



# STARLET COMPILER 2019

Δημόπουλος Γεώργιος Α.Μ. 2964

Λάμπρος Νικόλας Α.Μ. 2922

---

# Εισαγωγή

Ο μεταφραστής είναι ένα πρόγραμμα λογισμικού, το οποίο μετατρέπει πηγαίο κώδικα υψηλού επιπέδου, που γράφεται από έναν προγραμματιστή σε κάποια γλώσσα προγραμματισμού υψηλού επιπέδου, σε κώδικα χαμηλού επιπέδου (δυαδικός κώδικας) στη γλώσσα μηχανής, όπου ο τελευταίος μπορεί να γίνει κατανοητός από τον επεξεργαστή. Η διαδικασία αυτής της μετατροπής κώδικα υψηλού επιπέδου σε γλώσσα μηχανής είναι γνωστή ως “compilation”.

Στην παρούσα αναφορά κάνουμε λόγο για ένα μεταφραστή, ο οποίος δημιουργήθηκε για τη γλώσσα προγραμματισμού Starlet. Προκειμένου να έχουμε μία επιτυχή “compilation” διαδικασία, τη χωρίσαμε σε τρεις φάσεις, κάθε μια από τις οποίες περιγράφεται από διαφορετικές συναρτήσεις. Αναφορικά οι φάσεις αυτές είναι οι:

1. Λεκτική – Συντακτική Ανάλυση
2. Παραγωγή Ενδιάμεσου Κώδικα
3. Παραγωγή Τελικού Κώδικα

## 1. Λεκτική – Συντακτική Ανάλυση

Στην συγκεκριμένη φάση χρησιμοποιήσαμε δύο βασικά εργαλεία, που φαίνονται παρακάτω:

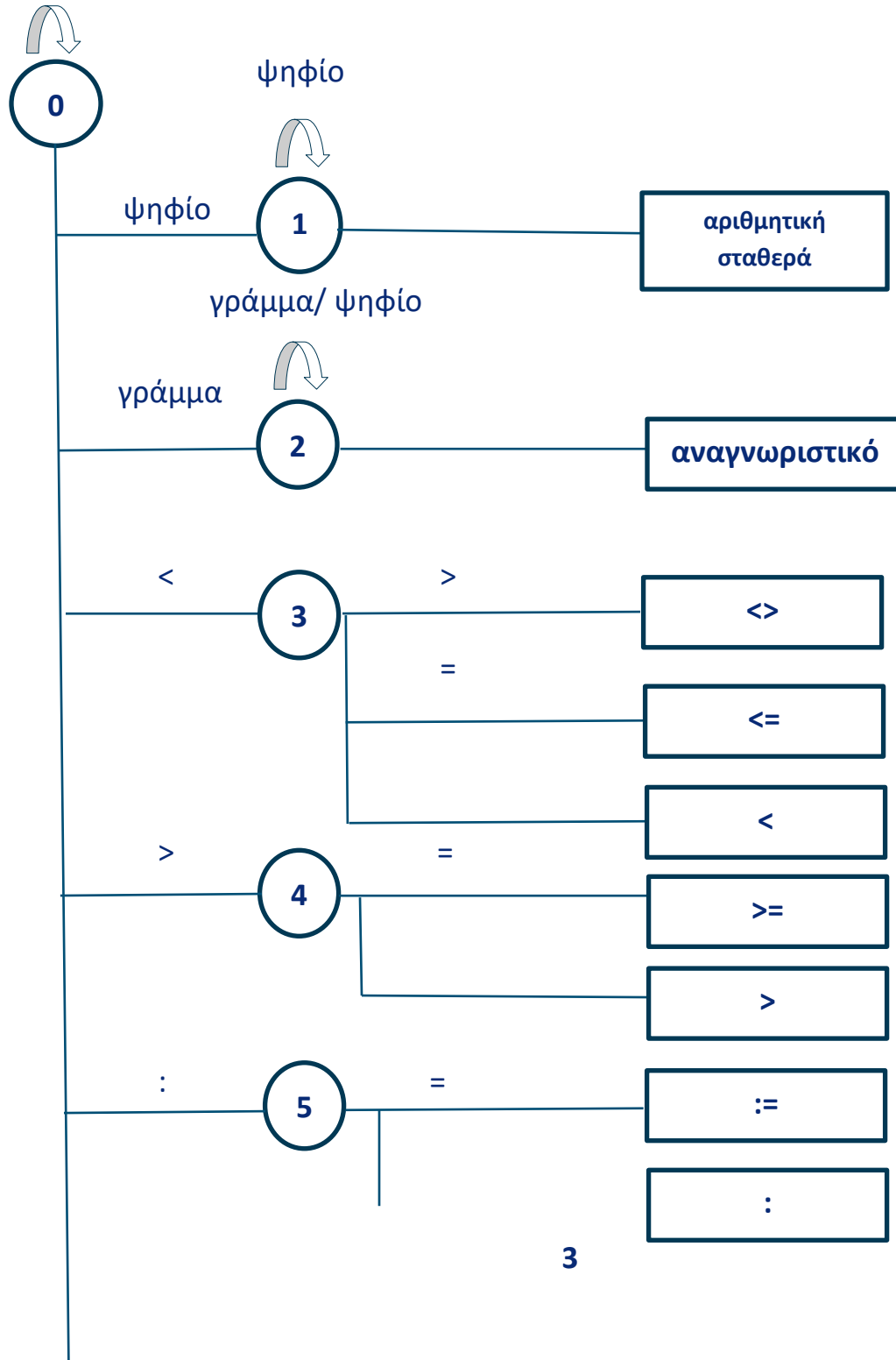
- Λεκτικός Αναλυτής
- Συντακτικός Αναλυτής

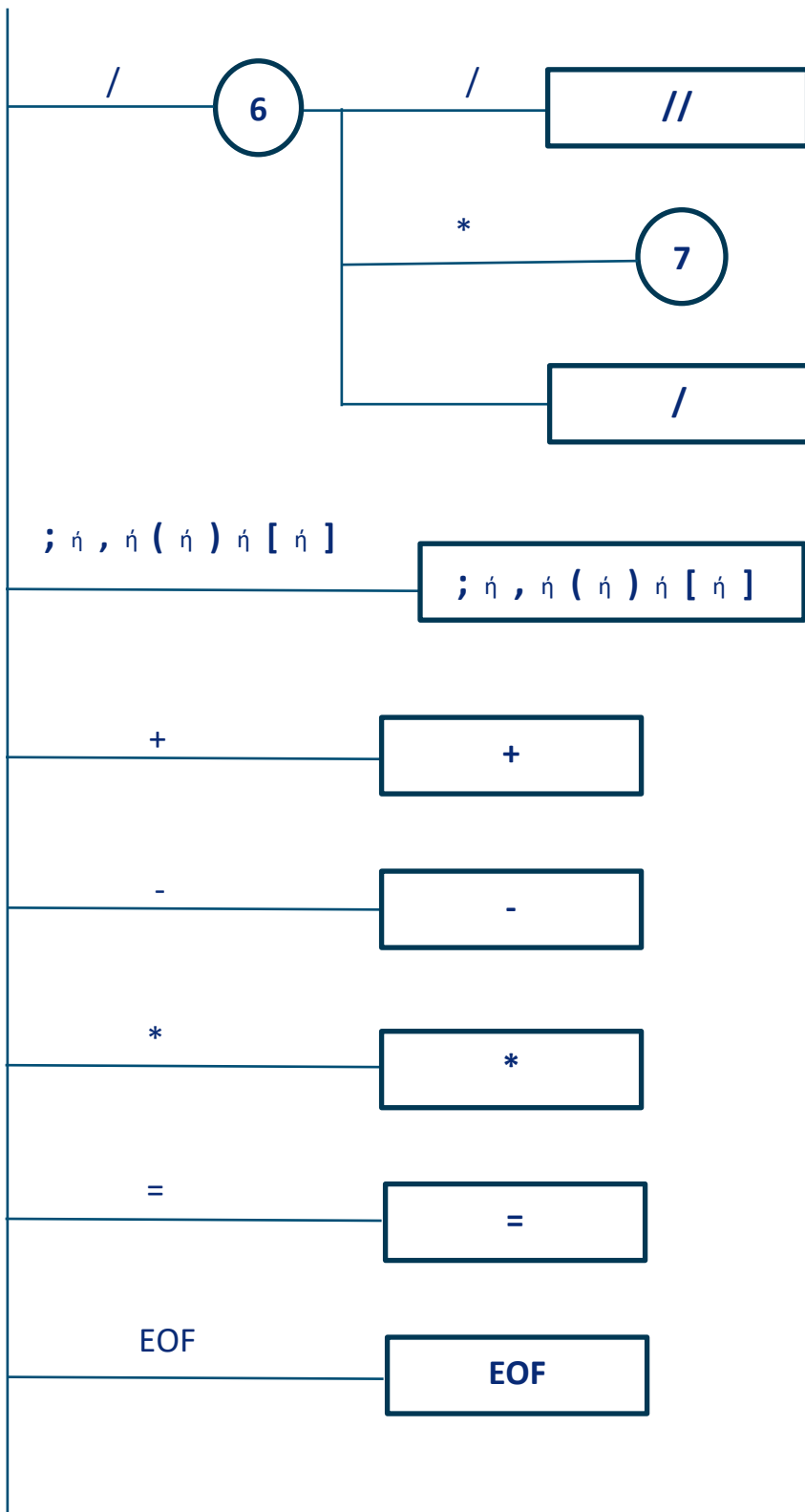
Γενικά η λεκτική ανάλυση είναι ένα εργαλείο το οποίο ξεκινώντας από το αρχικό πρόγραμμα και «διαβάζοντάς» το χαρακτήρα – χαρακτήρα, παράγει λεκτικές μονάδες καθώς επίσης και διαγνωστικά μηνύματα.

