

# ΠΑΡΟΥΣΙΑΣΗ ΑΣΚΗΣΗΣ 1

---

Το αρχείο `1.cpp` αφορά την προσέγγιση της προγραμματιστικής λύσης πάνω στο Θέμα 1, και αρχείο `1.exe` είναι η εκτελέσιμη λύση, στην οποία βρίσκεται η υλοποίηση της προσέγγισης. Τόσο το προγραμματιστικό, όσο και το εκτελέσιμο κομμάτι της λύσης, υλοποιήθηκε με τα εργαλεία της γλώσσας C++. Η εκτέλεση του προγράμματος πραγματοποιήθηκε με το Command Prompt & το Windows PowerShell.

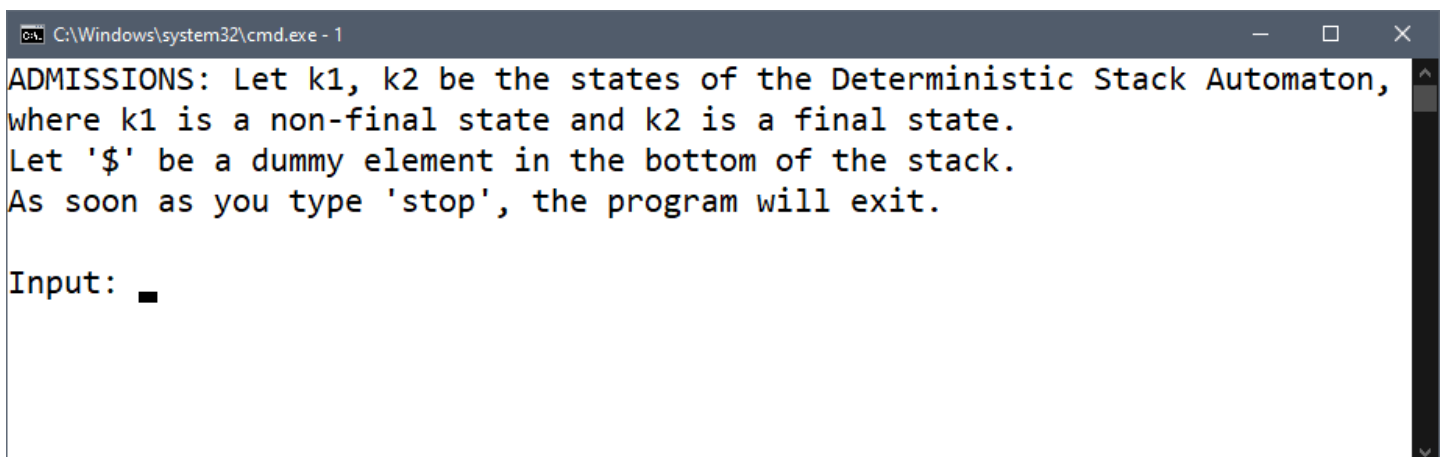
## 1. ΕΚΤΕΛΕΣΙΜΟ ΚΟΜΜΑΤΙ:

### ΕΙΣΑΓΩΓΗ

Με τη χρήση του μεταγλωττιστή GNU και την εντολή `g++ 1.cpp -o 1.exe`, στο λειτουργικό σύστημα των Windows, παρήχθη η εκτελέσιμη λύση.

Το εκτελέσιμο αρχείο μπορεί να ανοιχθεί, είτε από ένα τερματικό, είτε πατώντας διπλό κλικ πάνω στο αρχείο, καθώς έχει προγραμματιστεί έτσι, ώστε να μην κλείνει το αρχείο χωρίς τη βούληση του χρήστη. Η εντολή για την μεταγλώττιση και την παραγωγή του αρχείου `1.exe`, δεν θα έπρεπε να επιφέρει παρατηρήσεις ή σφάλματα.

Όταν γίνει η εκτέλεση του προγράμματος, θα έπρεπε να εμφανίζεται το εξής μενού.



```
C:\Windows\system32\cmd.exe - 1
ADMISSIONS: Let k1, k2 be the states of the Deterministic Stack Automaton,
where k1 is a non-final state and k2 is a final state.
Let '$' be a dummy element in the bottom of the stack.
As soon as you type 'stop', the program will exit.

Input: ■
```

## ΠΑΡΑΔΟΧΕΣ

Όπως αναγράφεται και στην αρχή της εκτέλεσης του προγράμματος, υπάρχουν κάποιες παραδοχές, που πρέπει οπωσδήποτε να αναφερθούν, ώστε να γίνει κατανοητή η λύση. **Οι παραδοχές έχουν ως εξής:**

- Το πρόγραμμα θα δέχεται είσοδο μόνο από το τερματικό και όχι μέσω εξωτερικού αρχείου.
- Αποδεκτοί χαρακτήρες εισόδου είναι μόνο οι λατινικοί.
- Κάθε είσοδος μετατρέπεται, αυτόματα, από κεφαλαία γράμματα σε πεζά.
- Θεωρούμε  $k_1$  μια μη τερματική κατάσταση του Αυτομάτου Στοίβας.
- Θεωρούμε  $k_2$  μια τερματική κατάσταση του Αυτομάτου Στοίβας.
- Κάθε φορά, που αναγνωρίζεται μία είσοδος, βάζουμε στην Στοίβα ένα βοηθητικό αντικείμενο, που το ονομάζουμε '\$'.
- Το πρόγραμμα θα κλείνει, αν και μόνο αν ο χρήστης εισάγει την έκφραση 'stop'. (Βάσει της τρίτης παραδοχής, **εκφράσεις όπως π.χ. 'StOp' ή 'STOP' είναι αποδεκτές**)

## ΑΝΑΛΥΣΗ ΕΙΣΟΔΟΥ

Μία έκφραση, αποτελούμενη μόνο από χαρακτήρες 'x' και 'y', είναι αποδεκτή και το πρόγραμμα θα προχωρήσει στη διαδικασία της ανάλυσης. Όπως αναφέρθηκε και πιο πριν, όποια κι αν είναι η είσοδος του χρήστη, αυτή θα μετατραπεί σε πεζά γράμματα. Οπότε, εκφράσεις όπως "XXXYYY" ή "xxxYYY", δεν θα αποτελέσουν πρόβλημα.

Το πρόγραμμα θα αναδείξει αναλυτικά τις ενέργειες στις οποίες προβαίνει, για την αποδοχή ή την απόρριψη της έκφρασης. **Κάθε φορά που αναλύεται μία έκφραση το πρόγραμμα προβάλλει:**

- 1) Τον αριθμό επανάληψης.
- 2) Την κατάσταση του Αυτομάτου Στοίβας.
- 3) Το περιεχόμενο της Στοίβας.
- 4) Το υπόλοιπο εισόδου.

Για λόγους υπόδειξης, **δίδονται τα παρακάτω τέσσερα παραδείγματα** (δύο ορθών και δύο μη ορθών) εισόδων.

## ΟΡΘΕΣ ΕΙΣΟΔΟΙ:

Υπόδειξη 1: Πρώτο παράδειγμα ορθής εισόδου.

Input: xxxxyyyy

1. Current State: k1, Stack Items: x\$, Remaining Input: xxxyyyy
2. Current State: k1, Stack Items: xx\$, Remaining Input: xyyyy
3. Current State: k1, Stack Items: xxx\$, Remaining Input: yyyyy
4. Current State: k1, Stack Items: xxxx\$, Remaining Input: yyyy
5. Current State: k1, Stack Items: xxx\$, Remaining Input: yyy
6. Current State: k1, Stack Items: xx\$, Remaining Input: yy
7. Current State: k1, Stack Items: x\$, Remaining Input: y
8. Current State: k2, Stack Items: \$, Remaining Input: (empty)

String 'xxxxyyyy' is accepted.

Υπόδειξη 2: Δεύτερο παράδειγμα ορθής εισόδου.

Input: xxxxyyxyyyy

1. Current State: k1, Stack Items: x\$, Remaining Input: xxxyyxyyyy
2. Current State: k1, Stack Items: xx\$, Remaining Input: xxyyxyyyy
3. Current State: k1, Stack Items: xxx\$, Remaining Input: xyxyxyyyy
4. Current State: k1, Stack Items: xxxx\$, Remaining Input: yyxyxyyyy
5. Current State: k1, Stack Items: xxx\$, Remaining Input: yxyyyy
6. Current State: k1, Stack Items: xx\$, Remaining Input: xxyyyy
7. Current State: k1, Stack Items: xxx\$, Remaining Input: yyyyy
8. Current State: k1, Stack Items: xxxx\$, Remaining Input: yyyy
9. Current State: k1, Stack Items: xxx\$, Remaining Input: yyy
10. Current State: k1, Stack Items: xx\$, Remaining Input: yy
11. Current State: k1, Stack Items: x\$, Remaining Input: y
12. Current State: k2, Stack Items: \$, Remaining Input: (empty)

String 'xxxxyyxyyyy' is accepted.

**Όλες οι παραπάνω εισοδοί, βρέθηκαν στην (τερματική) κατάσταση k<sub>2</sub>, πάνω στην λήξη της ανάλυσης της έκφρασης.** Παρατηρείται επίσης, ότι σε κάθε ορθή έκφραση, η στοίβα έχει ως μοναδικό στοιχείο το '\$' στο τελευταίο βήμα (αυτό θα εξηγηθεί παρακάτω).

Ωστόσο, αξίζει να δούμε τι γίνεται, όταν βάλουμε, για είσοδο, μία μη ορθή έκφραση.

## ΜΗ ΟΡΘΕΣ ΕΙΣΟΔΟΙ

Υπόδειξη 3: Πρώτο παράδειγμα μη ορθής εισόδου.

Input: xxxxyyy

1. Current State:  $k_1$ , Stack Items:  $x\$$ , Remaining Input: xxxxyyy
2. Current State:  $k_1$ , Stack Items:  $xx\$$ , Remaining Input: xxxyyy
3. Current State:  $k_1$ , Stack Items:  $xxx\$$ , Remaining Input: xxyyy
4. Current State:  $k_1$ , Stack Items:  $xxxx\$$ , Remaining Input: xyyy
5. Current State:  $k_1$ , Stack Items:  $xxxxx\$$ , Remaining Input: yyy
6. Current State:  $k_1$ , Stack Items:  $xxxx\$$ , Remaining Input: yy
7. Current State:  $k_1$ , Stack Items:  $xxx\$$ , Remaining Input: y
8. Current State:  $k_1$ , Stack Items:  $xx\$$ , Remaining Input: (empty)

String 'xxxxyyy' is NOT accepted.

