης οντοτήτων μέσα στον πίνακα deepCopySymbolTable που θα μοιάζει με την searchEntity μέθοδο του πίνακα συμβόλων αλλά θα έχει κάποιες διαφορές καθώς πλέον ο πίνακας μας έχει όλα τα επίπεδα όλες τις φορές που θα γίνει η αναζήτηση και έτσι δεν μας φθάνει να αναζητούμε μόνο την οντότητα με βάση το όνομα της. Επιπλέον, η συνάρτηση αυτή θα επιστρέφει την οντότητα που βρίσκει καθώς και το επίπεδο στο οποίο την βρήκε σε μορφή **[οντότητα,επίπεδο]**. Θα παίρνει ως ορίσματα το όνομα της οντότητας που αναζητούμε και το επίπεδο στο οποίο βρισκόμαστε καθώς δεν θα επιτρέπει αναζήτηση σε επίπεδα που βρίσκονται πιο πριν από αυτό, πράγμα που σημαίνει ότι δημιουργήθηκαν αργότερα από το ίδιο. Αν δεν μπορεί να βρει την οντότητα θα επιστρέφει το μήνυμα λάθους 22 (αναφέρεται στην εισαγωγή). Η μέθοδος θα έχει το όνομα **searchEntityInDeepCopy**.

```
def searchEntityInDeepCopy(name, currentScope):
    global deepCopySymbolTable
    entity = foundScope = None
    currentScopeIndex = deepCopySymbolTable.scopeList.index(currentScope)
    for i in deepCopySymbolTable.scopeList:                 # find the entity
        entity = i.searchEntity(name)
        if(entity != None) and (currentScopeIndex <= deepCopySymbolTable.scopeList.index(i)):
            foundScope = i
            break
    if(entity == None):                                 # if we don't find the entity -> error
        error(22,name)
    return [entity,foundScope]
```

3. Ακόμα δημιουργήσαμε την βοηθητική συνάρτηση **gnlvcode** που όπως είδαμε και στην θεωρία θα γράφει τελικό κώδικα που θα μας μετακινεί στο επίπεδο που βρίσκεται η οντότητα που της περάσαμε ως όρισμα. Η συνάρτηση αυτή θα έχει ως ορίσματα το όνομα της μεταβλητής που θέλουμε και το τωρινό μας επίπεδο για να μπορέσει να υπολογίσει την απόσταση από αυτό και να μας μετακινήσει ανάλογα. Ακόμα θα γυρνά τον αριθμό των επιπέδων που μας μετακινεί. Τέλος, ανάλογα τον αριθμό επιπέδων που θα πρέπει να μετακινηθεί θα εκτυπώνει και τον κατάλληλο τελικό κώδικα στο αρχείο .asm .

```
def gnlvcode(variable, currentScope):
    global deepCopySymbolTable,final_file,lineCounter
```

```
    search = searchEntityInDeepCopy(variable,currentScope)  # search the entity.
    entity = search[0]
    targetScope = search[1]
    targetScopeIndex = deepCopySymbolTable.scopeList.index(targetScope) # index of the target
    currentScopeIndex = deepCopySymbolTable.scopeList.index(currentScope)
    layersToClimb = targetScopeIndex - currentScopeIndex    # layers to climb up
    if(layersToClimb > 0):
        final_file.write("\tlw t0,-8(sp)\n")
        for i in range(0,layersToClimb-1):
            final_file.write("\tlw t0,-8(t0)\n")
        final_file.write("\taddi t0,t0,-"+str(entity.offset)+"\n")
    return layersToClimb
```