Όσον αφορά το **λεκτικό αναλυτή**, αυτός αποτελεί μια συνάρτηση του συντακτικού αναλυτή. Εσωτερικά λειτουργεί σαν ένα αυτόματο καταστάσεων. Ξεκινάμε δηλαδή

από μία αρχική κατάσταση, και εισάγοντας κάθε φορά τον επόμενο χαρακτήρα, αλλάζουμε κατάσταση, έως ότου φτάσουμε σε μια τελική κατάσταση. Όταν φτάσουμε στην τελική κατάσταση επιστρέφουμε στο συντακτικό αναλυτή την επόμενη λεκτική μονάδα και έναν ακέραιο ο οποίος τη χαρακτηρίζει. Το αυτόματο καταστάσεων που περιγράφει την λειτουργία του λεκτικού μας αναλυτή φαίνεται παρακάτω.

Λευκός χαρακτήρας





---

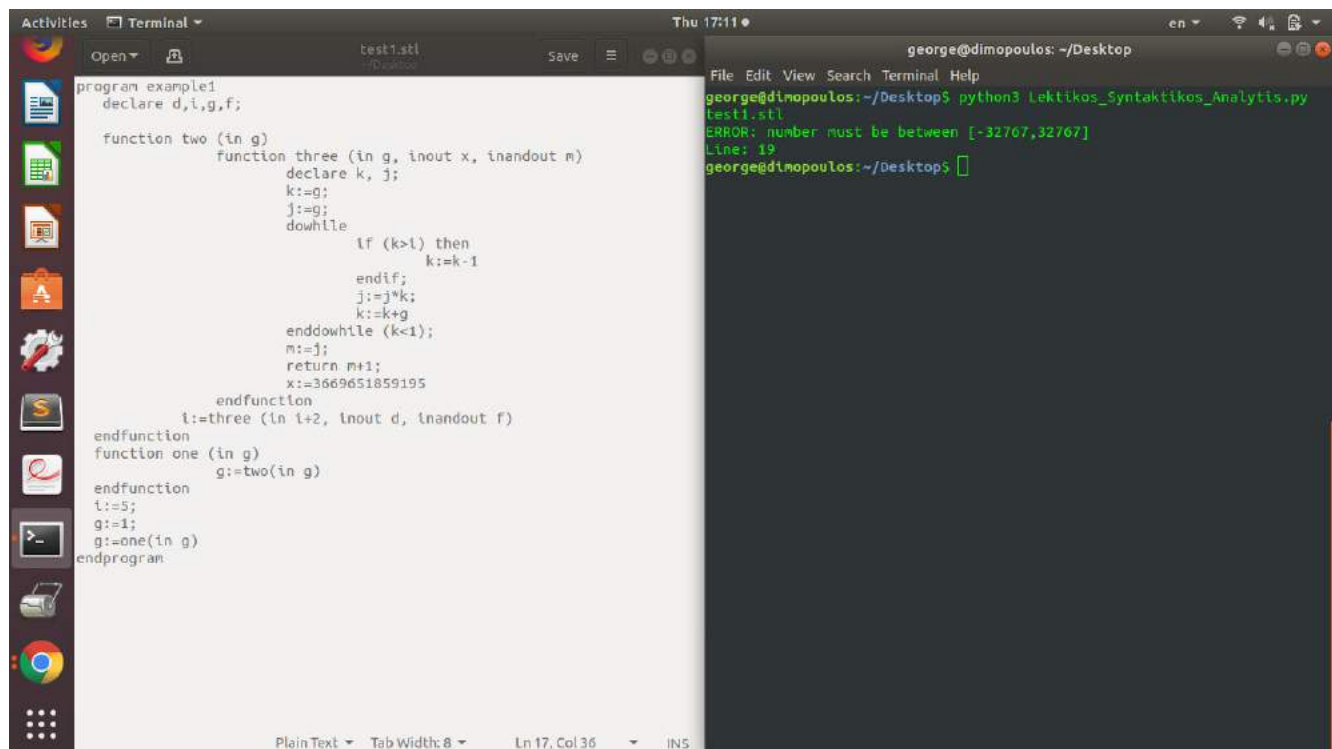
error

Όπως εξηγήσαμε προηγουμένως ο λεκτικός αναλυτής λειτουργεί σαν αυτόματο καταστάσεων. Συγκεκριμένα, για τη **σχεδίασή** του φτιάξαμε έντεκα (11) διαφορετικές καταστάσεις. Τελικές είναι οι καταστάσεις OK, EOF (EndOfFile) και error, στις οποίες θα αναφερθούμε στην συνέχεια. Οι υπόλοιπες «ενδιάμεσες» καταστάσεις μας βοηθούν να καταλάβουμε σε ποιο είδος λεκτικής μονάδας ανήκει ο κάθε χαρακτήρας που διάβασε. Υπάρχουν συνολικά οκτώ (8) διαφορετικά είδη λεκτικών μονάδων. Κάθε είδος αντιστοιχεί σε μια κατάσταση ξεχωριστά. Συγκεκριμένα στην κατάσταση (state) 0 αντιστοιχούν σύμβολα αριθμητικών πράξεων ( + , - , \* , / ), διαχωριστές ( ; , : , , ) και σύμβολα ομαδοποίησης ( [ , ( , ) , ] ). Στην κατάσταση (state) 1 ανήκουν αριθμητικά ψηφία, που δεν επιτρέπεται να έχουν απόλυτη τιμή μεγαλύτερη του 32767. Στην κατάσταση (state) 2 ανήκουν τα αλφαριθμητικά, όπου το μέγεθός τους δεν επιτρέπεται να ξεπερνά τους 30 χαρακτήρες. Αντίστοιχα στις καταστάσεις 3 και 4 αντιστοιχούν οι τελεστές συσχέτισης < , > και τα παράγωγα τους (<=, <>, >=). Συνεχίζοντας, στην κατάσταση 5 έχουμε το σύμβολο ανάθεσης := . Τέλος, στις καταστάσεις 6 και 7 έχουμε τα σχόλια. Ειδικότερα στην κατάσταση 6 έχουμε τα σχόλια μιας γραμμής, τα οποία ξεκινούν με // και ομοίως στην κατάσταση 7 ανήκουν τα σχόλια που τοποθετούνται ανάμεσα από τα σύμβολα /\* και \*/. Στο σημείο αυτό να αναφέρουμε, ότι δεν επιτρέπονται εμφωλευμένα σχόλια στην συγκεκριμένη γλώσσα. Φτάνοντας λοιπόν σε μία τελική κατάσταση, ο λεκτικός αναλυτής μεταφέρει, τα δεδομένα που έχει συλλέξει στο συντακτικό.

Όταν θέσουμε σε **λειτουργία** το πρόγραμμά μας, αρχικά ανοίγει το αρχείο Starlet που δώσαμε σαν όρισμα από το τερματικό και καλεί το συντακτικό αναλυτή. Εκείνος με την σειρά του καλεί τον λεκτικό, ο οποίος πηγαίνει στην αρχή του κώδικα και διαβάζει τον πρώτο χαρακτήρα του αρχείου. Ανάλογα με την κατηγορία του χαρακτήρα, που διάβασε, πηγαίνει στην αντίστοιχη κατάσταση, όπως παρουσιάσαμε προηγουμένως στο αυτόματο καταστάσεων. Όταν διαβάσει κενό χαρακτήρα, μεταβαίνει σε μια τερματική κατάσταση, στέλνει τα κατάλληλα δεδομένα στον

συντακτικό αναλυτή και τερματίζει την λειτουργία του. Συγκεκριμένα, όταν βρεθεί στην κατάσταση OK, σημαίνει ότι αναγνώρισε επιτυχώς την λεκτική μονάδα και την επιστρέφει στον συντακτικό αναλυτή μαζί με τον ακέραιο αριθμό που την χαρακτηρίζει. Αντιθέτως αν βρεθεί στην κατάσταση error, ενημερώνει το χρήστη ότι ανίχνευσε κάποιο σφάλμα και επιστρέφει στον συντακτικό την λεκτική μονάδα error και τον ακέραιο -1. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου ο λεκτικός αναλυτής φτάσει στο τέλος του αρχείου. Τότε πηγαίνει στην κατάσταση EOF, αυτό σημαίνει ότι έλεγξε όλο το αρχείο, και επιστρέφει στον συντακτικό αναλυτή το ανάλογο μήνυμα.