Error message: Stack still has items to iterate.

Υπόδειξη 4: Δεύτερο παράδειγμα μη ορθής εισόδου.

Input: xyxyyyxx

1. Current State:  $k_1$ , Stack Items:  $x\$$ , Remaining Input: xyxyyyxx
  2. Current State:  $k_1$ , Stack Items:  $xx\$$ , Remaining Input: yxyyyxx
  3. Current State:  $k_1$ , Stack Items:  $x\$$ , Remaining Input: yyyxx
  4. Current State:  $k_1$ , Stack Items:  $\$$ , Remaining Input: yyxx
- Tried to pop the element '\$'. Cannot continue.

String 'xyxyyyxx' is NOT accepted.

Error message: Tried to pop the element '\$'.

Καμμία από τις παραπάνω εισόδους δεν βρέθηκαν στο τέλος της ανάλυσης σε κατάσταση  $k_2$ , πράγμα που σημαίνει ότι αυτές οι είσοδοι δεν μπορούν να είναι αποδεκτές. Στην  $k_2$ , θα βρισκόμαστε αν και μόνο αν στο τέλος έχουμε στην στοίβα μόνο το στοιχείο '\$', αλλά παράλληλα δεν εκκρεμεί και υπόλοιπο εισόδου. Π.χ., στο δεύτερο παράδειγμα, στο βήμα 4, μολονότι έχουμε μοναδικό στοιχείο το '\$', εκκρεμεί το υπόλοιπο εισόδου, άρα η κατάσταση παραμένει  $k_1$ .

## ΕΣΦΑΛΜΕΝΕΣ ΕΙΣΟΔΟΙ

Κάθε φορά που ο χρήστης εισάγει μία είσοδο, που δεν αποτελείται μονάχα από χαρακτήρες 'x' και 'y', το πρόγραμμα δεν βαίνει στην διαδικασία ανάλυσης. Ωστόσο, δεν τερματίζεται! Ο χρήστης μπορεί να δώσει όσες εισόδους θέλει, καθώς το πρόγραμμα δεν τερματίζεται ποτέ, παρά μόνον κατόπιν εντολής του χρήστη.

Υπόδειξη 5: Παραδείγματα μη αναγνωρίσιμων εκφράσεων.

```
Input: TESTING
Incorrect Input. Only x or y characters are allowed. If you want to exit, type 'stop'.

Input: testing
Incorrect Input. Only x or y characters are allowed. If you want to exit, type 'stop'.

Input: test
Incorrect Input. Only x or y characters are allowed. If you want to exit, type 'stop'.

Input:
```

## ΟΡΘΟΣ ΤΕΡΜΑΤΙΣΜΟΣ

Αν ο χρήστης θέλει να τερματίσει την λειτουργία του προγράμματος, με τον βέλτιστο δυνατό τρόπο, τότε θα πρέπει να γράψει την λέξη "stop". Όταν γίνεται αυτό, το πρόγραμμα τερματίζεται με κωδικό εξόδου 0, πράγμα που εξασφαλίζει τη ασφαλέστερη δυνατή λήξη του προγράμματος. Ο χρήστης είναι πιθανό να δει και ένα ανάλογο μήνυμα, κατόπιν της λήξης.

Υπόδειξη 6: Ορθός τερματισμός του προγράμματος.

```
As soon as you type 'stop', the program will exit.
```

```
Input: stop
Program stopped regularly.
```

## 2. ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΥΛΟΠΟΙΗΣΗ:

### ΛΟΓΙΚΗ

Για να διευκολυνθεί ο χειρισμός του προγράμματος, **θα πρέπει το πρόγραμμα να μην τερματίζεται, όταν τελειώνει η ανάλυση της έκφρασης. Αυτό σημαίνει ότι ο χρήστης μπορεί να επαναλάβει όσες φορές επιθυμεί τη διαδικασία ανάλυσης εισόδου, χωρίς να χρειάζεται να ξανά-τρέξει το πρόγραμμα από την αρχή.** Με ένα σχέδιο, το πρόγραμμα θα μπορεί να ταυτοποιηθεί με το εξής διάγραμμα:



Έναν ακόμη παράγοντα διευκόλυνσης χειρισμού θα αποτελέσει **η μετατροπή της εισόδου του χρήστη σε πεζά γράμματα.** Αυτό δεν θα διευκολύνει **μόνχα** **χρήστη, αλλά και την κωδικοποίηση,** καθώς, θα χρειάζονται σαφέστερα λιγότεροι έλεγχοι, κανονικές εκφράσεις, μηνύματα σφάλματος και λοιποί άλλοι παράγοντες.

Η ανάλυση της έκφρασης θα γίνεται γράμμα-προς-γράμμα, ώστε να έχουμε πλήρη εικόνα της έκφρασης. Κατά τη διαδικασία της ανάλυσης, θα μπορούμε να βάζουμε χαρακτήρες ίδιους με την έκφραση. **Από τη στιγμή που έχουμε μόνο δύο πιθανούς χαρακτήρες ('x' και 'y'), μία «εύκολη» λύση είναι, όποτε πέφτουμε σε**



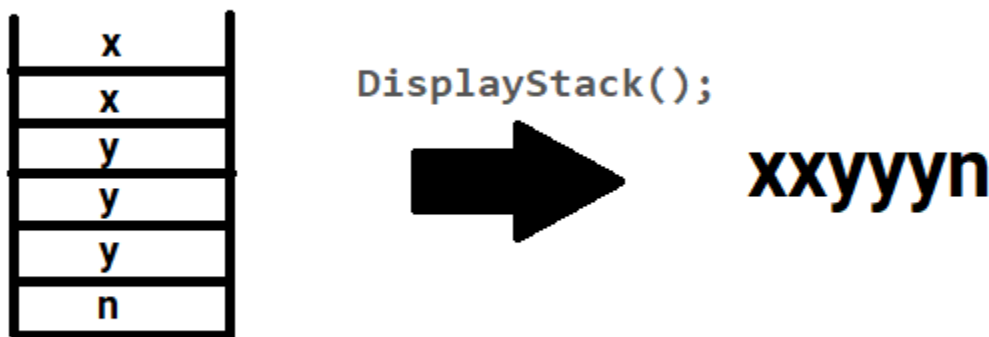
ένα γράμμα 'x', να βάζουμε και στη στοίβα τον χαρακτήρα 'x'. Αντιθέτως, όταν θα πέφτουμε σε ένα γράμμα 'y', θα βγάζουμε ένα στοιχείο από τη στοίβα. Για να επιτυγχάνεται αυτό, θα πρέπει να προσθέσουμε, στο βάθος της στοίβας, κι ένα βοηθητικό στοιχείο, το οποίο θα μας βοηθάει να εγκρίνουμε ή να απορρίπτουμε την έκφραση, ανάλογα με την παρουσία του στη κορυφή της στοίβας(ή την απουσία του) στο τέλος της διαδικασίας.

## ΕΦΑΡΜΟΓΗ

Όλα τα παραπάνω επιτυγχάνονται με βοηθητικές βιβλιοθήκες της γλώσσας C++, ώστε να βελτιστοποιηθεί η κωδικοποίηση. **Συγκεκριμένα, οι κατ' ελάχιστον τέσσερις βιβλιοθήκες που χρειαζόμαστε, είναι:**

1. Για την επικοινωνία υπολογιστή/χρήστη (in out stream ή iostream).
2. Για να την υλοποίηση της στοίβας.
3. Η <string>, που θα βοηθήσει στην επεξεργασία της έκφρασης.
4. Για ελέγχους με κανονικές εκφράσεις.

Ένα πρόβλημα, που δημιουργείται με την βιβλιοθήκη της στοίβας, είναι ότι δεν παρέχεται συνάρτηση προβολής των στοιχείων μιας στοίβας. Οπότε, δημιουργήθηκε μία από εμάς. **Η συνάρτηση παίρνει ένα αντίγραφο της στοίβας (ώστε να μην τροποποιηθεί η πρωτότυπη) και προβάλλει τα στοιχεία ένα-προς-ένα, ξεκινώντας από την κορυφή.**



Ύστερα, αφού το πρόγραμμα έχει δείξει τις παραδοχές και έχει πάρει την είσοδο του χρήστη, κάνει έναν έλεγχο με τα τρία πιθανά σενάρια, που έχουν επεξηγηθεί παραπάνω. **Συγκεκριμένα, ελέγχουμε τα εξής:**

- Η πρώτη περίπτωση είναι μια έκφραση αποτελούμενη από χαρακτήρες ‘x’ και ‘y’. Εδώ έρχεται στο προσκήνιο η βιβλιοθήκη των κανονικών εκφράσεων. **Συγκεκριμένα, με την κανονική έκφραση “[xy]+”, βλέπουμε αν υπάρχουν αποκλειστικά χαρακτήρες είτε ‘x’, είτε ‘y’, τουλάχιστον μία φορά.**
- Ύστερα, ελέγχουμε αν ο χρήστης έχει γράψει ακριβώς την έκφραση “stop”. Σε αυτήν την περίπτωση, **το πρόγραμμα τερματίζεται με ασφαλή τρόπο, με κωδικό εξόδου 0.**
- Αν δεν υποθέσουμε σε κάποιες από τις δύο παραπάνω περιπτώσεις, αναζητάμε πάλι είσοδο από το χρήστη και κάνουμε ξανά τους ελέγχους.

## ΑΝΑΛΥΣΗ ΕΚΦΡΑΣΗΣ

Αν έχουμε φτάσει στο σημείο της ανάλυσης της έκφρασης, σημαίνει ότι θα χρησιμοποιήσουμε τη στοίβα. **Άρα, πριν ξεκινήσει το οτιδήποτε, πρέπει να αρχικοποιήσουμε τη στοίβα, η οποία θα κρατάει χαρακτήρες μέσα της. Αμέσως μετά, βάζουμε τον βοηθητικό χαρακτήρα ‘\$’ εντός της στοίβας.** Παρακάτω, θα εξηγηθεί περεταίρω η χρήση του, αλλά και η βοήθεια που θα μας προσφέρει.

Για να αναλύσουμε μια έκφραση αποτελούμενη από ‘x’ και ‘y’, πρέπει πρώτα να την δούμε κατά γράμμα. **Άρα, θα φτιάξουμε μια επανάληψη, που εκτελείται ίσες φορές με τον αριθμό των γραμμάτων της έκφρασης.** Αν, για παράδειγμα έχουμε την έκφραση “xxxxxyyyyy”, θα φτιάξουμε μια δομή που θα επαναλαμβάνεται δέκα φορές.

