

# Απαλλακτική Εργασία

## Μεταγλωττιστών 2020-2021

**Ονοματεπώνυμο και Α.Μ:**

Αγγελική Αποστόλου Π19015

Αντωνία Νικολέττα Γιαννουλάκη Π19251

Ευστάθιος Λαζαράκης Π19083

### Άσκηση 1

Για την κατανόηση της άσκησης, πρέπει να καταλάβουμε πως δουλεύει το ντετερμινιστικό αυτόματο στοίβας.

- 1) Η στοίβα είναι μια δομή δεδομένων, γνωστή και ως LIFO (Last In First Out), που χρησιμοποιείται σαν μνήμη για το αυτόματο.

Στην προκειμένη άσκηση, μας ζητάει να αναγνωρίζονται

Οι χαρακτήρες 'x' και 'y', με τις προϋποθέσεις ότι:

- όσοι χαρακτήρες 'x' εμφανίζονται συνολικά, άλλοι τόσοι χαρακτήρες 'y' εμφανίζονται συνολικά
- κοιτάζοντας την έκφραση από αριστερά προς τα δεξιά, οι χαρακτήρες 'y' δεν είναι ποτέ περισσότεροι από τους χαρακτήρες 'x'.

Για να αναγνωρίζεται η έκφραση, πρέπει i) να ξεκινάει πάντα με τον χαρακτήρα 'x' ii) όσοι χαρακτήρες 'x' υπάρχουν, άλλοι τόσοι 'y' θα υπάρχουν.

Ο χαρακτήρας 'x' γίνεται **PUSH** στην στοίβα (θα τοποθετηθεί στην κορυφή της στοίβας) ενώ ο χαρακτήρας 'y' θα κάνει **POP** (δηλαδή θα αφαιρέσει το στοιχείο που βρίσκεται στην κορυφή της στοίβας).

## Υλοποίηση προγράμματος

Γλώσσα Προγραμματισμού: C++

Οι 4 κύριες συναρτήσεις για την υλοποίηση του προγράμματος είναι:

- 1.push()
- 2.pop()
- 3.empty()
- 4.top()

## Επεξήγηση

Ζητάμε από τον χρήστη να δώσει μία συμβολοσειρά και τη βάζουμε στη μεταβλητή `symbseira` τύπου `string`. Έστερα βάζουμε σε μία μεταβλητή `something` τύπου `int` το μέγεθος της συμβολοσειράς αυτής, έτσι ώστε να έχουμε το όριο των επαναλήψεων που θα πραγματοποιηθούν και που θα εξηγήσουμε παρακάτω.

Ορίζουμε μία στοίβα με χαρακτήρες και την ονομάζουμε `s`, μια μεταβλητή `check` τύπου `bool` που τη βάζουμε `true` και άλλες 2 μεταβλητές `counterx` και `countery` τύπου `int` που είναι ίσες με 0 και ο ρόλος τους είναι να μετράνε τα `x` και τα `y` αντίστοιχα.

Αρχικά, με μία `if` ελέγχουμε αν το 1<sup>ο</sup> στοιχείο της συμβολοσειράς (`symbseira[0]`) είναι `x`.

Αν `symbseira[0] == 'x'` τότε εμφανίζεται το κατάλληλο μήνυμα και η εκτέλεση συνεχίζεται ως εξής:

Ξεκινά μία επανάληψη `for` με μια μεταβλητή `int i` (που αρχικά ισούται με το 0) να αυξάνεται κατά 1 όσο ισχύει η συνθήκη `i < something` (όπου `something` -> το μέγεθος της συμβολοσειράς, όπως προαναφέραμε).