Παρακάτω παραθέτουμε μερικά παραδείγματα από την λειτουργία του λεκτικού:

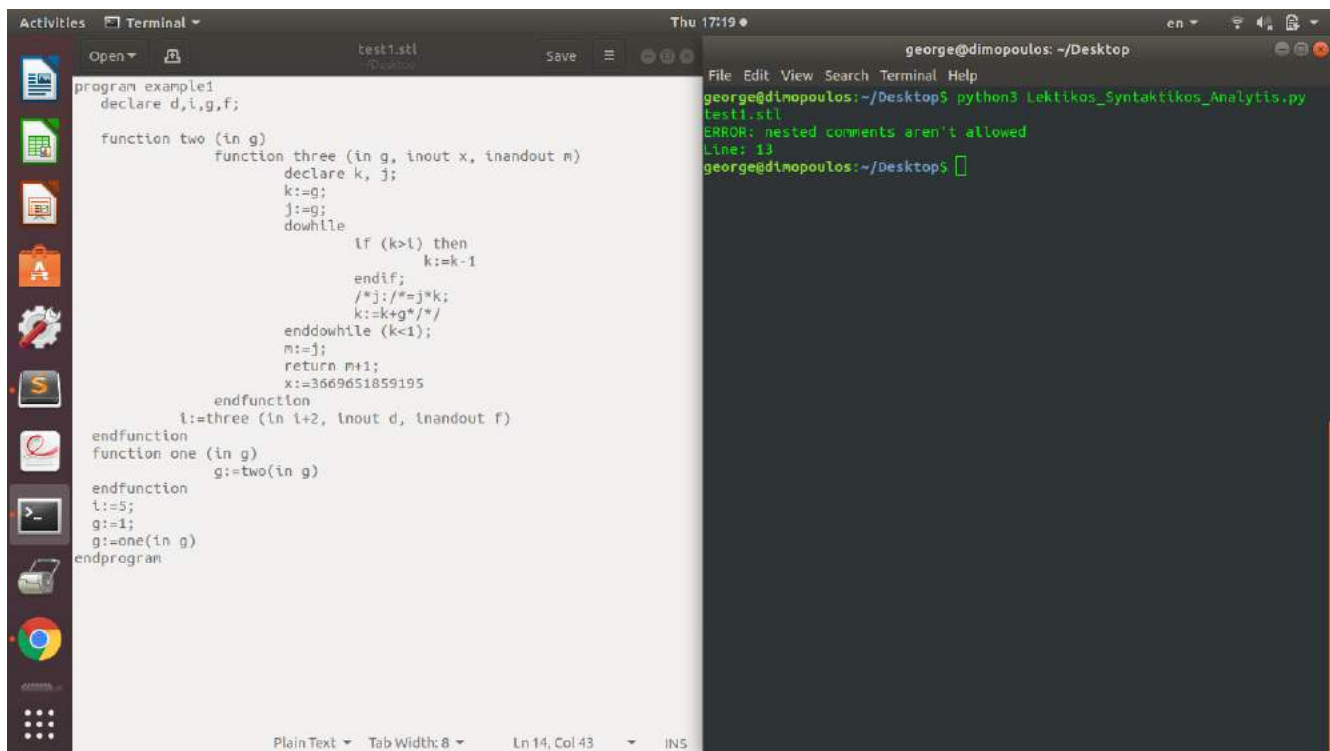


```
Activities Terminal
Open test1.stl Save
program example1
  declare d,i,g,f;

  function two (in g)
    function three (in g, inout x, inandout m)
      declare k, j;
      k:=g;
      j:=g;
      dowhile
        if (k>1) then
          k:=k-1
        endif;
        j:=j*k;
        k:=k+g
      enddowhile (k<1);
      m:=j;
      return m+1;
      x:=3669651859195
    endfunction
    i:=three (in i+2, inout d, inandout f)
  endfunction
  function one (in g)
    g:=two(in g)
  endfunction
  i:=5;
  g:=1;
  g:=one(in g)
endprogram

george@dimopoulos: ~/Desktop
File Edit View Search Terminal Help
george@dimopoulos:~/Desktop$ python3 Lektikos_Syntaktikos_Analytis.py
test1.stl
ERROR: number must be between [-32767,32767]
Line: 19
george@dimopoulos:~/Desktop$
```

Στην παραπάνω εικόνα φαίνεται το μήνυμα που εμφανίζει ο λεκτικός αναλυτής όταν εισάγουμε αριθμό με μεγαλύτερη απόλυτη τιμή του 3267.

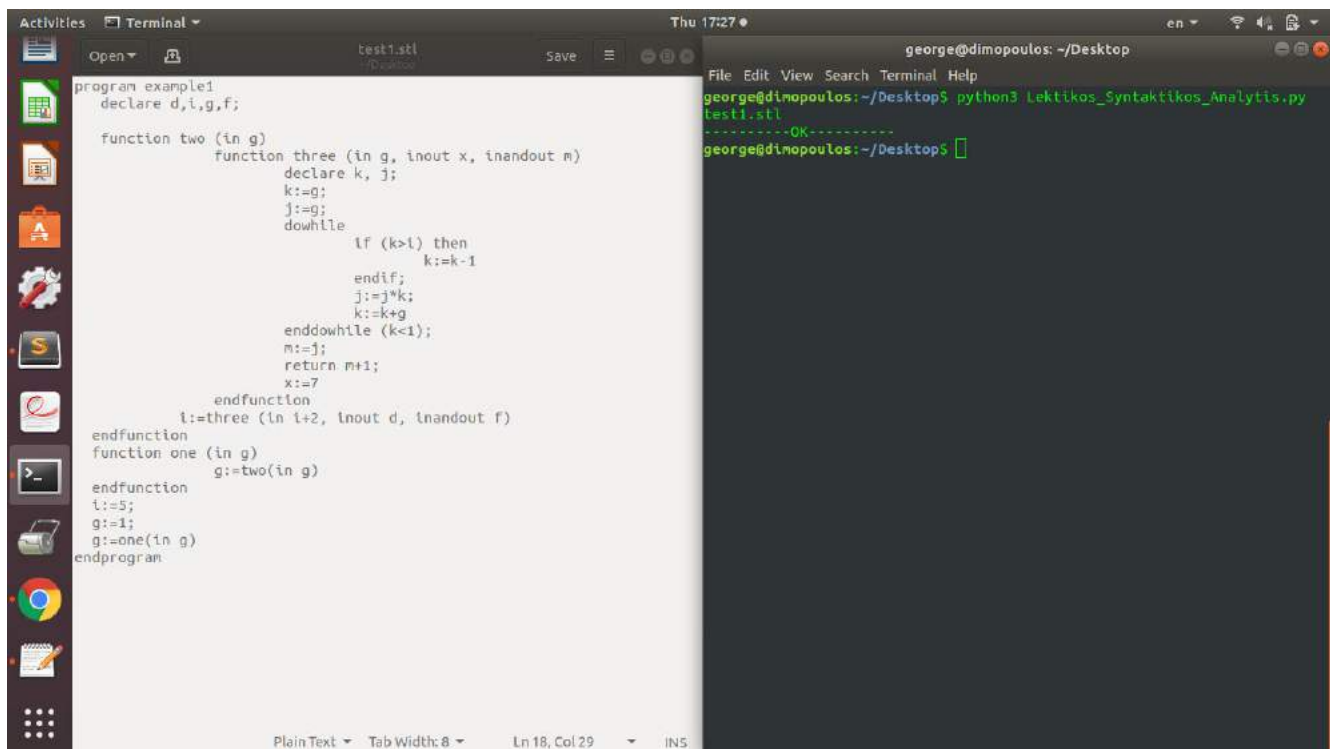


```
program example1
  declare d,i,g,f;

  function two (in g)
    function three (in g, inout x, inout m)
      declare k, j;
      k:=g;
      j:=g;
      dowhile
        if (k>1) then
          k:=k-1
        endif;
        /*j:=j*k;
        k:=k+g*/
      enddowhile (k<1);
      m:=j;
      return m+1;
      x:=3669651859195
    endfunction
    i:=three (in i+2, inout d, inout f)
  endfunction
  function one (in g)
    g:=two(in g)
  endfunction
  i:=5;
  g:=1;
  g:=one(in g)
endprogram
```

```
george@dimopoulos: ~/Desktop
george@dimopoulos:~/Desktop$ python3 Lektikos_Syntaktikos_Analytis.py
test1.stl
ERROR: nested comments aren't allowed
Line: 13
george@dimopoulos:~/Desktop$
```

Στην παραπάνω εικόνα βλέπουμε το μήνυμα που εμφανίζει ο λεκτικός αναλυτής όταν βρει εμφωλευμένα σχόλια.



```
program example1
  declare d,i,g,f;

  function two (in g)
    function three (in g, inout x, inout m)
      declare k, j;
      k:=g;
      j:=g;
      dowhile
        if (k>1) then
          k:=k-1
        endif;
        j:=j*k;
        k:=k+g
      enddowhile (k<1);
      m:=j;
      return m+1;
      x:=7
    endfunction
    i:=three (in i+2, inout d, inout f)
  endfunction
  function one (in g)
    g:=two(in g)
  endfunction
  i:=5;
  g:=1;
  g:=one(in g)
endprogram
```

```
george@dimopoulos: ~/Desktop
george@dimopoulos:~/Desktop$ python3 Lektikos_Syntaktikos_Analytis.py
test1.stl
ERROR: nested comments aren't allowed
Line: 18
george@dimopoulos:~/Desktop$
```

---

Αντίστοιχα ο **συντακτικός αναλυτής**, είναι μια συνάρτηση, που υλοποιεί τη συντακτική ανάλυση μιας συγκεκριμένης γλώσσας. Συγκεκριμένα δέχεται ως είσοδο μια ακολουθία λεκτικών μονάδων ενός προγράμματος, με την κλήση του λεκτικού αναλυτή και ελέγχει αν το πρόγραμμα είναι σύμφωνο με την γραμματική της γλώσσας που υλοποιεί.