4. Μετά δημιουργήσαμε την **loadvr** όπως είδαμε και στην θεωρία μας. Εδώ πρέπει να πούμε ότι η συνάρτηση θα έχει ως πεδία την μεταβλητή/αριθμό που θέλουμε να αποθηκεύσουμε, τον καταχωρητή που θέλουμε να την αποθηκεύσουμε και το τωρινό μας επίπεδο.

```
def loadvr(sourceVariable, targetRegister, currentScope):
    global deepCopySymbolTable,final_file,lineCounter
    # digit
    if(sourceVariable.isnumeric()):
        final_file.write("\tli "+targetRegister+","+sourceVariable+"\n")
    else:
        search = searchEntityInDeepCopy(sourceVariable,currentScope)
        entity = search[0]
        foundScope = search[1]
        foundScopeIndex = deepCopySymbolTable.scopeList.index(foundScope)
        currentScopeIndex = deepCopySymbolTable.scopeList.index(currentScope)
        # local variable
        if(foundScopeIndex == currentScopeIndex and entity.dataType == "VAR"):
            layers = gnlvcode(sourceVariable,currentScope)
            if (layers <= 0):
                final_file.write("\tlw "+targetRegister+",-"+str(entity.offset)+"(sp)\n")
            else:
                final_file.write("\tlw "+targetRegister+",(t0)\n")
        # global variable
        elif(foundScopeIndex == len(deepCopySymbolTable.scopeList)-1 and entity.dataType ==
"VAR"):
            final_file.write("\tlw "+targetRegister+",-"+str(entity.offset)+"(gp)\n")
        # temporary variable
        elif (entity.dataType in "TEMP"):
            final_file.write("\tlw "+targetRegister+",-"+str(entity.offset)+"(sp)\n")
        # parameter that was set with "in"
        elif(entity.dataType == "FORMAL" and entity.mode == "CV"):
            layers = gnlvcode(sourceVariable,currentScope)
            if (layers <= 0):
                final_file.write("\tlw "+targetRegister+",-"+str(entity.offset)+"(sp)\n")
            else:
                final_file.write("\tlw "+targetRegister+",(t0)\n")
        # parameter that was set with "inout"
        elif(entity.dataType == "FORMAL" and entity.mode == "REF"):
            layers = gnlvcode(sourceVariable,currentScope)
            if (layers <= 0):
                final_file.write("\tlw t0,-"+str(entity.offset)+"(sp)\n")
                final_file.write("\tlw "+targetRegister+",(t0)\n")
            else:
                final_file.write("\tlw t0,(t0)\n")
                final_file.write("\tlw "+targetRegister+",(t0)\n")
```

5. Τέλος, δημιουργήσαμε και την **savevr** όπως είδαμε στην θεωρία μας. Εδώ πρέπει να πούμε ότι η συνάρτηση θα έχει ως πεδία την μεταβλητή/αριθμό που θέλουμε να αποθηκεύσουμε, τον καταχωρητή που θέλουμε να την αποθηκεύσουμε και το τωρινό μας επίπεδο.

```
def storevr(sourceRegister, targetVariable, currentScope):
    global deepCopySymbolTable,final_file,lineCounter
    search = searchEntityInDeepCopy(targetVariable,currentScope)
    entity = search[0]
    foundScope = search[1]
    foundScopeIndex = deepCopySymbolTable.scopeList.index(foundScope)
    currentScopeIndex = deepCopySymbolTable.scopeList.index(currentScope)
    # local variable
    if(foundScopeIndex == currentScopeIndex and entity.dataType == "VAR"):
        layers = gnlvcode(targetVariable,currentScope)
        if (layers <= 0):
            final_file.write("\tsw "+sourceRegister+",-"+str(entity.offset)+"(sp)\n")
        else:
            final_file.write("\tsw "+sourceRegister+",(t0)\n")
    # global variable
    elif(foundScopeIndex == len(deepCopySymbolTable.scopeList)-1 and entity.dataType == "VAR"):
        final_file.write("\tsw "+sourceRegister+",-"+str(entity.offset)+"(gp)\n")
    # temporary variable
    elif (entity.dataType in "TEMP"):
        final_file.write("\tsw "+sourceRegister+",-"+str(entity.offset)+"(sp)\n")
    # parameter that was set with "in"
    elif(entity.dataType == "FORMAL" and entity.mode == "CV"):
        layers = gnlvcode(targetVariable,currentScope)
        if (layers <= 0):
            final_file.write("\tsw "+sourceRegister+",-"+str(entity.offset)+"(sp)\n")
        else:
            final_file.write("\tsw "+sourceRegister+",(t0)\n")
    # parameter that was set with "inout"
    elif(entity.dataType == "FORMAL" and entity.mode == "REF"):
        layers = gnlvcode(targetVariable,currentScope)
        if (layers <= 0):
            final_file.write("\tlw t0,-"+str(entity.offset)+"(sp)\n")
            final_file.write("\tsw "+sourceRegister+",(t0)\n")
        else:
            final_file.write("\tlw t0,(t0)\n")
            final_file.write("\tsw "+sourceRegister+",(t0)\n")
```

Συνάρτηση παραγωγής Τελικού κώδικα.