Εντός της προαναφερθείσας επανάληψης, υπάρχουν δύο πιθανότητες: να πέσουμε σε χαρακτήρα ‘x’, ή ‘y’. **Όταν πέφτουμε σε χαρακτήρα ‘x’, τον βάζουμε στη στοίβα. Αντιθέτως, όταν πέφτουμε σε χαρακτήρα ‘y’, βγάζουμε το κορυφαίο στοιχείο της στοίβας.** Ύστερα των ελέγχων, προβάλλουμε στον χρήστη την παρούσα κατάσταση, το περιεχόμενο της στοίβας και το υπόλοιπο της εισόδου. Το υπόλοιπο της εισόδου, το αποθηκεύουμε σε άλλη μεταβλητή και το



προβάλλουμε με την μέθοδο του `substring`, ώστε να μην τροποποιηθεί το ίδιο το `string` της εισόδου.

Όταν τελειώσει η επανάληψη, θα μας βοηθήσει αρκετά ο χαρακτήρας '\$' με την παρουσία του. Συγκεκριμένα, θα ελέγξουμε το κορυφαίο στοιχείο της στοίβας. Θα εγκρίνουμε την έκφραση, αν και μόνο αν το κορυφαίο στοιχείο της στοίβας είναι το '\$' και η είσοδος έχει διαβαστεί. Σε οποιαδήποτε άλλη περίπτωση, η έκφραση απορρίπτεται, διότι δεν θα υπάρχει ίσος αριθμός 'x' και 'y' σε αυτή, ή θα υπάρχει, αλλά τα 'y' θα είναι περισσότερα από τα 'x' από τα αριστερά στα δεξιά.

## ΤΕΛΟΣ ΠΑΡΟΥΣΙΑΣΗΣ ΑΣΚΗΣΗΣ 1.

Σύνταξη παρουσίασης και δημιουργία άσκησης:  
Γεώργιος Σεϊμένης Π19204

---

# ΠΑΡΟΥΣΙΑΣΗ ΑΣΚΗΣΗΣ 2

---

Το θέμα 2 είχε ως σκοπό την δημιουργία μιας γεννήτριας συμβολοσειρών η οποία ακολουθεί τους παρακάτω γραμματικούς κανόνες :

$$\langle E \rangle ::= (\langle Y \rangle)$$
$$\langle Y \rangle ::= \langle A \rangle \langle B \rangle$$
$$\langle A \rangle ::= v \mid \langle E \rangle$$
$$\langle B \rangle ::= -\langle Y \rangle \mid +\langle Y \rangle \mid \varepsilon$$

Το πρόγραμμα έπρεπε , σύμφωνα με την θεωρία , να ξεκινήσει από το μη τερματικό σύμβολο  $\langle E \rangle$  και ακολουθώντας τους κανόνες να βγάλει ως έξοδο μια αλληλουχία από τερματικά σύμβολα.

---

## Προβλήματα

---

Από την εκφώνηση προέκυψαν τα εξής αλγοριθμικά προβλήματα :

- Μπορεί να γίνει τέτοια επιλογή συμβόλων για αντικατάσταση ώστε το πρόγραμμα να μην τερματίζεται ποτέ.
- Υπάρχουν γραμματικοί κανόνες με παραπάνω από μια επιλογές επομένως θα πρέπει να επιλέγεται μια από αυτές τυχαία.

---

## Επίλυση προβλημάτων

---

Για την επίλυση των παραπάνω προβλημάτων θα γίνουν οι εξής παραδοχές :

- Για να τερματίζεται το πρόγραμμα η γεννήτρια συμβολοσειρών θα θεωρήσουμε ότι θα σταματάει στις 50 επαναλήψεις και αν παραμένουν μη τερματικά σύμβολα, αυτά δεν θα αλλάξουν διότι το πρόγραμμα πρέπει να σταματήσει και αν τα αλλάζαμε στα αντίστοιχα τους μη τερματικά, αυτό θα πρόσβαλλε το απροσδόκητο αποτέλεσμα που θα έχει το πρόγραμμα

- Η τυχαία επιλογή συμβόλων αντικατάστασης θα γίνει με συνάρτηση (function) που είναι γεννήτρια τυχαίων αριθμών που κάθε αριθμός θα αντιστοιχεί σε μια επιλογή
- Το πρόγραμμα όντας γεννήτρια δεν επιδέχεται καμία είσοδο από τον χρήστη, έχει μόνο έξοδο

---

### Υλοποίηση

---

#### Γλώσσα προγραμματισμού: C++

Για την επίλυση του παραπάνω προβλήματος δημιουργήθηκε:

1. Μία τάξη (Class) "Symbol" η οποία διαθέτει :

- Ένα πεδίο (attribute ) συμβολοσειράς (string) με το όνομα Expression
- Μια συνάρτηση κατασκευαστή (constructor) που παίρνει ως όρισμα (overload) μια συμβολοσειρά (string) με το όνομα Expression και αρχικοποιεί το πεδίο (attribute) Expression
- Μια λογική συνάρτηση (bool) Checker η οποία ελέγχει, εάν υπάρχει μη τερματικό σύμβολο στη συμβολοσειρά. Αυτό επιτυγχάνεται με μία δομή επιλογής (if ) η οποία ελέγχει έναν-έναν τους χαρακτήρες από τα αριστερά στα δεξιά. Αν βρεθεί ο χαρακτήρας '<', αυτό σημαίνει ότι η έκφραση περιέχει μη τερματικό σύμβολο.
- Μια συνάρτηση τύπου void ReplaceCharacters η οποία αλλάζει την συμβολοσειρά (string) ως εξής: Αρχικά καλώντας την συνάρτηση Checker() ελέγχει εάν υπάρχει μη τερματικό σύμβολο στη συμβολοσειρά. Αν υπάρχει μη τερματικό σύμβολο, τότε με μια δομή επανάληψης (while) μετατοπίζεται στο αριστερότερο. Όταν βρεθεί το πιο αριστερό μη τερματικό σύμβολο τότε με μία δομή επιλογής (switch) ταυτοποιείται και κατόπιν ακολουθείται ο ανάλογος γραμματικός κανόνας. Στην περίπτωση που ως πιο αριστερό μη τερματικό σύμβολο έχουμε το <A> ή το <B> χρησιμοποιούμε την συνάρτηση (function) rand ( ) η οποία μπορεί να μας παράγει τυχαίους ακέραιους αριθμούς από το 0 μέχρι και ένα άνω φράγμα που θα επιλέξουμε εμείς . Έτσι στην περίπτωση του <A> η rand ( ) θα δώσει τον αριθμό 0 ή 1. Στο 0 η ReplaceCharacters θα αντικαταστήσει το <A> με το ν , ενώ στην αντίθετη περίπτωση με το <E> . Στην περίπτωση του <B> η rand ( ) θα δώσει τους αριθμούς από το 0 , το 1 ή το 2 . Στην περίπτωση που έχουμε 0 το <B> θα αντικατασταθεί από το +<Y> ,στην 1 θα αντικατασταθεί από το -<Y> , ενώ στην 2 θα διαγραφεί το σύμβολο <B> με την συνάρτηση ( function ) erase η οποία βρίσκεται στην βιβλιοθήκη string . Τέλος σε

κάθε διεργασία που η ReplaceCharacter () διενεργεί δίνει και τα ανάλογα μηνύματα στον χρήστη.

2. Στην συνάρτηση (function ) main έχουμε :

- Την αρχικοποίηση της συνάρτησης (function) rand με την τιμή του ρολογιού του επεξεργαστή, αυτό γίνεται με την βοήθεια της συνάρτησης (function) srand
- Την δημιουργία αντικειμένου (instance) της τάξης Symbol με το όνομα testSymbol και την αρχικοποίηση του Expression με το αρχικό σύμβολο <E>
- Το κάλεσμα της συνάρτησης (function) ReplaceCharacters ( ) με το αντικείμενο(instance) που δημιουργήθηκε παραπάνω
- Την έξοδο της παραχθείσας συμβολοσειράς

---

### Εκτέλεση

---

- Στην main δίνεται το ρολόι του επεξεργαστή στην συνάρτηση rand με την συνάρτηση srand.
- Γίνεται αρχικοποίηση(instantiation) του αντικειμένου testSymbol της Τάξης (Class) Symbol και τοποθετείται στη συνάρτηση κατασκευαστή (Constructor) της τάξης το μη τερματικό σύμβολο «<E>».
- Καλείται η συνάρτηση(function) ReplaceCharacters() η οποία ακολουθεί τους γραμματικούς κανόνες της εκφώνησης και οδηγείται σε ένα αποτέλεσμα εκτυπώνοντας τα βήματα τα οποία ολοκληρώνει.

---

### Σημειώσεις

---

Το πρόγραμμα έχει αναπτυχθεί κυρίως σε περιβάλλον windows. Όμως, έχει δοκιμαστεί και σε περιβάλλον Linux.

## ΤΕΛΟΣ ΠΑΡΟΥΣΙΑΣΗΣ ΑΣΚΗΣΗΣ 2.

Σύνταξη παρουσίασης και δημιουργία άσκησης:

Γεώργιος-Παναγιώτης Ξανθός Π19124

---

## ΠΑΡΟΥΣΙΑΣΗ ΑΣΚΗΣΗΣ 3

---

Στο αρχείο word

“Thema\_3\Check\_if\_grammar\_is\_LL(1).docx” βρίσκεται ο υπολογισμός των συνόλων FIRST,FOLLOW,EMPTY και LOOKAHEAD, η εύρεση των οποίων μας βοηθάει να καταλήξουμε σε ένα συμπέρασμα για το αν η γραμματική είναι LL(1).

Χρησιμοποιούνται δύο τρόποι για τον έλεγχο της γραμματικής.

Ο ένας αποτελεί τον έλεγχο ύπαρξης κοινών στοιχείων στα σύνολα LOOKAHEAD για τους κανόνες με ίδιο αριστερό μέλος. Καταλήγουμε στο

ότι δεν υπάρχουν κοινά στοιχεία σε αυτά τα σύνολα, άρα η γραμματική είναι LL(1).