Προκειμένου να πετύχουμε μια αποτελεσματική συντακτική ανάλυση, δημιουργήσαμε μια σειρά από συναρτήσεις, που είναι ανάλογες των κανόνων που απαρτίζουν την γραμματική της γλώσσας Starlet. Αναλυτικότερα, οι συναρτήσεις που δημιουργήθηκαν είναι οι εξής:

- ✓ **program:** Αντιστοιχεί στο κεντρικό πρόγραμμα. Αν η λεκτική μονάδα έχει την λέξη program τότε καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η επόμενη λεκτική μονάδα δεν είναι κάποιο id τότε τυπώνουμε σφάλμα. Διαφορετικά καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε το **block**. Αν η λεκτική μονάδα δεν έχει το endprogram τότε τυπώνουμε το αντίστοιχο σφάλμα.
- ✓ **block:** Αντιστοιχεί σε ένα μπλοκ προγράμματος. Με το που μπαίνει στο **block** εκτελούμε τους κανόνες **declarations, subprograms, statements**.
- ✓ **declarations:** Αντιστοιχεί στις αρχικοποιήσεις. Αν η λεκτική μονάδα, έχει την λέξη declare τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **varlist**. Αν η λεκτική μονάδα έχει το ' ; ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά τυπώνουμε το αντίστοιχο μήνυμα.
- ✓ **varlist:** Αν η λεκτική μονάδα είναι κάποιο id τότε καταναλώνουμε την επόμενη λεκτική μονάδα. Τώρα όσο η λεκτική μονάδα είναι το ' , ' καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα είναι κάποιο id τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά τυπώνουμε το αντίστοιχο μήνυμα.



- 
- ✓ **subprograms:** Αντιστοιχεί στις συναρτήσεις του προγράμματος. Όσο η λεκτική μονάδα έχει την λέξη function εκτελούμε τον κανόνα **subprogram**.
  - ✓ **subprogram:** Αντιστοιχεί σε συνάρτηση. Αν η λεκτική μονάδα έχει την λέξη function τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει κάποιο id τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **funcbody**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει την λέξη endfunction τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.
  - ✓ **funcbody:** Αντιστοιχεί στον “κορμό” της συνάρτησης. Με το που μπαίνει στο **funcbody** εκτελούμε τους κανόνες **formalpars,block**.
  - ✓ **formalpars:** Αντιστοιχεί μαζί με τις **formalparlist, formalparitem** στα ορίσματα μιας συνάρτησης. Αν η λεκτική μονάδα έχει το ‘ ( ’ καταναλώνουμε την επόμενη λεκτική μονάδα διαφορετικά εκτυπώνουμε το αντίστοιχο μήνυμα. Μετά εκτελούμε τον κανόνα **formalparlist**. Αν η λεκτική μονάδα τώρα έχει το ‘ ) ’ καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε το αντίστοιχο μήνυμα.
  - ✓ **formalparlist:** Εκτελούμε τον κανόνα **formalparitem**. Μετά ενόσω η λεκτική μονάδα έχει μέσα το ‘ , ’ καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **formalparitem**.
  - ✓ **formalparitem:** Αν μπούμε στον κανόνα **formalparitem** τότε η λεκτική μονάδα θα έχει είτε το in είτε το inout είτε το inandout. Αν η λεκτική μονάδα έχει μέσα το in τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει κάποιο id καταναλώνουμε την επόμενη λεκτική μονάδα διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν η λεκτική μονάδα έχει μέσα το inout τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει κάποιο id καταναλώνουμε την επόμενη λεκτική

---

μονάδα διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν η λεκτική μονάδα έχει μέσα το inandout τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει κάποιο id καταναλώνουμε την επόμενη λεκτική μονάδα διαφορετικά εκτυπώνουμε σχετικό μήνυμα.

- ✓ **statements:** Αντιστοιχεί στις εκφράσεις. Συγκεκριμένα υπάρχουν διάφορα είδη εκφράσεων, τα οποία θα αναλύσουμε παρακάτω. Εκτελούμε τον κανόνα **statement**. Μετά ενόσω η λεκτική μονάδα έχει μέσα το ' ; ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statement**.
- ✓ **statement:** Αντιστοιχεί σε μια έκφραση. Αν η λεκτική μονάδα έχει μέσα κάποιο id τότε εκτελώ τον κανόνα **assignment\_stat**. Αν η λεκτική μονάδα έχει την λέξη if εκτελώ τον κανόνα **if\_stat**. Αν η λεκτική μονάδα έχει την λέξη while εκτελώ τον κανόνα **while\_stat**. Αν η λεκτική μονάδα έχει την λέξη dowhile εκτελώ τον κανόνα **do\_while\_stat**. Αν η λεκτική μονάδα έχει την λέξη loop εκτελώ τον κανόνα **loop\_stat**. Αν η λεκτική μονάδα έχει την λέξη exit εκτελώ τον κανόνα **exit\_stat**. Αν η λεκτική μονάδα έχει την λέξη forcase εκτελώ τον κανόνα **forcase\_stat**. Αν η λεκτική μονάδα έχει την λέξη incase εκτελώ τον κανόνα **incase\_stat**. Αν η λεκτική μονάδα έχει την λέξη return εκτελώ τον κανόνα **return\_stat**. Αν η λεκτική μονάδα έχει την λέξη input εκτελώ τον κανόνα **input\_stat**. Αν η λεκτική μονάδα έχει την λέξη print εκτελώ τον κανόνα **print\_stat**.
- ✓ **assignment\_stat:** Αντιστοιχεί στην ανάθεση τιμής. Με το που μπαίνουμε σε αυτό τον κανόνα καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα είναι ' := ' τότε παίρνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **expression**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.
- ✓ **if\_stat:** Αντιστοιχεί στην συνθήκη if. Με το που μπαίνουμε στον κανόνα αυτό καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το ' ( ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα

---

**condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Έπειτα αν η επόμενη λεκτική μονάδα έχει το ' ) ' καταναλώνουμε την επόμενη λεκτική μονάδα διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Μετά αν η λεκτική μονάδα έχει την λέξη then τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τους κανόνες **statements**, **elsepart**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τελειώνουμε με την λέξη endif τότε καταναλώνουμε την επόμενη λεκτική μονάδα διαφορετικά εμφανίζουμε σχετικό μήνυμα.

- ✓ **eslepart**: Αν η λεκτική μονάδα έχει την λέξη else τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statements**.
- ✓ **while\_stat**: Αντιστοιχεί στην δομή επανάληψης while. Με το που μπαίνουμε στον κανόνα αυτό καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το ' ( ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει ' ) ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statements**. Τώρα αν τελειώσουμε με την λέξη endwhile καταναλώνουμε την επόμενη λεκτική μονάδα διαφορετικά εκτυπώνουμε σχετικό μήνυμα.
- ✓ **do\_while\_stat**: Αντιστοιχεί στην δομή επανάληψης do while. Με το που μπαίνουμε στον κανόνα αυτό καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statements**. Αν η λεκτική μονάδα έχει την λέξη enddowhile τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν η λεκτική μονάδα τώρα έχει ' ( ' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Τέλος άμα η λεκτική μονάδα έχει ' ) ' καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.
- ✓ **loop\_stat**: Αντιστοιχεί στην δομή επανάληψης loop. Με το που μπαίνουμε σε αυτό τον κανόνα καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statements**. Αν η λεκτική μονάδα έχει την λέξη endloop τότε

---

καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.

- ✓ **exit\_stat:** Όταν μπορούμε σε αυτό τον κανόνα απλά καταναλώνουμε την επόμενη λεκτική μονάδα.
- ✓ **incase\_stat:** Αντιστοιχεί στην δομή επανάληψης incase. Με το που μπαίνουμε σε αυτό τον κανόνα καταναλώνουμε την επόμενη λεκτική μονάδα. Τώρα όσο η λεκτική μονάδα έχει την λέξη when καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα τώρα έχει το ' ( ' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει το ' ) ' καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν η λεκτική μονάδα έχει ' : ' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statements**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Τέλος αν η λεκτική μονάδα αυτή την στιγμή έχει την λέξη endincase καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.
- ✓ **return\_stat:** Όταν μπορούμε στον κανόνα αυτό καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **expression**.
- ✓ **print\_stat:** Όταν μπορούμε στον κανόνα αυτό καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **expression**.
- ✓ **input\_stat:** Με το που μπαίνουμε στον κανόνα αυτό καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει κάποιο id τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εμφανίζουμε σχετικό μήνυμα.
- ✓ **actualpars:** Αν η λεκτική μονάδα έχει ' ( ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **actualparlist**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Τώρα αν η λεκτική μονάδα έχει ' ) '

---

καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.

- ✓ **actualparlist:** Όταν μπορούμε σε αυτό τον κανόνα εκτελούμε τον κανόνα **actualparitem**. Τώρα όσο η λεκτική μονάδα έχει ' , ' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **actualparitem**.
- ✓ **actualparitem:** Όταν μπαίνουμε στον κανόνα αυτό περιμένουμε η λεκτική μονάδα να έχει μία από τις λέξεις: in, inout, inandout. Αν η λεκτική μονάδα έχει την λέξη in τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **expression**. Αν η λεκτική μονάδα έχει την λέξη inout καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα τώρα έχει κάποιο id καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν η λεκτική μονάδα έχει την λέξη inandout καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα τώρα έχει κάποιο id καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.
- ✓ **condition:** Αντιστοιχεί στις συνθήκες, που ενώνονται με or . Όταν μπορούμε στον κανόνα αυτό εκτελούμε τον κανόνα **boolterm**. Ενόσω τώρα η λεκτική μονάδα έχει την λέξη or καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **boolterm**.
- ✓ **boolterm:** Αντιστοιχεί στα στοιχεία της συνθήκης, που ενώνονται με and. Όταν μπορούμε στον κανόνα αυτό εκτελούμε τον κανόνα **boolfactor**. Ενόσω τώρα η λεκτική μονάδα έχει την λέξη and καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **boolfactor**.
- ✓ **boolfactor:** Αντιστοιχεί στα στοιχεία της συνθήκης . Όταν μπαίνουμε σε αυτό τον κανόνα περιμένουμε να έχει ή την λέξη not ή το σύμβολο ' [ ' ή να εκτελεστούν κάποιοι κανόνες. Άρα η λεκτική μονάδα έχει την λέξη not τότε καταναλώνουμε την επόμενη λεκτική μονάδα. Αν τώρα η λεκτική μονάδα έχει ' [ ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα

---

**condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει το ' ] ' εκτελούμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα είμασταν στην δεύτερη περίπτωση που η λεκτική μονάδα είχε ' [ ' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει ' ] ' καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα είμαστε στην τρίτη περίπτωση απλά εκτελούμε τους κανόνες: **expression, relation\_oper, expression**.