Στην συνέχεια δημιουργήσαμε μία συνάρτηση με το όνομα **produceFinalCode** που θα δημιουργεί και θα γράφει στο αρχείο .asm τον τελικό μας κώδικα. Η συνάρτηση αυτή θα μελετά κάθε γραμμή του παραγώμενου ενδιάμεσου κώδικα (από την λίστα μας quadList) και θα πράττει ανάλογα σε κάθε περίπτωση δημιουργώντας κώδικα όπως είδαμε στο μάθημα αλλά και στην θεωρία που μας δόθηκε. Δεν υπάρχει νόημα να μπούμε σε περισσότερες λεπτομέρειες αφού ακολουθήσαμε πιστά τις οδηγίες που μας δόθηκαν για την δημιουργία κώδικα της RISC-V. Παρακάτω παραθέτουμε την συνάρτηση αυτή, η οποία περιέχει σχόλια που διαχωρίζουν την κάθε περίπτωση.

```
def produceFinalCode():
    global quadList,final_file,lineCounter,deepCopySymbolTable
    currentScope = deepCopySymbolTable.scopeList[0]
```

```
    parCounter = 0
    for i in range(0,len(quadList)):

        one = str(quadList[i][1][0])
        two = str(quadList[i][1][1])
        three = str(quadList[i][1][2])
        four = str(quadList[i][1][3])

        if(i == 0):                                         # start of the
program
            final_file.write("L"+str(lineCounter)+":\n\tj main\n")
            lineCounter = lineCounter + 1
        if(one == "begin_block"):                           # start of function or procedure
(including our main)
            if("main_" in two):
                final_file.write("\nmain:\n")
                final_file.write("L"+str(lineCounter)+":\n\taddi
sp,sp,"+str(currentScope.getOffset())+"\n\tmv gp,sp\n")
                lineCounter = lineCounter + 1
            else:
                final_file.write("\n"+two+":\n")
                final_file.write("L"+str(lineCounter)+":\n\tsw ra,-0(sp)\n")
                lineCounter = lineCounter + 1
        elif(one == "end_block"):                           # end of function or procedure
(including our main)
            currentScope = currentScope.nextScope           # moving to the nextScope since we
end the current function/procedure.
            if("main_" not in two):
                final_file.write("L"+str(lineCounter)+":\n\tlw ra,-0(sp)\n\tjr ra")
                lineCounter = lineCounter + 1
        elif(one == "halt"):                                # end of program
            final_file.write("L"+str(lineCounter)+":\n\tli a0,0\n\tli a7,93\n\tecall")
            lineCounter = lineCounter + 1
        elif(one in ["+","-","/","*",":="]):                # math and assignments
            final_file.write("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            if(three == "_"):                               # if we have assignment with 2
variables
                loadvr(two,"t1",currentScope)
                storevr("t1",four,currentScope)
            else:                                           # if we have assignment or
operation with 3 variables
                loadvr(two,"t1",currentScope)
                loadvr(three,"t2",currentScope)
                if("+" in one):                             # addition.
                    final_file.write("\tadd t1,t1,t2\n")
                elif("-" in one):                           # subtraction.
                    final_file.write("\tsub t1,t1,t2\n")
                elif("/" in one):                           # division.
                    final_file.write("\tdiv t1,t1,t2\n")
                elif("*" in one):                           # multiplication.
                    final_file.write("\tmul t1,t1,t2\n")
                storevr("t1",four,currentScope)
        elif(one == "jump"):                                # for jump.
            final_file.write("L"+str(lineCounter)+":\n\tj L"+four)
            lineCounter = lineCounter + 1
        elif(one in [">",">=","<","<=","<>","="]):          # logical operations
            final_file.write("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            loadvr(two,"t5",currentScope)
            loadvr(three,"t6",currentScope)
```

```
            if(one == ">"):
                final_file.write("\tbgt t5,t6,L"+four+"\n") # bgt
            elif(one == ">="):
                final_file.write("\tbge t5,t6,L"+four+"\n") # bge
            elif(one == "<"):
                final_file.write("\tblt t5,t6,L"+four+"\n") # blt
            elif(one == "<="):
                final_file.write("\tble t5,t6,L"+four+"\n") # ble
            elif(one == "<>"):
                final_file.write("\tbne t5,t6,L"+four+"\n") # bne
            elif(one == "="):
                final_file.write("\tbeq t5,t6,L"+four+"\n") # beq
        elif(one == "out"):                                 # printing
            final_file.write("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            loadvr(two,"t1",currentScope)
            final_file.write("\tli s1,1\n\tli a0,"+two+"\n\tsyscall\n")
        elif(one == "in"):                                  # inputs
            final_file("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            loadvr(two,"t1",currentScope)
            final_file("\tli s1,5\n\tsyscall\n")
        elif(one == "ret"):                                 # returns
            final_file.write("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            loadvr(two,"t1",currentScope)
            final_file.write("\tlw t0,-8(sp)\n\tsw t1,(t0)\n")
        elif(one == "par"):                                 # parameters
            final_file.write("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            if(parCounter == 0):
                final_file.write("\taddi fp,sp,"+str(currentScope.getOffset())+"\n")
            parCounter = parCounter + 1
            d = 12 + (parCounter - 1) * 4
            if(three == "cv"): #in
                loadvr(two,"t1",currentScope)
                final_file.write("\tsw t1,-"+str(d)+"(fp)\n")
            elif(three == "ret"): #return
                search = searchEntityInDeepCopy(two,currentScope)
                entity = search[0]
                targetScope = search[1]
                if(currentScope.nextScope == targetScope.nextScope):
                    if(entity.dataType == "VAR") or (entity.dataType == "FORMAL" and
entity.mode == "CV"):
                        final_file.write("\tadd t0,sp,-"+str(entity.offset)+"\n")
                        final_file.write("\tadd t0,sp,-"+str(d)+"(fp)\n")
                    elif(entity.dataType == "FORMAL" and entity.mode == "REF"):
                        final_file.write("\tlw t0,-("+str(entity.offset)+"(sp)\n")
                        final_file.write("\tsw t0,-"+str(d)+"(fp)\n")
                else:
                    if(entity.dataType == "VAR") or (entity.dataType == "FORMAL" and
entity.mode == "CV"):
                        gnlvcode(two,currentScope)
                        final_file("\tsw t0,-"+str(d)+"(fp)\n")
                    elif(entity.dataType == "FORMAL" and entity.mode == "REF"):
                        gnlvcode(two,currentScope)
                        final_file("\tlw t0,(t0)\n")
                        final_file("\tsw t0,-"+str(d)+"(fp)\n")
            elif(three == "ref"): #inout
                search = searchEntityInDeepCopy(two,currentScope)
                entity = search[0]
```

```
                targetScope = search[1]
                final_file.write("\taddi t0,sp,-"+str(entity.offset)+"\n")
                final_file.write("\tsw t0,-"+str(entity.offset)+"(fp)\n")
        elif(one == "call"):                            # function or procedure calls.
            final_file.write("L"+str(lineCounter)+":\n")
            lineCounter = lineCounter + 1
            search = searchEntityInDeepCopy(two,currentScope)
            entity = search[0]
            targetScopeIndex = deepCopySymbolTable.scopeList.index(search[1])-1
            targetScope = deepCopySymbolTable.scopeList[targetScopeIndex]
            if(len(entity.formalParameters) == 0):
                final_file.write("\taddi fp,sp,"+str(entity.frameLength)+"\n")
            if(currentScope.nextScope == targetScope.nextScope):
                final_file.write("\tlw t0,-4(sp)\n\tsw t0,-4(fp)\n")
            else:
                final_file.write("\tsw sp,-4(fp)\n")
            final_file.write("\taddi sp,sp,"+str(entity.frameLength)+"\n")
            final_file.write("\tjal "+two+"\n")
            final_file.write("\taddi sp,sp,-"+str(entity.frameLength)+"\n")
```