Μέσα στην επανάληψη γίνονται τα εξής:

Αν το εκάστοτε στοιχείο της συμβολοσειράς (`symbseira[i]`) ισούται με `x` ή `y` τότε εμφανίζεται το κατάλληλο μήνυμα. Στη συνέχεια, αν το στοιχείο είναι `x` τότε το εισάγουμε στη στοίβα, αυξάνεται ο `counter` κατά 1 και εμφανίζεται το κατάλληλο μήνυμα. Αν το στοιχείο είναι `y` τότε εμφανίζεται το κατάλληλο μήνυμα, ο `countery` αυξάνεται κατά 1 και με μία `if` ελέγχεται αν η στοίβα είναι κενή. Αν δεν είναι κενή τότε αφαιρούμε ένα στοιχείο από τη στοίβα και εμφανίζεται το κατάλληλο μήνυμα. Αν είναι κενή τότε εμφανίζεται το

κατάλληλο μήνυμα και το πρόγραμμα βγαίνει από την επανάληψη.

Αν το εκάστοτε στοιχείο της συμβολοσειράς (`symbseira[i]`) δεν είναι ούτε `x` ούτε `y` τότε εμφανίζεται το κατάλληλο μήνυμα και το πρόγραμμα βγαίνει από την επανάληψη.

Αν `symbseira[0] == 'γ'` τότε εμφανίζεται το κατάλληλο μήνυμα και η μεταβλητή `check` (που στην αρχή την είχαμε ορίσει `true`) γίνεται `false`.

Τέλος, με μία `if` τσεκάρουμε αν η συμβολοσειρά είναι αποδεκτή.

Αν η στοίβα είναι άδεια (`s.empty()`) και τα `x` είναι ίσα με τα `y` (`counterx == countery`) και η `check` είναι `true` (δηλαδή η συμβολοσειρά δεν ξεκινάει με `y`) τότε η συμβολοσειρά είναι αποδεκτή. Σε οποιαδήποτε άλλη περίπτωση η συμβολοσειρά είναι μη αποδεκτή.

## Άσκηση 2

Η άσκηση μας δίνει μια γραμματική σε μορφή BNF:

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

Για μεγαλύτερη ευκολία, μπορούμε να μετατρέψουμε την BNF γραμματική σε κανονική γραμματική παραγωγής, δηλαδή:

$$E \rightarrow (Y)$$
$$Y \rightarrow AB$$
$$A \rightarrow v \mid E$$
$$Z \rightarrow -Y \mid +Y \mid \varepsilon$$

Τα σύμβολα  $E, Y, A, B$  και  $Z$  είναι **μη τερματικά σύμβολα** ενώ τα σύμβολα  $(, ), v, -, +$  και  $\varepsilon$  είναι **τερματικά σύμβολα**.

## Υλοποίηση προγράμματος

Γλώσσα προγραμματισμού: C++

### Επεξήγηση

Ορίζουμε ένα string  $s$  να είναι ίσο με "E", 3 μεταβλητές  $steps$ ,  $maxr$  και  $r$  τύπου  $int$  (εκ των οποίων η  $steps$  ισούται με 0) και μια  $bool done$  που ισούται με  $false$ . Η  $steps$  μετράει τα βήματα, η  $maxr$  είναι η μέγιστη τιμή που μπορεί να πάρει το  $r$  στον κανόνα του  $A$  έτσι ώστε να τερματίζεται το πρόγραμμα, η  $r$  βάζουμε τις «τυχαίες» τιμές και το  $done$  χρησιμοποιείται για τον τερματισμό της επανάληψης.

Μπαίνουμε σε μία επανάληψη while η οποία τρέχει όσο !done δηλαδή όσο το done παραμένει false (γιατί με το ! παίρνουμε το αντίθετό του, δηλαδή true). Το done το αλλάζουμε σε false όπου ο κανόνας προσθέτει μη τερματικό στοιχείο.

Με την εντολή srand(time(nullptr)) βάζουμε στο seed της random τον τωρινό χρόνο. Το κάνουμε για να είναι πιο «τυχαία» τα αποτελέσματα. Μετά, βάζουμε το done = true και υποθέτουμε ότι η διαδικασία θα τελειώσει στην εκάστοτε επανάληψη. Ύστερα ακολουθούν 4 if.