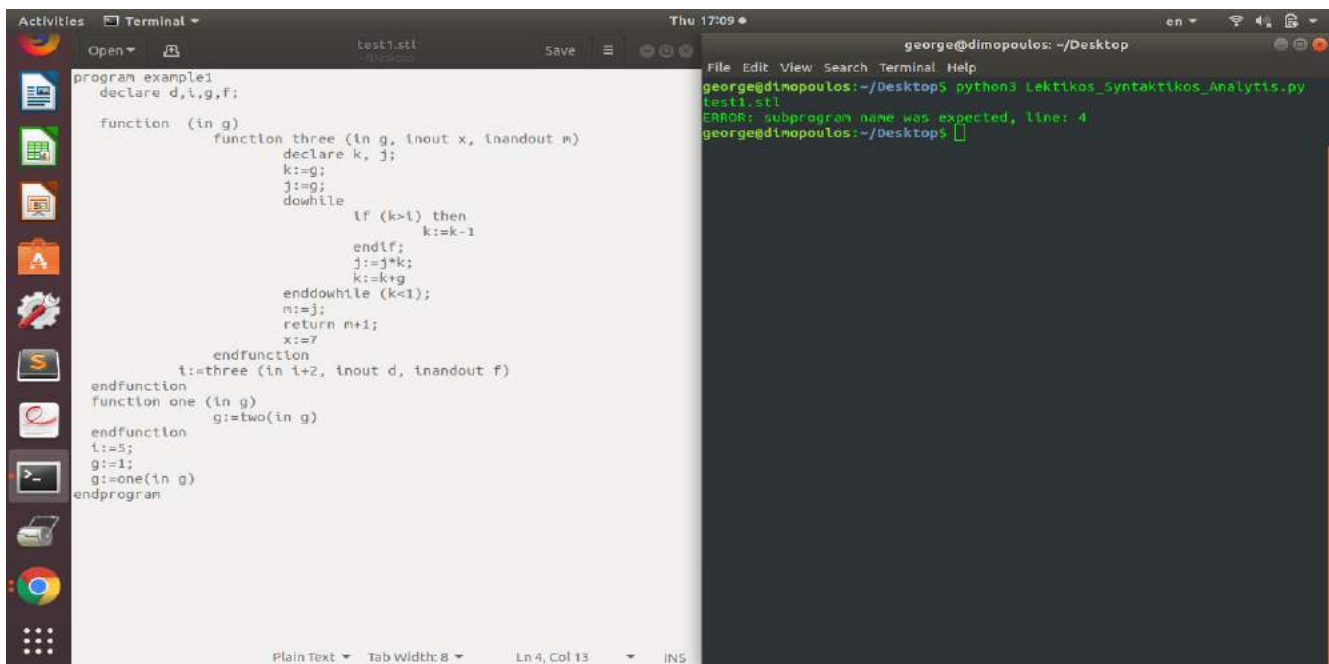
- ✓ **expression**: Αντιστοιχεί στα στοιχεία της πρόσθεσης. Όταν μπαίνουμε σε αυτόν τον κανόνα εκτελούμε τους κανόνες **optional\_sign, term**. Τώρα όσο η λεκτική μονάδα έχει το σύμβολο είτε το ' + ' είτε το ' - ' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τους κανόνες **add\_oper, term**.
- ✓ **term**: Αντιστοιχεί στα στοιχεία πολ/σμου. Όταν μπαίνουμε σε αυτόν τον κανόνα εκτελούμε τον κανόνα **factor**. Τώρα όσο η λεκτική μονάδα έχει το σύμβολο είτε το ' \* ' είτε το ' -/' καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τους κανόνες **mul\_oper, factor**.
- ✓ **factor**: Όταν μπαίνουμε σε αυτό τον κανόνα περιμένουμε είτε κάποια σταθερά είτε το σύμβολο ' ( ' είτε κάποιο id. Αν η λεκτική μονάδα έχει κάποια σταθερά τότε απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα είμαστε στην περίπτωση που έχει ' ( ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **expression**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει ' ) ' καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν είμαστε στην τρίτη περίπτωση που η λεκτική μονάδα έχει κάποιο id τότε απλά καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **idtail**.
- ✓ **idtail**: Αν η λεκτική μονάδα έχει το σύμβολο ' ( ' απλά εκτελούμε τον κανόνα **actualpars**.



- ✓ **relation\_oper:** Αντιστοιχεί στους τελεστές συνθήκης. Αν η λεκτική μονάδα έχει το σύμβολο ' = ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' <= ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' >= ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' > ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' < ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' <> ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα.
- ✓ **add\_oper:** Αντιστοιχεί στην πρόσθεση. Αν η λεκτική μονάδα έχει το σύμβολο ' + ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' - ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα.
- ✓ **mul\_oper:** Αντιστοιχεί στον πολ/σμο. Αν η λεκτική μονάδα έχει το σύμβολο ' \* ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα. Αν η λεκτική μονάδα έχει το σύμβολο ' / ' απλά καταναλώνουμε την επόμενη λεκτική μονάδα.
- ✓ **optional\_sign:** Αντιστοιχεί στους τελεστές πρόσθεσης και αφαίρεσης. Αν η λεκτική μονάδα έχει είτε το σύμβολο ' + ' είτε το σύμβολο ' - ' τότε απλά εκτελούμε τον κανόνα **add\_oper**.
- ✓ **forcase\_stat:** Αντιστοιχεί στην δομή επανάληψης forcase. Αν η λεκτική μονάδα έχει την λέξη when τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει το σύμβολο ' ( ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **condition**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν τώρα η λεκτική μονάδα έχει το σύμβολο ' ) ' τότε καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Τώρα αν η λεκτική μονάδα έχει το σύμβολο ' : ' εκτελούμε τον κανόνα **statements**. Αν στην λεκτική μας μονάδα έχουμε την λέξη default τότε καταναλώνουμε την επόμενη λεκτική μονάδα και εκτελούμε τον κανόνα **statements**, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Αν έχουμε στην λεκτική μας μονάδα την λέξη enddefault

καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα. Τέλος αν έχουμε στην λεκτική μας μονάδα την λέξη endforcase καταναλώνουμε την επόμενη λεκτική μονάδα, διαφορετικά εκτυπώνουμε σχετικό μήνυμα.

Στις παρακάτω εικόνες παρουσιάζονται μερικά παραδείγματα από την λειτουργία του συντακτικού αναλυτή.



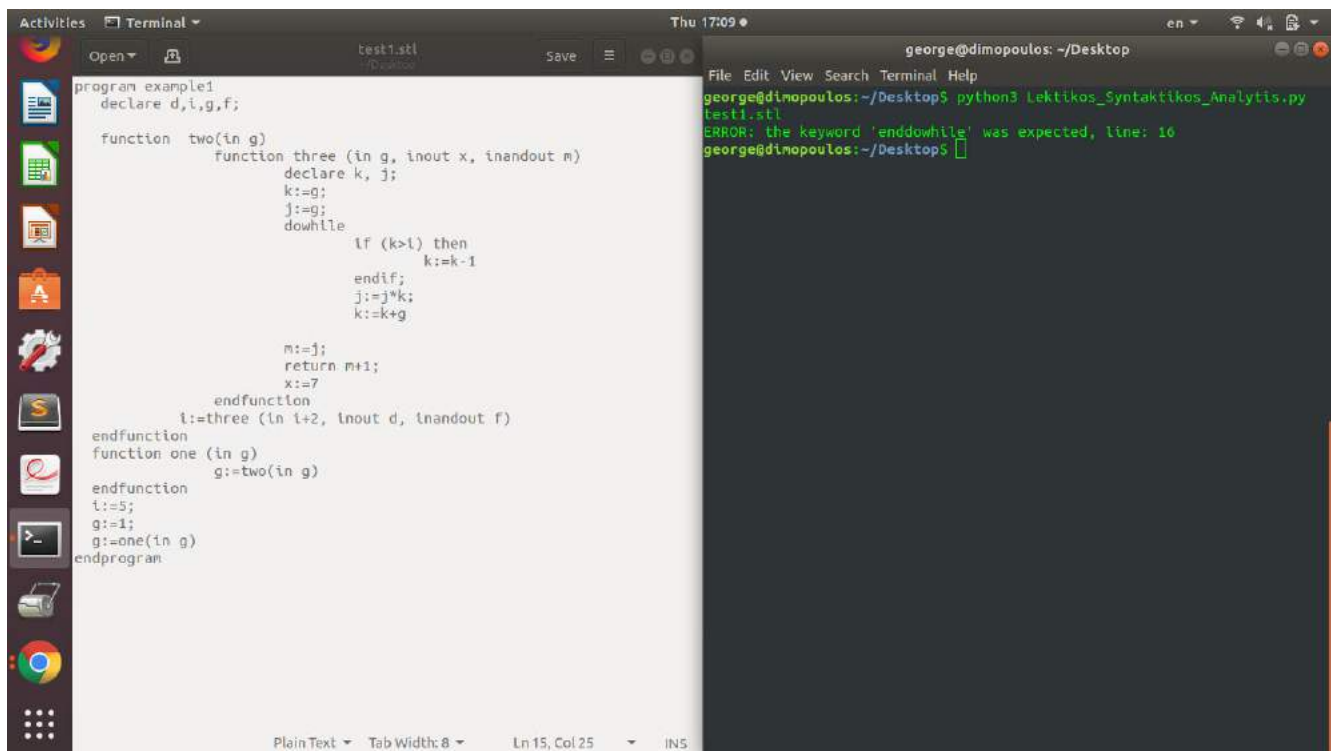
```
program example1
  declare d,i,g,f;

  function (in g)
    function three (in g, inout x, inoutout m)
      declare k, j;
      k:=g;
      j:=g;
      dowhile
        if (k>1) then
          k:=k-1
        endif;
        j:=j*k;
        k:=k+g;
      enddowhile (k<1);
      m:=j;
      return m+1;
      x:=7;
    endfunction
    i:=three (in i+2, inout d, inoutout f)
  endfunction
  function one (in g)
    g:=two(in g)
  endfunction
  i:=5;
  g:=1;
  g:=one(in g)
endprogram
```

```
george@dimopoulos: ~/Desktop
george@dimopoulos:~/Desktop$ python3 Lektikos_Syntaktikos_Analytis.py
test1.stl
ERROR: subprogram name was expected, line: 4
george@dimopoulos:~/Desktop$
```