Ο άλλος τρόπος, αποτελεί την δημιουργία του συντακτικού πίνακα. Παρακάτω φαίνεται ο συντακτικός πίνακας που

	(	)	$\alpha$	$\beta$	*	-	+	\$
S	$S \rightarrow (X)$							
X	$X \rightarrow YZ$		$X \rightarrow YZ$	$X \rightarrow YZ$				
Y	$Y \rightarrow S$		$Y \rightarrow \alpha$	$Y \rightarrow \beta$				
Z		$Z \rightarrow \epsilon$			$Z \rightarrow *X$	$Z \rightarrow -X$	$Z \rightarrow +X$	

αντιστοιχεί στη γραμματική

Παρατηρούμε πως δεν υπάρχει πάνω από ένας κανόνας σε κάποιο κελί, άρα η γραμματική είναι LL(1).

Στη συνέχεια, γίνεται επεξήγηση του αλγόριθμου και του σκεπτικού με τα οποία υλοποιήθηκε ο πηγαίος κώδικας c++ για την δημιουργία του συντακτικού αναλυτή topdown και του συντακτικού δέντρου. Προηγούνται αναφορές και στο εκτελέσιμο.

(το πρόγραμμα βρίσκεται στον προορισμό

Thema\_3\top\_down\_parser\top\_down\_parser.cpp

μαζί με το εκτελέσιμο και το header αρχείο που αναφέρεται παρακάτω. Τα παραπάνω έχουν υλοποιηθεί σε windows)

Να σημειωθεί πως έχουν χρησιμοποιηθεί τύποι μεταβλητών, συναρτήσεις, βιβλιοθήκες και εντολές όπως:

“wstring”, “wchar\_t”, “wcout”, “wcin”, “<fcntl.h> “  
“<io.h>“, “setmode(\_fileno(stdout/stdin), \_O\_WTEXT) “ κ.α.  
για την επίτευξη χρήσης ελληνικών γραμμάτων στη ροή εισόδου και εξόδου.



Κατά την εκκίνηση εκτέλεσης του προγράμματος, ζητείται από τον χρήστη να δώσει μια έκφραση έγκυρης μορφής:

```
Give an expression that contains only the following terminal characters: (, ), α, β, *, -, +
```

```
-
```

Στην περίπτωση λανθασμένης εισόδου εμφανίζεται το ανάλογο μήνυμα και ξαναζητείτε είσοδος από τον χρήστη:

```
Give an expression that contains only the following terminal characters: (, ), α, β, *, -, +  
α-γ
```

```
Incorrect expression format
```

```
Give an expression that contains only the following terminal characters: (, ), α, β, *, -, +
```

Ας κάνουμε επίδειξη της ζητούμενης έκφρασης της εκφώνησης  $((\beta - \alpha) * (\alpha + \beta))$ , για να εξηγήσουμε το σκεπτικό και στη συνέχεια τον αλγόριθμο:

Give an expression that contains only the following terminal characters: (, ), α, β, \*, -, +  
 $((\beta - \alpha) * (\alpha + \beta))$

Stack:	Input:	Table element:	Production:
\$S	$((\beta - \alpha) * (\alpha + \beta))\$$	$M(S, ($	$S \rightarrow (X)$
\$)X(	$((\beta - \alpha) * (\alpha + \beta))\$$		
\$)X	$(\beta - \alpha) * (\alpha + \beta))\$$	$M(X, ($	$X \rightarrow YZ$
\$)ZY	$(\beta - \alpha) * (\alpha + \beta))\$$	$M(Y, ($	$Y \rightarrow S$
\$)ZS	$(\beta - \alpha) * (\alpha + \beta))\$$	$M(S, ($	$S \rightarrow (X)$
\$)Z)X(	$(\beta - \alpha) * (\alpha + \beta))\$$		
\$)Z)X	$\beta - \alpha) * (\alpha + \beta))\$$	$M(X, \beta)$	$X \rightarrow YZ$
\$)Z)ZY	$\beta - \alpha) * (\alpha + \beta))\$$	$M(Y, \beta)$	$Y \rightarrow \beta$
\$)Z)Z\beta	$\beta - \alpha) * (\alpha + \beta))\$$		
\$)Z)Z	$-\alpha) * (\alpha + \beta))\$$	$M(Z, -)$	$Z \rightarrow -X$
\$)Z)X-	$-\alpha) * (\alpha + \beta))\$$		
\$)Z)X	$\alpha) * (\alpha + \beta))\$$	$M(X, \alpha)$	$X \rightarrow YZ$
\$)Z)ZY	$\alpha) * (\alpha + \beta))\$$	$M(Y, \alpha)$	$Y \rightarrow \alpha$
\$)Z)Z\alpha	$\alpha) * (\alpha + \beta))\$$		
\$)Z)Z	$) * (\alpha + \beta))\$$	$M(Z, )$	$Z \rightarrow \epsilon$
\$)Z)	$) * (\alpha + \beta))\$$		
\$)Z	$* (\alpha + \beta))\$$	$M(Z, *)$	$Z \rightarrow *X$
\$)X*	$* (\alpha + \beta))\$$		
\$)X	$(\alpha + \beta))\$$	$M(X, ($	$X \rightarrow YZ$
\$)ZY	$(\alpha + \beta))\$$	$M(Y, ($	$Y \rightarrow S$
\$)ZS	$(\alpha + \beta))\$$	$M(S, ($	$S \rightarrow (X)$
\$)Z)X(	$(\alpha + \beta))\$$		
\$)Z)X	$\alpha + \beta))\$$	$M(X, \alpha)$	$X \rightarrow YZ$
\$)Z)ZY	$\alpha + \beta))\$$	$M(Y, \alpha)$	$Y \rightarrow \alpha$
\$)Z)Z\alpha	$\alpha + \beta))\$$		
\$)Z)Z	$+\beta))\$$	$M(Z, +)$	$Z \rightarrow +X$
\$)Z)X+	$+\beta))\$$		
\$)Z)X	$\beta))\$$	$M(X, \beta)$	$X \rightarrow YZ$
\$)Z)ZY	$\beta))\$$	$M(Y, \beta)$	$Y \rightarrow \beta$
\$)Z)Z\beta	$\beta))\$$		
\$)Z)Z	$)\$$	$M(Z, )$	$Z \rightarrow \epsilon$
\$)Z)	$)\$$		
\$)Z	$)\$$	$M(Z, )$	$Z \rightarrow \epsilon$
\$)	$)\$$		
\$	$\$$		

Stack is empty, so the expression has been recognized

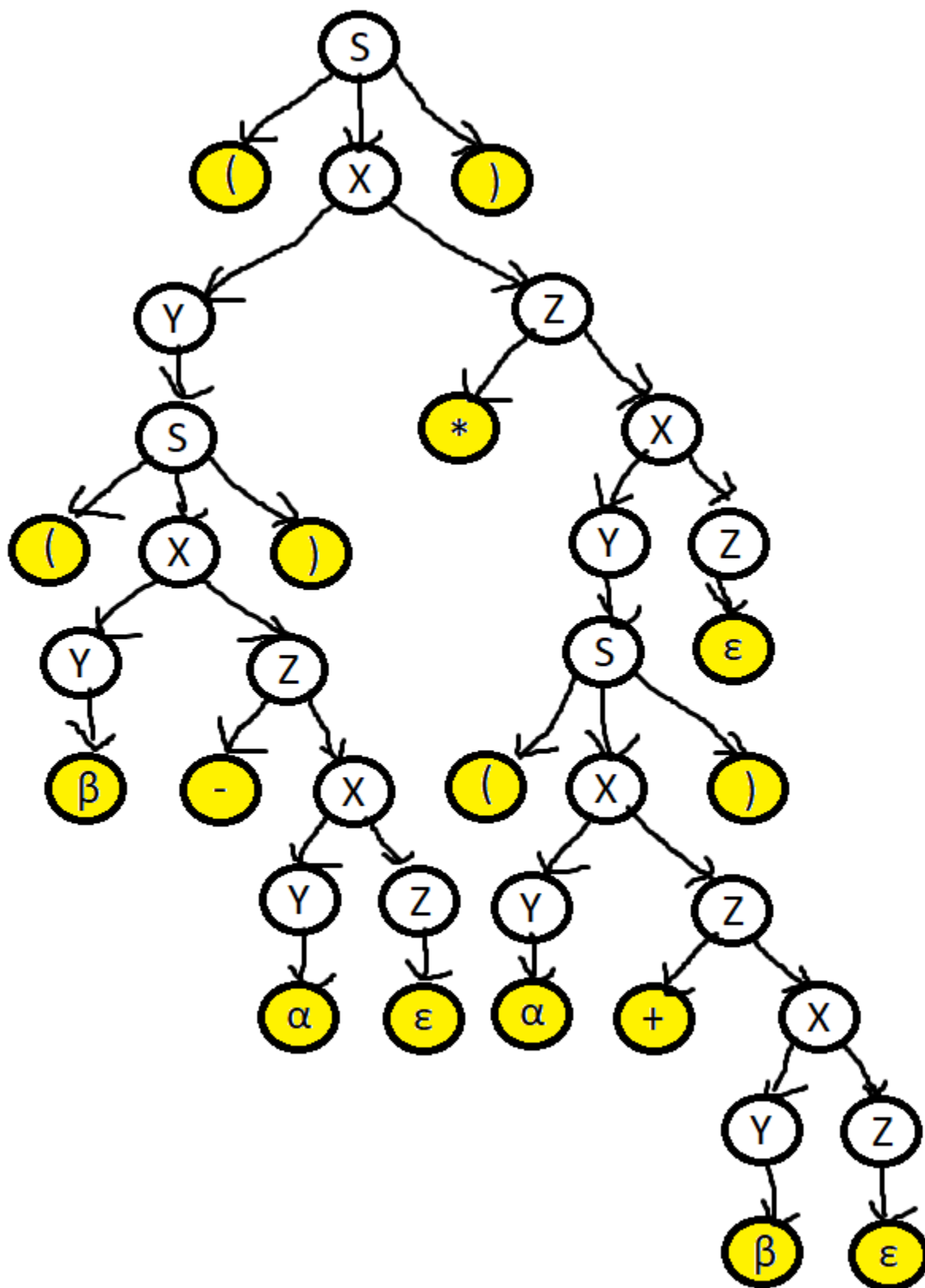
- Η πρώτη στήλη αντιστοιχεί στο περιεχόμενο της στοίβας σε κάθε βήμα.
- Η δεύτερη στήλη αντιστοιχεί στην είσοδο που διαβάζεται από το πρόγραμμα.
- Η τρίτη στήλη αντιστοιχεί στο κελί του συντακτικού πίνακα που ψάχνουμε έναν κανόνα.
- Η τέταρτη στήλη αντιστοιχεί στον κανόνα αντικατάστασης, εάν και εφόσον βρεθεί κάποιος στο κελί.