Αλλαγές στην συνάρτηση syntaxAnalyser για την εκτύπωση όλων των αρχείων

Τέλος, για την ομαλή και πιο οργανωμένη σε σειρά λειτουργία όλων των φάσεων που προαναφέραμε, μαζί και του τελικού κώδικα προχωρήσαμε στην αλλαγή της συνάρτησης syntaxAnalyser με σκοπό να καλούμε μόνο αυτή για να λειτουργήσει ο μεταφραστής μας.

```
def syntaxAnalyser():
    global currentToken,table_file,final_file,deepCopySymbolTable
    currentToken = lexicalAnalyser()                            # getting the first token of
the file.
    try:                                                        # *Change for final code*
opening the .symb AND .asm file for writing
        table_file = open("pinakas.symb","w")
        final_file = open("final.asm","w")
    except FileExistsError:
        table_file = open("pinakas.symb","x")
        final_file = open("final.asm","x")
    program()                                                   # starting the syntax
recognition.
    printList(quadList)                                         # *Change for middleware*
    table_file.close()                                          # *Change for symbol table*
    produceFinalCode()                                          # *Change for final code*
    final_file.close()                                          # *Change for final code*
    print("\n\tCompilation successfully completed.Yay.\n")      # if all goes well.
```

Προσθέσαμε την αρχικοποίηση του αρχείου για τον τελικό κώδικα “final_file” καθώς και την δημιουργία του τελικού κώδικα και μεταφέραμε την δημιουργία και των άλλων αρχείων των ενδιάμεσων φάσεων εδώ. Τέλος το πρόγραμμα θα τρέχει μόνο με την κλήση της συνάρτησης αυτής όπως φαίνεται παρακάτω.

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Program Run

syntaxAnalyser()                                                # syntax analyser start.
```