Στην παραπάνω εικόνα βλέπουμε το μήνυμα λάθους που εμφανίζει ο συντακτικός αναλυτής όταν δεν βρει όνομα συνάρτησης.





The screenshot shows a code editor window with a file named `test1.stl` open. The code is as follows:

```
program example1
  declare d,i,g,f;

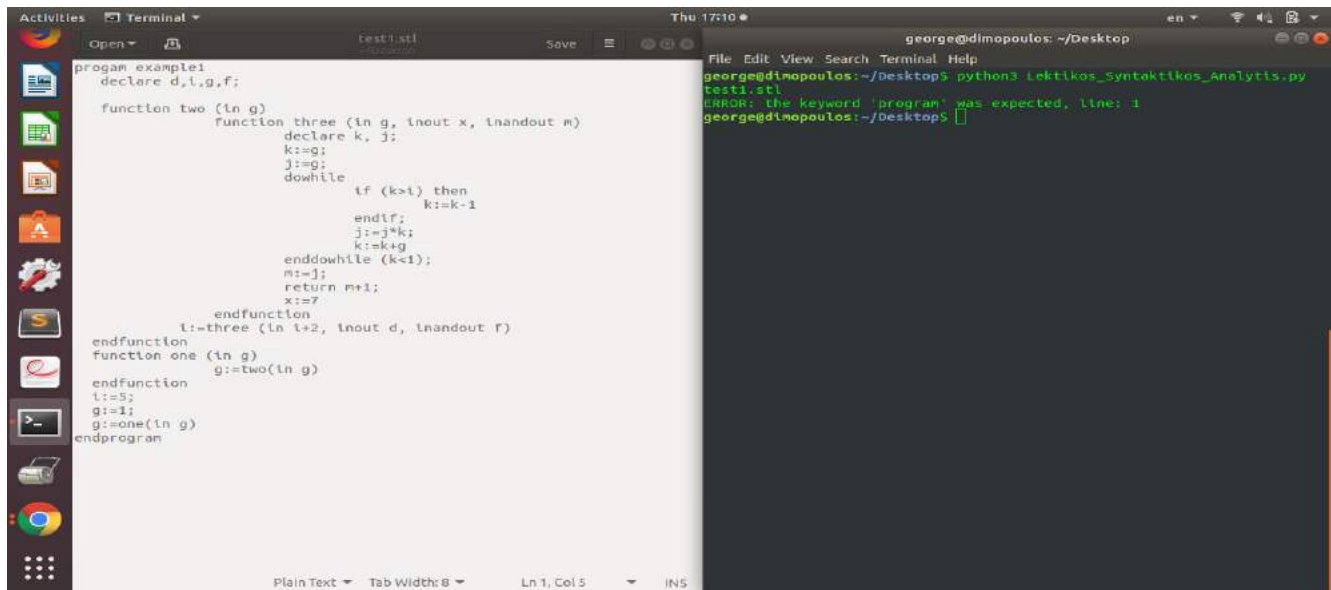
  function two(in g)
    function three (in g, inout x, inandout m)
      declare k, j;
      k:=g;
      j:=g;
      dowhile
        if (k>1) then
          k:=k-1
        endif;
        j:=j*k;
        k:=k+g;

        m:=j;
        return m+1;
        x:=7;
      endfunction
    i:=three (in i+2, inout d, inandout f)
  endfunction
  function one (in g)
    g:=two(in g)
  endfunction
  i:=5;
  g:=1;
  g:=one(in g)
endprogram
```

The terminal window on the right shows the command `python3 Lektikos_Syntaktikos_Analytis.py test1.stl` being executed, which results in the following error message:

```
ERROR: the keyword 'enddowhile' was expected, line: 16
george@dimopoulos: ~/Desktop$
```

Στην παραπάνω εικόνα βλέπουμε το μήνυμα λάθους που εμφανίζει ο συντακτικός αναλυτής όταν δεν κλείσει σωστά δομή επανάληψης `dowhile`.



The screenshot shows the same code editor window with the `test1.stl` file. The code is identical to the previous one, but the terminal window now shows a different error message:

```
ERROR: the keyword 'program' was expected, line: 1
george@dimopoulos: ~/Desktop$
```

Στην παραπάνω εικόνα βλέπουμε το μήνυμα λάθους που εμφανίζει ο συντακτικός αναλυτής όταν δεν δώσουμε την λέξη `program` καθόλου ή λάθος στην ορθογραφία.

## 2. Παραγωγή Ενδιάμεσου Κώδικα

Κατά την διαδικασία μετάφρασης ενός πηγαίου κώδικα σε τελικό κώδικα ένας μεταγλωττιστής μπορεί να κατασκευάσει μια ή περισσότερες ενδιάμεσες αναπαραστάσεις. Στη δική μας περίπτωση, την ενδιάμεση αναπαράσταση αποτελεί ο ενδιάμεσος κώδικας. Πρακτικά ο **ενδιάμεσος κώδικας** είναι ένα σύνολο από τετράδες, οι οποίες αποτελούνται από έναν τελεστή και τρία τελούμενα ( $op, x, y, z$ ), όπου ο τελεστής  $op$  εφαρμόζεται στα τελούμενα  $x, y$  και το αποτέλεσμα αποθηκεύεται στο τελούμενο  $z$ . Το πεδίο  $op$  μπορεί να είναι ένα εκ των:  $+, -, *, /$ . Αντίστοιχα τα τελούμενα  $x, y$  μπορεί να περιέχουν ονόματα μεταβλητών ή αριθμητικές σταθερές. Τέλος το πεδίο  $z$  μπορεί να είναι όνομα μεταβλητής. Οι τετράδες είναι αριθμημένες. Κάθε τετράδα έχει μπροστά της έναν μοναδικό αριθμό που τη χαρακτηρίζει. Μόλις τελειώσει η εκτέλεση μιας τετράδας εκτελείται η τετράδα που έχει τον αμέσως μεγαλύτερο αριθμό, εκτός εάν η τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό.

• Παράδειγμα μιας τετράδας:  $*$ ,  $a$ ,  $b$ ,  $c$  αντιστοιχεί στην πράξη  $c = a * b$ .

Για την δημιουργία του ενδιάμεσου κώδικα χρησιμοποιήσαμε επίσης τις παρακάτω βοηθητικές συναρτήσεις:

- ❖ **nextquad()**: Η οποία επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.
- ❖ **genquad( $op, x, y, z$ )**: Η οποία δημιουργεί την επόμενη τετράδα ( $op, x, y, z$ ).
- ❖ **newtemp()**: Η οποία δημιουργεί και επιστρέφει μια νέα προσωρινή μεταβλητή, που είναι της μορφής  $T_1, T_2, T_3, \dots$

- 
- ❖ **emptylist()**: Η οποία δημιουργεί μια κενή λίστα ετικετών τετράδων.
  - ❖ **makelist(x)**: Η οποία δημιουργεί μια λίστα ετικετών τετράδων που περιέχει μόνο το x.
  - ❖ **merge(list1,list2)**: Η οποία δημιουργεί μια λίστα ετικετών τετράδων από την συνένωση των λιστών list1,list2.
  - ❖ **backpatch(list,z)**: Σε αυτή η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η συνάρτηση αυτή επισκέπτεται μια από τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

Για την παραγωγή του ενδιάμεσου κώδικα χρησιμοποιήσαμε τις παραπάνω συναρτήσεις τοποθετημένες στις συναρτήσεις του συντακτικού αναλυτή. Σύμφωνα με την δομή που θέλουμε να μετατρέψουμε σε ενδιάμεσο κώδικα, εφαρμόσαμε τους εξής κανόνες σε συνδυασμό με τις βοηθητικές συναρτήσεις. Παρακάτω ακολουθούν αναλυτικά όλες οι περιπτώσεις.