Στο τέλος, εμφανίζεται το σχετικό μήνυμα, ανάλογα με το αν αναγνωρίστηκε η έκφραση ή όχι.

Αν αναγνωρισθεί, εκτυπώνεται το συντακτικό δέντρο σε προ-,ενδο- και μεταδιάταξη:

```
Syntax tree output in preorder: S ( X Y S ( X Y β Z - X Y α Z ε ) Z * X Y S ( X Y α Z + X Y β Z ε ) Z ε )
Syntax tree output in inorder: ( S Y ( S Y β X - Z Y α X Z ε ) X * Z Y ( S Y α X + Z Y β X Z ε ) X Z ε )
Syntax tree output in postorder: ( ( β Y - α Y ε Z X Z X ) S Y * ( α Y + β Y ε Z X Z X ) S Y ε Z X Z X ) S
```

Το δέντρο παρουσιάζεται και γραφικά παρακάτω:



## Μη αναγνώριση της έκφρασης λόγω μη εύρεσης κανόνα:

```

Stack:      Input:      Table element:      Production:
$$          α-β$       M(S,α)
There is not any replacement rule on table index: [1][3] so the expression is not beeing recognized;

```

Μη αναγνώριση της έκφρασης λόγω μη κενής στοίβας στο τέλος:

```

Give an expression that contains only the following terminal characters: (, ), α, β, *, -, +
(
Stack:      Input:      Table element:      Production:
$$          ($         M(S,( )          S-->(X)
$)X(       ($
$)X        $
Stack is not empty, so the expression is not beeing recognized

```

## Σκεπτικό/Περιεχόμενα

- Γίνεται αρχικοποίηση ενός δυσδιάστατου πίνακα τύπου string, ο οποίος έχει την ίδια δομή με τον συντακτικό μας πίνακα:

	(	)	α	β	*	-	+	\$
S	$S \rightarrow (X)$							
X	$X \rightarrow YZ$		$X \rightarrow YZ$	$X \rightarrow YZ$				
Y	$Y \rightarrow S$		$Y \rightarrow \alpha$	$Y \rightarrow \beta$				
Z		$Z \rightarrow \epsilon$			$Z \rightarrow *X$	$Z \rightarrow -X$	$Z \rightarrow +X$	

```
wstring syntax_table[5][8] = {{L"", L"(", L")", L"α", L"β", L"*", L"-", L"+"}, /
                               {L"S", L")X("}, /
                               {L"X", L"ZY", L"", L"ZY", L"ZY"},
                               {L"Y", L"S", L"", L"α", L"β"},
                               {L"Z", L"", L"ε", L"", L"", L"X*", L"X-", L"X+"}};
```

Τα κελιά του πίνακα που δεν έχουν κάποιο string, αντιστοιχούν στο κενό string "". (Για τα περιεχόμενα γίνεται επεξήγηση παρακάτω)

- Χρησιμοποιούμε δύο βασικά string, τα οποία μεταβάλλονται κατά τη διάρκεια του προγράμματος (σε κάθε βήμα). Κάθε χαρακτήρας του string stack από τα αριστερά στα δεξιά, εκφράζει το περιεχόμενο της στοίβας. Συνεπώς, ο δεξιότερος χαρακτήρας εκφράζει το περιεχόμενο στην κορυφή της στοίβας:

```
wstring stack = L"$S"; //At the beginning, stack has only $ and S(starting character) characters
```

^^ Αρχικοποίηση της string μεταβλητής stack κατά την εκκίνηση.

Το string input περιέχει σε κάθε βήμα το υπόλοιπο της εισόδου που πρέπει να διαβαστεί:



`wstring` input; //a string variable to store user's input. We are going to edit this string along the procedure, in order to show the remaining input characters.

- Κατά την εκτέλεση του κύριου κομματιού του προγράμματος χρησιμοποιούνται επίσης οι μεταβλητές:

`wstring` stack\_top, input\_start; //strings to store stack's top and input's first characters

Οι οποίες σε κάθε βήμα, περιέχουν τον χαρακτήρα στην κορυφή της στοίβας και τον πρώτο χαρακτήρα εισόδου αντίστοιχα.

- Έχουμε επιλέξει στον συντακτικό πίνακα του προγράμματος να μην συμπεριλάβουμε την στήλη του “\$” επειδή είναι κενή και δεν θα είχε κάποια χρησιμότητα.

- Έχουμε επιλέξει τα περιεχόμενα του συντακτικού πίνακα του προγράμματος, να αποτελούν τα δεξιά μέλη των αντίστοιχων κανόνων αντικατάστασης. Οι χαρακτήρες τους βρίσκονται σε αντίστροφη σειρά, με στόχο να γίνεται άμεση αντικατάσταση του συμβόλου στην κορυφή της στοίβας με το string του πίνακα. Λόγω αυτού, γίνεται ανάποδη εκτύπωση των χαρακτήρων του κανόνα, όταν πρέπει να δείξουμε ποιος κανόνας αντικατάστασης εφαρμόστηκε σε ένα βήμα.
- Έχει χρησιμοποιηθεί header αρχείο, το οποίο περιέχει μία δομή κλάσης για τριαδικά δέντρα, η οποία θα μας βοηθήσει να φτιάξουμε το συντακτικό δέντρο.

Η κλάση περιέχει:

1.Την ιδιότητα τύπου `wchar_t` με όνομα `key` που αναπαριστά τον χαρακτήρα του κόμβου.

2.Τις ιδιότητες τύπου `ThreenodeTree*` με ονόματα `leftnode`, `middlenode`, `rightnode`

που αναπαριστούν δείκτες στον αριστερό, μεσαίο και δεξί κόμβο παιδί αντίστοιχα, ενός κόμβου.

3. Τις μεθόδους τύπου `void` με ονόματα `Preorder_Output`, `Inorder_Output` και `Postorder_Output` που εκτυπώνουν τα περιεχόμενα των κόμβων του δέντρου σε προδιάταξη, ενδοδιάταξη και μεταδιάταξη αντίστοιχα.

4. Την μέθοδο τύπου `static ThreenodeTree*` με όνομα `searchNewrootInpreorder` που επιστρέφει έναν δείκτη στον κόμβο που θα αναλυθεί στη συνέχεια. Ο κόμβος που αναλύεται έχει σαν κλειδί το μη τερματικό σύμβολο στην κορυφή της στοίβας, και αποκτάει για παιδιά, τους χαρακτήρες του κανόνα αντικατάστασης.

Περαιτέρω εξηγήσεις των μεθόδων γίνονται στο κομμάτι του αλγόριθμου πιο κάτω.

- Στο κύριο πρόγραμμα, για την δημιουργία του δέντρου, γίνεται αρχιοθέτηση ενός αρχικού pointer “tree” που δείχνει στον κόμβο ρίζα του δέντρου, ενός pointer “root” που δείχνει πάντα στον κόμβο που θα αναλυθεί σε ένα βήμα, και των pointer “rightnd”, “middlend” και “leftnd”

που δείχνουν στα παιδιά του root σε κάθε βήμα. Ο tree παραμένει σταθερός κατά τη διάρκεια του προγράμματος, ενώ ο root και οι υπόλοιποι αλλάζουν σε κάθε βήμα:

```
ThreenodeTree* tree = new ThreenodeTree ('S'); //make the first tree node, that has as key the starting character
```

^^Αρχειοθέτηση του tree

```
ThreenodeTree* root = tree; //root pointer is being used in order to point at the node we are going to expand
```

```
ThreenodeTree* rightnd = 0;  
ThreenodeTree* middlend = 0;  
ThreenodeTree* leftnd = 0; //Children node we are going to add at each expansion of a root node
```

^^Αρχική αρχειοθέτηση του root και των υπόλοιπων

Σε μεταγενέστερα βήματα, όταν πρέπει να γίνει αντικατάσταση μη τερματικού συμβόλου, ψάχνουμε τον

νέο κόμβο προς ανάλυση με τη βοήθεια της  
`searchNewrootInpreorder`

η οποία επιστρέφει τον καινούριο δείκτη στον `root`.

Όταν βρεθεί ο `root`, γίνεται προσθήκη παιδιών σε αυτόν. Τα παιδιά από αριστερά προς τα δεξιά, έχουν για κλειδιά τους χαρακτήρες του δεξιού μέλους του κατάλληλου κανόνα από τα δεξιά προς τα αριστερά (εφόσον στον πίνακα μας οι χαρακτήρες είναι αντίστροφα).

Στο τέλος, γίνεται εκτύπωση των κλειδιών του δέντρου με τη βοήθεια του δείκτη `tree` της ρίζας και των μεθόδων: `Preorder_Output`, `Inorder_Output` και `Postorder_Output` σε προ-, ενδο- και μετα- διάταξη.

## Αλγόριθμοι

- Αλγόριθμοι των μεθόδων του συντακτικού δέντρου:

i) Αλγόριθμος Αναγωγικής μεθόδου εκτύπωσης

Preorder\_Output:

1. Ξεκίνα από την ρίζα

2. Αν υπάρχει ο κόμβος:

- Εκτύπωσε το κλειδί

- Εκτέλεσε τον αλγόριθμο για το αριστερό υποδέντρο.

- Εκτέλεσε τον αλγόριθμο για το μεσαίο υποδέντρο.

- Εκτέλεσε τον αλγόριθμο για το δεξί υποδέντρο.

Αλλιώς, μην κάνεις τίποτα.

ii) Αλγόριθμος Αναγωγικής μεθόδου εκτύπωσης

Inorder\_Output:

1. Ξεκίνα από την ρίζα

2. Αν υπάρχει ο κόμβος:

- Εκτέλεσε τον αλγόριθμο για το αριστερό υποδέντρο.

- Εκτύπωσε το κλειδί

- Εκτέλεσε τον αλγόριθμο για το μεσαίο υποδέντρο.



-Εκτέλεσε τον αλγόριθμο για το δεξί υποδέντρο.

Αλλιώς, μην κάνεις τίποτα.

iii) Αλγόριθμος Αναγωγικής μεθόδου εκτύπωσης

Postorder\_Output:

1. Ξεκίνα από την ρίζα

2. Αν υπάρχει ο κόμβος:

-Εκτέλεσε τον αλγόριθμο για το αριστερό υποδέντρο.