Στην πρώτη if ελέγχουμε αν υπάρχει το E στο string s. Αν υπάρχει, αντικαθιστούμε το E με το (Υ) (σύμφωνα με τον 1<sup>ο</sup> κανόνα), εμφανίζουμε την αλλαγή, αυξάνουμε το steps κατά 1 και το done = false. Αν το E δεν υπάρχει στο s τότε το find επιστρέφει -1 (αφού το έχουμε βάλει != string::npos).

Στην δεύτερη if ελέγχουμε αν υπάρχει το Υ στο string s. Αν υπάρχει, αντικαθιστούμε το Υ με το AB (σύμφωνα με τον 2<sup>ο</sup> κανόνα), εμφανίζουμε την αλλαγή, αυξάνουμε το steps κατά 1 και το done = false. Αν το Υ δεν υπάρχει στο s τότε το find επιστρέφει -1 (αφού το έχουμε βάλει != string::npos).

Στην τρίτη if ελέγχουμε αν υπάρχει το A στο string s. Αν υπάρχει, τσεκάρουμε με άλλο ένα if πόσα είναι τα steps. Αν είναι μικρότερα ή ίσα του 20 τότε το maxr (η μέγιστη τιμή που μπορεί να πάρει το r στον κανόνα του A έτσι ώστε να τερματίζεται το πρόγραμμα) γίνεται 2, δηλαδή, αν έχει περάσει τα 20 steps δεν αφήνουμε να

εφαρμοστεί η 2<sup>η</sup> περίπτωση του κανόνα, στην οποία δεν θα τερματιζόταν το πρόγραμμα. Αν δεν έχει περάσει τα 20 steps, το maxr γίνεται 1. Μετά, αφού βγούμε από την τελευταία if, με την εντολή `r = rand() % maxr + 1` δίνουμε στο `r` μία «τυχαία» τιμή μεταξύ του 1 και maxr. Έπειτα, με άλλη μια if ελέγχουμε αν το `r = 1`. Αν ναι, τότε, στο string `s` αντικαθιστούμε το `A` με `n`, εμφανίζουμε τις αλλαγές και αυξάνουμε τη `steps` κατά 1. Αλλιώς, στο string `s` αντικαθιστούμε το `A` με `E`, εμφανίζουμε τις αλλαγές και αυξάνουμε τη `steps` κατά 1 και το `done = false`. Ουσιαστικά, κάνουμε την αντικατάσταση του γράμματος `A` να είναι τυχαία. Ύστερα, βγαίνουμε από την τρίτη if.

Στην τέταρτη if ελέγχουμε αν υπάρχει το `B` στο string `s`. Αν υπάρχει, τότε, δίνουμε στο `r` μία τυχαία τιμή από το 1 έως το 3. Με άλλη μια if τσεκάρουμε την τιμή του `r`. Αν `r = 1`, τότε αντικαθιστούμε το `B` με το `-Y`, εμφανίζουμε την αλλαγή, αυξάνουμε το `steps` κατά 1 και το `done = false`. Αν `r = 2`, τότε αντικαθιστούμε το `B` με το `+Y`, εμφανίζουμε την αλλαγή, αυξάνουμε το `steps` κατά 1 και το `done = false`. Αν `r = 3`, τότε αντικαθιστούμε το `B` με το κενό, εμφανίζουμε την αλλαγή, αυξάνουμε το `steps` κατά 1 και το `done = false`.

### **Άσκηση 3**

Δίνεται η γραμματική

$S \rightarrow (X)$

$$X \rightarrow YZ$$

$$Y \rightarrow \alpha \mid \beta \mid S$$

$$Z \rightarrow *X \mid -X \mid +X \mid \varepsilon$$

Για να είναι εφικτή η υλοποίηση του προγράμματος, μας ζητάει να βρούμε αν στην γραμματική είναι LL(1).

Για να ορίσουμε αν η γραμματική είναι LL(1), πρέπει να ορίσουμε τα σύνολα **FIRST** και **FOLLOW** καθώς και τις συναρτήσεις **EMPTY** και **LOOKAHEAD**.