### Αριθμητικές παραστάσεις:

Για τον υπολογισμό των αριθμητικών παραστάσεων τροποποιήθηκαν οι συναρτήσεις **expression()**, **term()**, **factor()**. Συγκεκριμένα:

Για τις αριθμητικές παραστάσεις προσθέσαμε δύο μεταβλητές στην **expression** τις *TPlace* και *T1Place* καθώς και την *EPlace*. Κατά την εκτέλεση της συνάρτησης **expression** η μεταβλητή *TPlace* παίρνει την τιμή που επέστρεψε η συνάρτηση *term*. Στην συνέχεια καλούνται οι συναρτήσεις **add\_oper** και **term** και αρχικοποιούνται τις μεταβλητές *op* και *T1Place* αντίστοιχα. Προκειμένου να αποθηκεύσουμε το αποτέλεσμα, δημιουργούμαι μέσω της συνάρτησης **newTemp** την προσωρινή μεταβλητή *temp* και καλούμε την συνάρτηση **genQuad(op,TPlace,T1Place,temp)**, όπου η μεταβλητή *op* είναι ο τελεστής και *temp* η προσωρινή μεταβλητή. Αφού

---

δημιουργήσουμε την τετράδα με την χρήση της **genQuad** εκχωρούμε στην μεταβλητή *TPlace* την προσωρινή μεταβλητή *temp*. Τέλος η μεταβλητή *EPlace* παίρνει την τιμή της *TPlace*. Αυτό θα γίνεται όσο ο λεκτικός αναλυτής δίνει “+” ή “-”.

Με το ίδιο σκεπτικό τροποποιήσαμε τις συναρτήσεις **term** και **factor**.

### Λογικές Παραστάσεις:

Για τον υπολογισμό των λογικών παραστάσεων τροποποιήθηκαν οι συναρτήσεις **condition()**, **boolterm()**, **boolfactor()**. Συγκεκριμένα:

Στην συνάρτηση **condition** δημιουργήθηκαν οι λίστες *trueList* και *falseList*. Αφού καλέσουμε την **boolterm**, η λίστα που επιστρέφει αποθηκεύεται στην μεταβλητή *bool list* και διαμορφώνει κατάλληλα τις λίστες *trueList* και *falseList*. Ενόσω είμαστε στην **while** με “or” καλείται η **backpatch(falseList,nextQuad())**. Αυτό γίνεται έτσι ώστε να συμπληρωθούν οι τετράδες που δημιουργήθηκαν από την **boolterm,boolfactor** με τον αριθμό της ετικέτας της επόμενης τετράδας, θεωρώντας πως οι τετράδες αυτές έχουν τις ψευδείς αποτιμήσεις των συνθηκών. Στην συνέχεια καλούμε την **boolterm** και αρχικοποιείται εκ νέου η *bool list*. Τότε καλούμε την **merge(trueList,bool\_list[0])** προκειμένου να δημιουργηθεί μια λίστα με τις ετικέτες των τετράδων στις οποίες θα μεταβεί η συνθήκη για αληθή αποτίμηση. Με την ίδια λογική και το ίδιο σκεπτικό δημιουργούμε και την **boolterm** και **boolfactor**. Στην **boolfactor** επίσης κάνουμε χρήση της **makelist** ώστε να θυμόμαστε ποια ετικέτα πρέπει να συμπληρώσει σε περίπτωση αληθούς αποτίμησης. Τέλος παράγουμε και την τετράδα **jump** με κενά με την **genquad(“jump”, “\_”, “\_”, “\_”)**.

### Κλήση υποπρογραμμάτων:

Για να μετατρέψουμε σε ενδιάμεσο κώδικα την κλήση υποπρογραμμάτων καλούμε την συνάρτηση **genQuad** στην συνάρτηση **actualpars**.

### Δομή while:

---

Στην συνάρτηση **while\_stat** εκχωρούμε στην μεταβλητή **quad** την τιμή που επιστρέφει η συνάρτηση **nextQuad**, προκειμένου να μεταβούμε στην σωστή τετράδα στην περίπτωση που αποτιμηθεί ψευδώς η συνθήκη. Έπειτα καλεί την **condition** και η **backpatch** ώστε να συμπληρωθούν οι τετράδες με την ετικέτα που θα μεταβούν για αληθή αποτίμηση. Τέλος καλούμε την **genQuad("jump","\_","\_",quad)** για να επανέλθουμε στη συνθήκη.

#### Δομή if\_stat:

Στην συνάρτηση **if\_stat** καλούμε την **condition** και με την λίστα που επιστρέφει αρχικοποιούμε την μεταβλητή **bool\_list**. Αυτή χρησιμοποιείται στην κλήση της **backpatch(bool\_list[0],nextquad())** με αποτέλεσμα να συμπληρωθεί η τετράδα στην οποία θα μεταβούμε εάν η συνθήκη είναι αληθής. Στη συνέχεια δημιουργούμε την **List** για να ξέρουμε που θα μεταβούμε εάν εκτελεστεί το **if** ή το **else**.

#### Δομή forcase:

Για την υλοποίηση του ενδιάμεσου κώδικα αρχικοποιούμε την μεταβλητή **fQuad**, μέσω της συνάρτησης **nextQuad**, ώστε να γνωρίζουμε σε ποιο σημείο να επιστρέψουμε μόλις τελειώσει η **forcase**. Έπειτα αρχικοποιούμε τη μεταβλητή **bool\_list**, μέσω της συνάρτησης **condition**, και καλούμε την συνάρτηση **backpatch(bool\_list[0],nextquad())**. Στην συνέχεια καλούμε την συνάρτηση **genQuad("jump","\_","\_",fQuad)** για να επιστρέψουμε στην αρχή της **forcase**. Τέλος καλείται η **backpatch(bool\_list[1],nextquad())** για να βγούμε από την εντολή.

Παρακάτω απεικονίζουμε τον παραγόμενο ενδιάμεσο κώδικα ενός προγράμματος Starlet.

```
program example1
  declare d,i,g,f;

  function two (in g)
    function three (in g, inout x, inandout m)
      declare k, j;
      k:=g;
      j:=g;
      dowhile
        if (k>l) then
          k:=k-1
        endif;
        j:=j*k;
        k:=k+g;
      enddowhile (k<1);
      m:=j;
      return m+1;
      x:=7;
    endfunction
    i:=three (ln i+2, inout d, inandout f)
  endfunction
  function one (in g)
    g:=two(in g)
  endfunction
  i:=5;
  g:=1;
  g:=one(in g)
endprogram
```

```
1: begin_block three
2: := g k
3: := g j
4: > k i 6
5: jump
6: k 1 T_1
7: := T_1 k
8: jump
9: * j k T_2
10: := T_2 j
11: + k g T_3
12: := T_3 k
13: < k 1 4
14: jump
15: := j m 15
16: + m 1 T_4
17: retv T_4
18: := 7 x
19: end_block three
20: begin_block two
21: + i 2 T_5
22: par T_5 CV
23: par d REF
24: par f REF
25: par T_6 RET
26: call three
27: := three i
28: end_block two
29: begin_block one
30: par g CV
31: par T_7 RET
32: call two
33: := two g
34: end_block one
35: begin_block example1
36: := 5 i
37: := 1 g
38: par g CV
39: par T_8 RET
40: call one
41: := one g
42: halt
43: end_block example1
```

---

Παρακάτω θα αναλύουμε κάθε γραμμή του ενδιάμεσου κώδικα με βάση τους παραπάνω κανόνες. Συγκεκριμένα:

1. Δημιουργείται η τετράδα που δηλώνει την αρχή της συνάρτησης `three`.
2. Γίνεται η ανάθεση τιμής του `g` στο `k`.
3. Γίνεται η ανάθεση τιμής του `g` στο `j`.
4. Υλοποιείται η τετράδα για το `condition k>i`. Σε περίπτωση αληθούς αποτίμησης μεταβαίνουμε στην ετικέτα 6. Αντίθετα αν η συνθήκη αποτιμηθεί ψευδής συνεχίζει στην επόμενη ετικέτα.
5. Δημιουργεί την ετικέτα στην οποία θα μεταβεί αν η προηγούμενη συνθήκη αποτιμηθεί ψευδής. Συγκεκριμένα η τετράδα αυτή κάνει `jump` στην ετικέτα 9.
6. Υλοποιείται η αριθμητική παράσταση  $k:=k-1$ . Συγκεκριμένα στην γραμμή 6 γίνεται η αφαίρεση  $k-1$  και την εκχωρεί στην προσωρινή μεταβλητή `T_1`. Στη συνέχεια στην  $\rightarrow$
7. Η αποτίμηση της τιμής `T_1` στην μεταβλητή `k`.
8. Βγαίνει από την συνθήκη `if` και κάνει `jump` στην αμέσως επόμενη ετικέτα, δηλαδή στην 9.
9. Υλοποιείται η αριθμητική παράσταση  $j:=j*k$ . Συγκεκριμένα στην γραμμή 9 γίνεται ο πολλαπλασιασμός  $j*k$  και το αποτέλεσμα εκχωρείται στην προσωρινή μεταβλητή `T_2`. Στη συνέχεια στη  $\rightarrow$
10. Εκχωρείται η τιμή της `T_2` στην μεταβλητή `j`.
11. Υλοποιείται η αριθμητική παράσταση του  $k:=k+j$ . Συγκεκριμένα στην γραμμή 11 υλοποιείται η πρόσθεση  $k+g$  και το αποτέλεσμα εκχωρείται στην προσωρινή μεταβλητή `T_3`. Στην συνέχεια στη  $\rightarrow$
12. Εκχωρείται η τιμή της `T_3` στην μεταβλητή `k`.
13. Υλοποιείται η τετράδα που περιγράφει τη συνθήκη εξόδου από την δομή επανάληψης `dowhile`. Συγκεκριμένα αν η συνθήκη  $k<1$  είναι αληθής θα μεταβούμε στην ετικέτα νούμερο 4. Διαφορετικά ακολουθούμε την αμέσως επόμενη ετικέτα.
14. Υλοποιείται η τετράδα για την περίπτωση της ψευδούς αποτίμησης της παραπάνω συνθήκης. Ειδικότερα αν η παραπάνω συνθήκη αποτιμηθεί ψευδής μεταβαίνουμε στην επόμενη τετράδα.