-Εκτέλεσε τον αλγόριθμο για το μεσαίο υποδέντρο.

-Εκτέλεσε τον αλγόριθμο για το δεξί υποδέντρο.

-Εκτύπωσε το κλειδί

Αλλιώς, μην κάνεις τίποτα.

iv) Αλγόριθμος αναγωγικής μεθόδου αναζήτησης κόμβου  
root με κλειδί κ.

(υπενθυμίζεται ότι root είναι ο δείκτης στον κόμβο με κλειδί το μη τερματικό σύμβολο που αναλύεται):

1.Ξεκίνα από την ρίζα

2.Αν υπάρχει ο κόμβος:

- Αν ο κόμβος έχει το κλειδί  $k$  προς αναζήτηση και δεν έχει παιδιά, τότε είναι ο κόμβος προς ανάλυση. Οπότε, επέστρεψε αυτόν.

- Εκτέλεσε τον αλγόριθμο για το αριστερό υποδέντρο, και επέστρεψε το αποτέλεσμα στο  $p1$ . Αν ο  $p1$  δεν είναι 0 επέστρεψε  $p1$ . Αλλιώς, συνέχισε.

- Εκτέλεσε τον αλγόριθμο για το μεσαίο υποδέντρο, και επέστρεψε το αποτέλεσμα στο  $p2$ . Αν ο  $p2$  δεν είναι 0 επέστρεψε  $p2$ , αλλιώς συνέχισε.

- Εκτέλεσε τον αλγόριθμο για το δεξιό υποδέντρο, και επέστρεψε το αποτέλεσμα στο  $p3$ . Αν ο  $p3$  δεν είναι 0 επέστρεψε  $p3$ , αλλιώς συνέχισε.

- Αν  $p1, p2, p3$  επιστρέψουν 0, τότε επέστρεψε 0.

Αλλιώς, επέστρεψε 0.

- Αλγόριθμος για την δημιουργία του αναλυτή topdown και κατασκευής του συντακτικού δέντρου:

1. Θέσε `stack = "$S"` και πρόσθεσε στο τέλος του `input` το `"$"`.

2. Θέσε `root=tree` (υπενθυμίζεται ότι `tree` είναι ο δείκτης στον κόμβο ρίζα με κλειδί το `"S"`).

3. Θέσε `stack_top = stack[stack.length()-1]`

(ο τελευταίος χαρακτήρας του `stack` (κορυφή της στοίβας))

4. Θέσε `input_start = input[0]`

(ο πρώτος χαρακτήρας του `input`)

5. Τρέξε την αναγωγική μέθοδο αναζήτησης κόμβου `root` με κλειδί `stack_top` και θέσε στον `root` το αποτέλεσμα

6. **i)** Αν το `stack_top` είναι μη τερματικό σύμβολο και το `input_start` δεν είναι `"$"`, θέσε `i` τη γραμμή του συντακτικού πίνακα που βρίσκεται ο χαρακτήρας `stack_top` και `j` τη στήλη που βρίσκεται ο χαρακτήρας `input_start` και:

- Αν υπάρχει κανόνας στο κελί  $ij$ :
  - Ανάλογα με το πλήθος των χαρακτήρων του κανόνα, βάλε τόσα σε πλήθος κλαδιά στον root. Το κάθε κλαδί έχει έναν χαρακτήρα του κανόνα.
  - Σβήσε τον τελευταίο χαρακτήρα του stack.
  - Αν ο κανόνας δεν είναι ο “ε”, πρόσθεσε το string του κανόνα στο τέλος του stack. Πήγαινε πίσω στο βήμα 3.
  - Αλλιώς, ο κανόνας είναι ο “ε” οπότε μην προσθέσεις κάτι στο τέλος. Πήγαινε πίσω στο βήμα 3.
- Αλλιώς, δεν υπάρχει κανόνας στο κελί, οπότε η έκφραση δεν αναγνωρίζεται και τερματίζεται ο αλγόριθμος.

ii) Αλλιώς, αν το `stack_top` δεν είναι “\$” και το `input_start` δεν είναι “\$”, τότε:

`stack_top` = `input_start` = κάποιο τερματικό σύμβολο, οπότε σβήσε τον τελευταίο και τον πρώτο χαρακτήρα των `stack` και `input` αντίστοιχα. Πήγαινε πίσω στο βήμα 3.

- iii) Αλλιώς, αν  $stack\_top \neq '\$'$  και  $input\_start = '\$'$ , το input διαβάστηκε και η στοίβα είναι μη κενή, άρα η έκφραση δεν αναγνωρίζεται και ο αλγόριθμος τερματίζεται.
- iv) Αλλιώς, αν  $stack\_top = '\$'$  και  $input\_start \neq '\$'$ , το input δεν διαβάστηκε και η στοίβα είναι κενή, άρα η έκφραση δεν αναγνωρίζεται και ο αλγόριθμος τερματίζεται.
- v) Αλλιώς,  $stack\_top = input\_start = '\$'$ , το input διαβάστηκε και η στοίβα είναι κενή, άρα η έκφραση αναγνωρίζεται και ο αλγόριθμος τερματίζεται.

## ΤΕΛΟΣ ΠΑΡΟΥΣΙΑΣΗΣ ΑΣΚΗΣΗΣ 3.

Σύνταξη παρουσίασης και δημιουργία άσκησης:

Νικόλαος Γεωργιάδης Π19032

---

## ΠΑΡΟΥΣΙΑΣΗ ΑΣΚΗΣΗΣ 4

---

Στο αρχείο:

"Thema\_4\BNF,EBNF\_description.pdf" υπάρχει η περιγραφή BNF και EBNF της γραμματικής. Εξηγείται αναλυτικά ο συλλογισμός με τον οποίο φτάσαμε στην υλοποίηση του αποτελέσματος.

Στο αρχείο:

"Thema\_4\Syntax\_diagram.png" φαίνεται το συντακτικό διάγραμμα της έκφρασης:

$$E ::= X \mid "Y O Y \{ O Y \}";$$

Η οποία αποτελεί μία αναπαράσταση της ζητούμενης κανονικής έκφρασης σε μορφή EBNF.

Στο αρχείο με προορισμό:

"Thema\_4\flex program\thema\_4.l"



βρίσκεται το πηγαίο πρόγραμμα σε flex μαζί με το εκτελέσιμο τα οποία υλοποιήθηκαν σε windows.

## Εκτελέσιμο:

```
Give some variable declaration expressions.Examples:'x=random_name+4;', 'my_var =5 +1/y;'
Types as 'x=y;' are not accepted.Zero constant cant be contained.
Variable names cant start with number and they must contain only these characters:A-Z,a-z,1-9 and '_'.
```

```
56
Unrecognized character: 5
Unrecognized character: 6
x=i+7;
The following expression: x=i+7; is acceptable
r=p-0;
0 constant is not allowed.Expression was: r=p-0;
g=a+9
You forgot the ';' symbol in the following expression: g=a+9
=g+0;
You forgot some constants or/and variables in the following expression: =g+0;
r+=k;
You forgot some constants or/and variables in the following expression: r+=k;
g=y%;
You forgot some constants or/and variables in the following expression: g=y%;
k=*a+1;
You forgot some constants or/and variables in the following expression: k=*a+1;
t=3+j*j/1%7-k;
The following expression: t=3+j*j/1%7-k; is acceptable
a = 8+1;_
```

Κατά την εκτέλεση του προγράμματος, ζητείται είσοδος από τον χρήστη. Δίνονται μερικά παραδείγματα δεκτών τύπων εκφράσεων από το πρόγραμμα και άλλα βοηθητικά σχόλια.

Κάθε φορά, που ο χρήστης γράφει μία γραμμή και πατάει enter, εμφανίζεται το ανάλογο αποτέλεσμα της έκφρασης του.

Πιο συγκεκριμένα:

- Γίνονται δεκτές οι εκφράσεις με κενά.
- Εμφανίζεται σχετικό μήνυμα, αν ο χρήστης ξεχάσει το “;” στο τέλος.
- Εμφανίζεται σχετικό μήνυμα, αν ο χρήστης συμπεριλάβει μηδενικό στην έκφρασή του.
- Εμφανίζεται σχετικό μήνυμα, αν ο χρήστης ξεχάσει κάποια πιθανή σταθερά ή όνομα μεταβλητής στην έκφραση του.
- Εμφανίζεται σχετικό μήνυμα αν ο χρήστης δώσει μια έκφραση της μορφής: “ $x=y$ ”.

Αν δεν ανιχνευθεί καποιο από τα παραπάνω σφάλματα, το/τα σύμβολο/α θεωρούνται μη αναγνωρίσιμα.

## Σκεπτικό/Περιεχόμενα

- Η μεταβλητή θα πρέπει να ξεκινάει με γράμμα ή “\_” και να ακολουθείται από μηδέν ή περισσότερες φορές με γράμμα, “\_” ή αριθμό 1-9.

Αυτό αντιστοιχεί στην έκφραση:

`[a-zA-Z_][a-zA-Z1-9_]*`

Την οποία ονομάζουμε X.

- Δεξιά από το “=” της έκφρασης, θέλουμε να υπάρχει μεταβλητή ή σταθερά. Σύμφωνα με την παραπάνω μορφή της μεταβλητής, μπορούμε να πούμε πως η έκφραση για την μεταβλητή ή σταθερά είναι:

`[a-zA-Z_][a-zA-Z1-9_]* | [1-9]`

και την ονομάζουμε Y.

- Χρειαζόμαστε, επίσης, μία έκφραση για τους τελεστές δεξιά του “=”. Η έκφραση: `[+/%*-]` σημαίνει ένας από τους τελεστές μία ακριβώς φορά και ανταποκρίνεται σε αυτό που θέλουμε.

Ας ονομάσουμε O την `[+/%*-]`

- Μπορούμε να χρησιμοποιήσουμε την έκφραση:  
[ \t\n] που σημαίνει ένας από τους χαρακτήρες tab,space ή αλλαγή γραμμής μία ακριβώς φορά, για να αγνοηθούν αυτά αν υπάρχουν πριν ή μετά την έκφραση, που δίνει ο χρήστης.  
Ας πούμε W1 την [ \t\n]
- Τέλος, μπορούμε να συμπεριλάβουμε την έκφραση: [ \t] που σημαίνει ένας από τους χαρακτήρες tab ή space μία ακριβώς φορά, ανάμεσα σε άλλες εκφράσεις που θα δημιουργήσουμε, για να αγνοηθούν τα κενά στις εκφράσεις που δίνει ο χρήστης.

Ονομάζουμε W2 την [ \t]

Αφού ορίσαμε όλες τις εκφράσεις που μας χρειάζονται, τις συνδιάζουμε για να παραχθούν οι κανόνες στο τμήμα κανόνων.

Οι πρώτοι 5 και ο 7<sup>ος</sup> κανόνες, θα εκτυπώνουν μήνυμα, όταν εντοπιστεί η έκφραση ενώ ο 6ος θα χρησιμοποιείται για να αγνοηθούν χαρακτήρες.

1) Ο πρώτος κανόνας θα είναι για την έγκυρη μορφή της έκφρασης του χρήστη:

$$\{X\}\{W2\}^* "=" \{W2\}^* \{Y\}\{W2\}^* (\{O\}\{W2\}^* \{Y\}\{W2\}^*) + ";"$$

Όλη η έκφραση, θα πρέπει να περιέχει με τη σειρά: Μία φορά την κανονική έκφραση X (μεταβλητή), μηδέν ή περισσότερες την W2 (κενά ή tab), μία ακριβώς το "=", μηδέν ή περισσότερες την W2, μία φορά την Y (μεταβλητή ή σταθερά), μηδέν ή περισσότερες την W2, μία ή περισσότερες την εξής: «μία φορά το O (τελεστής), μηδέν ή περισσότερες την W2, μία το Y και μηδέν ή περισσότερες φορές

την W2». Στο τέλος, μία φορά το ";".

Να σημειωθεί, ότι η έκφραση  $\{O\}\{W2\}^* \{Y\}\{W2\}^*$  πρέπει να εμφανίζεται, αναγκαστικά, τουλάχιστον μία φορά (για αυτό βάζουμε ένα + δεξιά της) μετά το πρώτο  $\{Y\}\{W2\}^*$  αφού εκφράσεις της μορφής  $x=y$  δεν είναι έγκυρες.

2) Ο δεύτερος κανόνας θα είναι για παρόμοιες εκφράσεις με τις παραπάνω:

$$\{X\}\{W2\}^* = \{\{W2\}^*({Y}|"0")\{W2\}^*({O}\{W2\}^*({Y}|"0")\{W2\}^*)+";";$$

Η διάφορα είναι ότι στη θέση του Y (μεταβλητή ή σταθερά) μπορεί να περιέχεται και το 0 (για αυτό λέμε  $\{Y|"0"$ ) αντί για {Y}, που σημαίνει Y ή 0) .

Να σημειωθεί, ότι αυτή η περίπτωση μπορεί να συμπίπτει με την πρώτη, αν σε όλα τα σημεία του  $\{Y|''0''\}$  έχουμε {Y}. Ωστόσο, δεν υπάρχει σύγκριση, διότι ο πρώτος κανόνας προηγείται.

3) Ο τρίτος κανόνας θα είναι για εκφράσεις τύπου  $x=y$ :

$$\{X\}\{W2\}^* = \{\{W2\}^*({Y}|"0")\{W2\}^*[:];?$$

Παρατηρούμε ότι είναι το ίδιο με το παραπάνω, απλώς χωρίς το  $\{O\}\{W2\}^*({Y}|"0")\{W2\}^*+ ,$  επειδή δεν θέλουμε την ακολουθία τελεστή και  $\{Y|0\}$  σε αυτή τη περίπτωση.

Να σημειωθεί, ότι δεν μας ενδιαφέρει αν υπάρχει ";" στο τέλος για να αναγνωριστεί αυτός ο τύπος έκφρασης, οπότε γράφουμε  $[:];?$  (μηδέν ή μία φορά το ";" ). Όπως επίσης, δεν μας ενδιαφέρει αν το Y έχει τον ρόλο του 0, καθώς είναι μία περίπτωση λανθασμένης έκφρασης.

4) Ο τέταρτος κανόνας είναι ο ίδιος με τον πρώτο, απλά χωρίς το “;”:

$$\{X\}\{W2\}^* = \{W2\}^* (\{Y\} | "0") \{W2\}^* (\{O\}\{W2\}^* (\{Y\} | "0") \{W2\}^*) +$$

Αυτός ο κανόνας αναγνωρίζει εκφράσεις σαν τον πρώτο, απλά χωρίς “;”.

(Το Y μπορεί να είναι και 0)

5) Ο πέμπτος κανόνας θα είναι για εκφράσεις, στις οποίες παραλείπεται μία ή περισσότερες μεταβλητές/σταθερές. Ωστόσο, υπάρχει τουλάχιστον ένας τελεστής δεξιά του “=”:

$$\{X\}?\{W2\}^* = \{W2\}^* (\{Y\} | "0")?\{W2\}^* (\{O\}\{W2\}^* (\{Y\} | "0")?\{W2\}^*) + [;]?$$

Για αυτόν τον λόγο, έχουμε βάλει “?” στα {X} και ({Y} | "0") ακόμα και στο “;” (λόγω λανθασμένης έκφρασης), επειδή σε αυτή την περίπτωση, μπορεί να υπάρχουν, μπορεί και όχι. Να σημειωθεί, ότι και αυτός ο κανόνας μπορεί να συμπίπτει με τον πρώτο, αλλά λόγω προτεραιότητας του πρώτου, ούτε εδώ υπάρχει σύγκληση.

6) Ο έκτος κανόνας θα είναι για τα κενά και την αλλαγή γραμμής, που μπορεί να υπάρχουν πριν ή μετά την έκφραση που δίνει ο χρήστης:

$$\{W1\} +$$

Ο κανόνας σημαίνει space, tab ή αλλαγή γραμμής μία ή περισσότερες φορές. Όταν εντοπιστεί, δεν θα εκτυπώνεται κάποιο μήνυμα και απλά θα «τρώει» τους χαρακτήρες.

7) Ο έβδομος κανόνας θα είναι για οποιοδήποτε άλλο χαρακτήρα/ες που δεν ικανοποιεί/ούν τους παραπάνω κανόνες:

.

Η τελεία σημαίνει οποιοσδήποτε χαρακτήρας.

Επιθυμούμε, κατά την εκτέλεση του προγράμματος, να εμφανίζεται ένα βοηθητικό μήνυμα και να αναμένεται είσοδος από τον χρήστη, οπότε προσθέτουμε μία εντολή `printf` πριν την `yylex()`;

## **ΤΕΛΟΣ ΠΑΡΟΥΣΙΑΣΗΣ ΑΣΚΗΣΗΣ 4.**

Σύνταξη παρουσίασης και δημιουργία άσκησης:

**Νικόλαος Γεωργιάδης Π19032**

---



# ΠΑΡΟΥΣΙΑΣΗ ΑΣΚΗΣΗΣ 5

---

Η παρουσίαση αυτή έχει ως σκοπό την αναλυτική εξήγηση του 5<sup>ου</sup> θέματος της εργασίας στο μάθημα των μεταγλωττιστών , καθώς και την ανάλυση στον τρόπο σκέψης και πως αυτό μεταφράζεται σε κώδικα.

Το "Thema5" αρχείο αντιπροσωπεύει τον κώδικα που έχει γραφτεί χρησιμοποιώντας visual studio code και το "a" αρχείο το εκτελέσιμο κομμάτι του παραπάνω κώδικα. Το πρόγραμμα έχει γραφτεί σε γλώσσα Flex με βάση αυτά που διδαχτήκαμε στο φετινό μάθημα του εξαμήνου μας.

---

## ΠΛΗΡΟΦΟΡΙΕΣ ΚΑΙ ΠΑΡΑΤΗΡΗΣΕΙΣ

Ο σκοπός του προγράμματός μας είναι να **δέχεται μία φράση που θα αντιπροσωπεύει ένα γεωμετρικό σχήμα και δίπλα τα σημεία** (ή αλλιώς τις κορυφές) που το συνοδεύουν και να κρίνει αν αυτή η έκφραση είναι γεωμετρικά αποδεκτή.

Να σημειωθεί εδώ ότι τα γράμματα που μπορούν να δοθούν ως κορυφές είναι: **A,B,C,D,E,F,G,H**

Για να ξεκινήσουμε είναι σημαντικό να κατανοήσουμε μερικά πράγματα πρώτα:

Για παράδειγμα **ένα τρίγωνο συνοδεύεται από 3 μοναδικές κορυφές** ,και μόνο 3,αλλιώς δεν λέγεται τρίγωνο. Οπότε μια φράση του τύπου «*triangle ABC*» είναι γεωμετρικά **σωστή**.

Μια φράση του τύπου «*triangle ABCD*» είναι σαφώς **λανθασμένη** καθώς ένα τρίγωνο δεν μπορεί να έχει πέρα από 3 κορυφές.