- 
15. Δημιουργείται η τετράδα για την εκχώρηση τιμής της μεταβλητής  $j$  στην μεταβλητή  $m$ .
  16. Υλοποιείται η αριθμητική πράξη  $m+1$ , το αποτέλεσμα της οποίας τοποθετείται στην προσωρινή μεταβλητή  $T\_4$ , προκειμένου να δημιουργηθεί η τιμή που θα επιστραφεί στην επόμενη ετικέτα.
  17. Επιστρέφεται η τιμή  $T\_4$ .
  18. Γίνεται η αρχικοποίηση της μεταβλητής  $x$  με την τιμή 7.
  19. Δημιουργείται η τετράδα που δηλώνει το τέλος της συνάρτησης `three`.
  20. Δημιουργείται η τετράδα που δηλώνει την αρχή της συνάρτησης `two`.
  21. Δημιουργείται η τετράδα που περιγράφει την αριθμητική πράξη  $i+2$  και την εκχωρεί στην προσωρινή μεταβλητή  $T\_5$ , η οποία θα περαστεί σαν παράμετρος στην επόμενη ετικέτα.
  22. Γίνεται πέρασμα της παραμέτρου  $T\_5$  με τιμή.
  23. Γίνεται πέρασμα της παραμέτρου  $d$  με αναφορά.
  24. Γίνεται πέρασμα της παραμέτρου  $f$  με αναφορά.
  25. Γίνεται πέρασμα της παραμέτρου  $T\_6$  στην οποία θα εκχωρηθεί η τιμή που θα επιστρέψει η συνάρτηση `three`.
  26. Καλείται η συνάρτηση `three`.
  27. Εκχωρείται στην μεταβλητή  $i$  η τιμή που επέστρεψε η συνάρτηση `three`.
  28. Δημιουργείται η τετράδα που δηλώνει το τέλος της συνάρτησης `two`.
  29. Δημιουργείται η τετράδα που δηλώνει την αρχή της συνάρτησης `one`.
  30. Γίνεται πέρασμα της μεταβλητής  $g$  με τιμή.
  31. Γίνεται πέρασμα της παραμέτρου  $T\_7$  στην οποία θα εκχωρηθεί η τιμή που θα επιστρέψει η συνάρτηση `two`.
  32. Καλείται η συνάρτηση `two`.
  33. Εκχωρείται στην μεταβλητή  $g$  η τιμή που επέστρεψε η συνάρτηση `two`.
  34. Δημιουργείται η τετράδα που δηλώνει το τέλος της συνάρτησης `one`.
  35. Δημιουργείται η τετράδα που δηλώνει την αρχή του προγράμματος `example1`.
  36. Γίνεται η αρχικοποίηση της μεταβλητής  $i$  με την τιμή 5.
  37. Γίνεται η αρχικοποίηση της μεταβλητής  $g$  με την τιμή 1.
  38. Γίνεται το πέρασμα της μεταβλητής  $g$  με τιμή.
  39. Γίνεται το πέρασμα της παραμέτρου  $T\_8$  στην οποία θα εκχωρηθεί η τιμή που θα επιστρέψει η συνάρτηση `one`.



- 
- 40. Καλείται η συνάρτηση 1.
  - 41. Εκχωρείται στην μεταβλητή g η τιμή που επέστρεψε η συνάρτηση one.
  - 42. Δημιουργείται τετράδα που δηλώνει τον τερματισμό του προγράμματος.
  - 43. Δημιουργείται η τετράδα που δηλώνει το τέλος του προγράμματος example1.

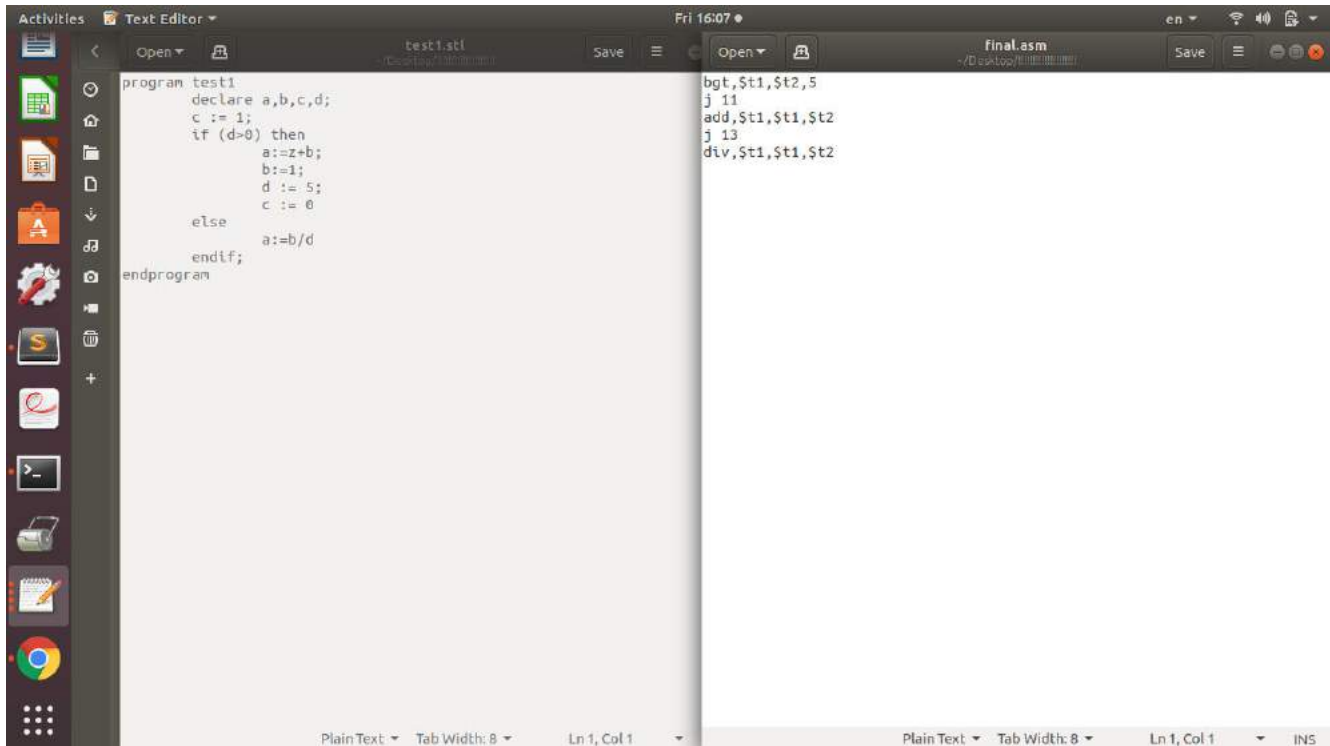
## 3. Παραγωγή Τελικού Κώδικα

Η παραγωγή τελικού κώδικα αποτελεί την τελευταία φάση της μετάφρασης. Για κάθε εντολή του ενδιάμεσου κώδικα παράγονται αντίστοιχες εντολές τελικού κώδικα. Συγκεκριμένα για κάθε τετράδα του ενδιάμεσου κώδικα γράψαμε την αντίστοιχη σε γλώσσα μηχανής (assembly). Αναλυτικότερα σε κάθε τετράδα, αναλόγως με τον τελεστή που διάβαζε, τον μετέτρεπε στον αντίστοιχο τελεστή της γλώσσας μηχανής όπως φαίνεται παρακάτω:

- < → blt
- <= → ble
- <> → bne
- > → bgt
- >= → bge
- = → beq
- := → loadvr( , 1)  
          storerv(1 , )
- + → add
- - → sub
- \* → mul
- / → div
- Jump → j
- Out → li \$v0  
         li \$a0

## starlet

Παρακάτω απεικονίζεται ο παραγόμενος τελικός κώδικας ενός προγράμματος starlet.



```
program test1
  declare a,b,c,d;
  c := 1;
  if (d>0) then
    a:=z+b;
    b:=1;
    d := 5;
    c := 0;
  else
    a:=b/d;
  endif;
endprogram
```

```
bgt,$t1,$t2,5
j 11
add,$t1,$t1,$t2
j 13
div,$t1,$t1,$t2
```

**ΣΗΜΕΙΩΣΗ:** Στην παραπάνω φάση παραγωγής τελικού κώδικα δεν έχουν υλοποιηθεί όλες οι συναρτήσεις που ζητήθηκαν. Συγκεκριμένα δεν υλοποιήθηκαν οι συναρτήσεις loadvr, storerv, gnlncode και η φάση της παραγωγής του πίνακα συμβόλων. Για το λόγο αυτό ο παραγόμενος τελικός κώδικας είναι ελλιπής.