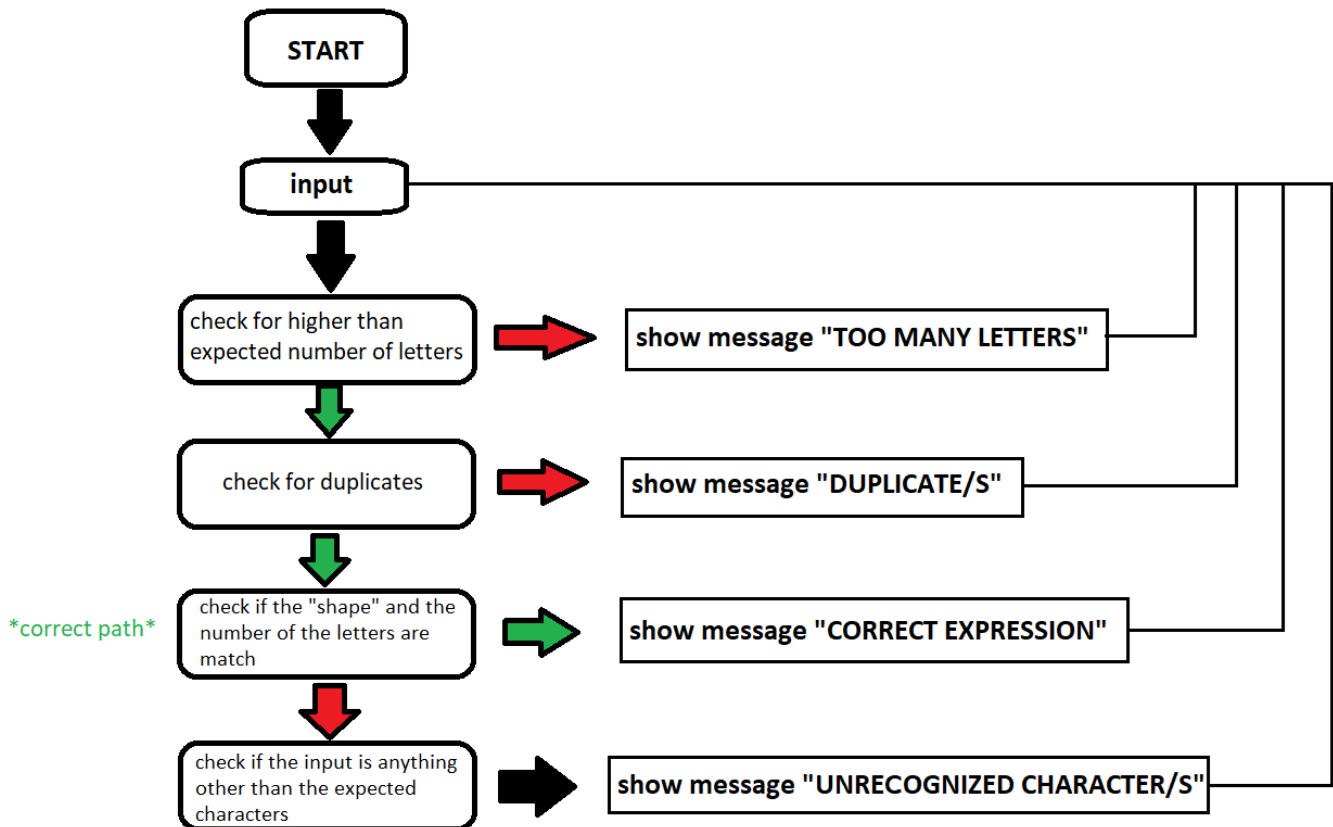
Μια ακόμη **λανθασμένη** έκφραση θα ήταν «*triangle AAC*» καθώς η κάθε κορυφή πρέπει να είναι μοναδική.

Με βάση τα παραπάνω καταλήγουμε σε δύο πολύ απλά συμπεράσματα:

1. Κάθε ν-γώνο πρέπει να συνοδεύεται από τον αντίστοιχο αριθμό κορυφών του, δηλαδή ν.
2. Κάθε κορυφή (δηλαδή γράμμα) μπορεί να εμφανιστεί μόνο ΜΙΑ ΦΟΡΑ.

Οπότε το πρόγραμμα μας θα πρέπει να απορρίπτει οτιδήποτε εκτός από αυτά!

### ΣΥΛΛΟΓΙΣΤΙΚΗ ΠΟΡΕΙΑ / ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



**\*Το πρόγραμμα σταματά να δέχεται είσοδο με την εντολή ctrl+c**

## ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΚΟΜΜΑΤΙ

Στο πρώτο τμήμα το πρόγραμμα θα ελέγξει αν υπάρχουν τυχόν «περισσευούμενα» γράμματα στο μήνυμα του χρήστη.

και επίσης αξιοποιούμε τον πρώτο κανόνα που θέσαμε:

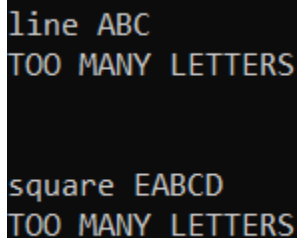
**«Κάθε ν-γώνο πρέπει να συνοδεύεται από τον αντίστοιχο αριθμό κορυφών του, δηλαδή ν.»**

Οι εκφράσεις που χρησιμοποιούμε εδώ μεταφράζονται ως εξής:

«point +οποιοδήποτε από τα επιτρεπτά γράμματα δύο ή περισσότερες φορές»

Και φυσικά αν αυτό ισχύει θα εμφανίσει το μήνυμα «TOO MANY LETTERS» στο χρήστη, ενημερώνοντας τον πως έχει χρησιμοποιήσει **περισσότερα** απ' τα γράμματα που αντιστοιχούν στη λέξη.

(χρησιμοποιούμε τις αντίστοιχες για να καλύψουμε και τις υπόλοιπες περιπτώσεις όπως «square +οποιοδήποτε από τα επιτρεπτά γράμματα πέντε ή περισσότερες φορές» όπως φαίνεται και στο ακόλουθο παράδειγμα)



```
line ABC
TOO MANY LETTERS

square EABCD
TOO MANY LETTERS
```

---

Αφού κάναμε τον έλεγχο για «περισσευούμενα» γράμματα, τώρα θα ελέγξουμε για διπλά, τριπλά, και λοιπά.

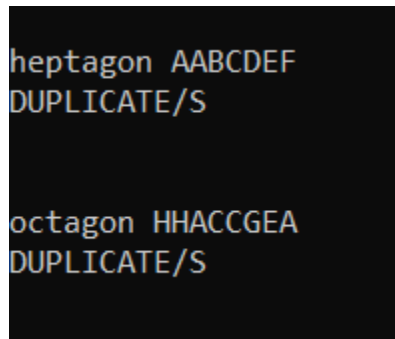
Οι εκφράσεις που χρησιμοποιούμε μεταφράζονται ως εξής:

«οτιδήποτε από τα BCDEFGH μηδέν ή περισσότερες φορές, A, οτιδήποτε από τα BCDEFGH μηδέν ή περισσότερες φορές» και το ακόλουθο: «A, οτιδήποτε από τα

BCDEFGH μηδέν ή περισσότερες φορές» μπορεί να επαναληφθεί μία ή περισσότερες φορές.

Η γενική σημασία της πρότασης με άλλα λόγια σημαίνει δύο ή περισσότερα Άλφα, σε μία έκφραση που περιέχει τα γράμματα A-H. Η παραπάνω μορφή, χρησιμοποιείται για τον έλεγχο επανάληψης και στα υπόλοιπα γράμματα.

**Αν εμφανιστεί οποιοδήποτε γράμμα δύο ή περισσότερες φορές θα εμφανίσει το μήνυμα «DUPLICATE/S».**



```
heptagon AABCDEF
DUPLICATE/S

octagon HHACCGEA
DUPLICATE/S
```

\*Στην αρχή, ελέγχεται αν έχουμε τον σωστό αριθμό γραμμάτων για την κάθε λέξη. Σε αυτή τη περίπτωση έχουμε, αλλά υπάρχουν duplicates.\*

---

**Αφού έχουν ολοκληρωθεί όλοι οι παραπάνω έλεγχοι, έφτασε η ώρα για τα αποτελέσματα μας..**

Εφόσον τώρα ξέρουμε ότι ΔΕΝ υπάρχουν διπλά γράμματα ΚΑΙ για κάθε λέξη έχουμε τον αντίστοιχο αριθμό γραμμάτων μπορούμε να κάνουμε τον εξής έλεγχο!

Χρησιμοποιούμε τις εκφράσεις:

«point + οποιοδήποτε από τα επιτρεπόμενα γράμματα»

«line + οποιοδήποτε δύο από τα επιτρεπόμενα γράμματα»

«triangle + οποιοδήποτε τρία από τα επιτρεπόμενα τρία γράμματα»

«square + οποιοδήποτε τέσσερα από τα επιτρεπόμενα γράμματα»

...

«octagon+ όλα τα επιτρεπόμενα γράμματα»

Και μόλις βρεθεί η περίπτωση που αντιστοιχεί στο μήνυμα του χρήστη θα εμφανίσει το μήνυμα «CORRECT EXPRESSION»!!

```
triangle HGB  
CORRECT EXPRESSION  
  
octagon HGFEDCBA  
CORRECT EXPRESSION  
  
pentagon ACBED  
CORRECT EXPRESSION
```

---

Η τελική έκφραση που χρησιμοποιούμε έχει την εξής σημασία:

«οτιδήποτε άλλο εκτός από αλλαγή γραμμής, μία ή περισσότερες φορές»

Με άλλα λόγια μπορούμε να την εκφράσουμε πιο απλά «οτιδήποτε άλλο»(εκτός από τις αποδεκτές εκφράσεις βέβαια)

```
qewtqwg  
UNRECOGNIZED CHARACTER/S  
  
point Awaegwg  
UNRECOGNIZED CHARACTER/S  
  
qwgwegpoint A  
UNRECOGNIZED CHARACTER/S  
  
point qwffwf A  
UNRECOGNIZED CHARACTER/S
```

Όπως βλέπουμε οποιαδήποτε άλλη δήλωση πέρα από τις αποδεκτές θα απορριφθεί εμφανίζοντας το μήνυμα «UNRECOGNIZED CHARACTER/S», ακόμα και αν εμπεριέχει αποδεκτή έκφραση σε κάποιο σημείο της.

## Τερματικό

Όταν ο χρήστης ξεκινήσει την εκτέλεση του προγράμματος, το εκτελέσιμο αρχείο θα έχει τη μορφή:

```
C:\Users\Lambrakos\Documents\GitHub\Metaglottistes\Thema_5\a.exe
This program will accept any Point,Line or Shape given, using the following letters: A,B,C,D,E,F,G,H.
It is not necessary to use all of them.
Some acceptable examples are 'triangle ABC'/'square ACHG'/'line HE' and some unacceptable examples are
'square ABC'/'octagon EDCBA'
EACH LETTER CAN BE USED ONE TIME
The acceptable references are: point, line, triangle, square, pentagon, hexagon, heptagon, octagon
USE SPACE BETWEEN THE WORD AND THE LETTERS A-H. Any other expression will be considered unacceptable. e
tc 'pentagon ABCDE'
```

Το τμήμα της `int main()` εμπεριέχει όλα τα παραπάνω μηνύματα που εμφανίζονται στον χρήστη καθώς και την εντολή `yylex()`; η οποία είναι υπεύθυνη για τη αποδοχή έκφρασης από τον χρήστη. Η εκτέλεση των μηνυμάτων γίνεται με τη χρήση της εντολής `printf("μήνυμα");`

Αξίζει να σημειωθεί εδώ ότι η Flex είναι περιορισμένη γλώσσα και όταν χρησιμοποιούμε τον όρο «οτιδήποτε», εννοούμε τους χαρακτήρες ASCII.

## ΤΕΛΟΣ ΠΑΡΟΥΣΙΑΣΗΣ ΑΣΚΗΣΗΣ 5.

Σύνταξη παρουσίασης και δημιουργία άσκησης:  
Λαμπράκος Ελευθέριος Π19085