Οι κανόνες Y και Z έχουν περισσότερα του ενός δεξιά μέλη και θα μπορούσαν να γραφούν :

$$S \rightarrow (X)$$

$$X \rightarrow YZ$$

$$Y \rightarrow \alpha$$

$$Y \rightarrow \beta$$

$$Y \rightarrow S$$

$$Z \rightarrow *X$$

$$Z \rightarrow -X$$

$$Z \rightarrow +X$$

$$Z \rightarrow \varepsilon$$

## 1. Σύνολο FIRST

Για να βρούμε το σύνολο FIRST, βρίσκουμε ποια είναι τα μη τερματικά και τα τερματικά σύμβολα.



Μετά κοιτάμε σε κάθε κανόνα μετά το βελάκι το αριστερότερο τερματικό σύμβολο.

Αν ο κανόνας μετά από το βελάκι ξεκινάει με τερματικό σύμβολο, τότε το  $FIRST(\text{κανόνα}) = \{\text{τα τερματικά σύμβολα}\}$

Αν ο κανόνας μετά από το βελάκι αρχίζει με μη τερματικό σύμβολο, τότε πάμε στον κανόνα που είναι το μη τερματικό σύμβολο και κοιτάμε μετά το βελάκι το αριστερό μέρος του κανόνα αυτού

Μη τερματικά σύμβολα  $N = \{S, X, Y, Z\}$

Τερματικά σύμβολα  $T = \{ (, ), \alpha, \beta, *, +, -, \epsilon \}$

$FIRST(S) = \{ ( \}$

$FIRST(X) = FIRST(Y) = \{ \alpha, \beta, ( \}$

$FIRST(Y) = \{ \alpha, \beta, ( \}$

$FIRST(Z) = \{ *, -, +, \epsilon \}$

## 2. Σύνολο FOLLOW

Υπάρχουν 3 κανόνες που μπορούν να υλοποιηθούν για τον υπολογισμό του FOLLOW, αναλόγως τον κανόνα της γραμματικής, εφαρμόζουμε τους κανόνες του FOLLOW.

Οι 3 κανόνες είναι οι εξής:

i)  $\$ \in FOLLOW(S)$

ii)  $A \rightarrow \alpha X \beta : FOLLOW(X) \supseteq FIRST(\beta) - \{\epsilon\}$

iii) 3. 3i)  $A \rightarrow \alpha X$  ή 3ii)  $A \rightarrow \alpha X \beta$  με  $\epsilon \in \text{FIRST}(\beta)$  :  
 $\text{FOLLOW}(X) \supseteq \text{FOLLOW}(A)$

### 1<sup>ος</sup> κανόνας FOLLOW

Εφαρμόζεται πάντα στο αρχικό σύμβολο της γραμματικής, δηλαδή το S.

Άρα  $\$ \in \text{FOLLOW}(S) \rightarrow$  σχέση 1.

### 2<sup>ος</sup> κανόνας FOLLOW

Βρίσκουμε που μπορούμε να κάνουμε την αντικατάσταση  $\alpha X \beta$ . Μπορούμε να τον εφαρμόσουμε στους κανόνες γραμματικής:

$X \rightarrow YZ$  όπου  $X = Y$  και  $\beta = Z$

$S \rightarrow (\alpha X \beta)$  όπου  $\alpha = ($  ,  $X = X$  και  $\beta = )$

Άρα

$(\text{FIRST}(Z) - \{\epsilon\}) \in \text{FOLLOW}(Y) \Leftrightarrow \{*, -, +\} \in \text{FOLLOW}(Y) \rightarrow$  σχέση 2.

$(\text{FIRST}(')') - \{\epsilon\}) \in \text{FOLLOW}(X) \Leftrightarrow \{')'\} \in \text{FOLLOW}(X) \rightarrow$  σχέση 3.

### 3<sup>ος</sup> κανόνας του FOLLOW

Ο 3<sup>ος</sup> κανόνας μπορεί να «σπάσει» σε 2 μέρη, το 3i) και το 3ii).

Το 3i) εφαρμόζεται όταν σε μια σχέση της λείπει το β, ενώ στο 3ii) είναι το ίδιο αλλά με την διαφορά ότι στον κανόνα αυτό εμπεριέχετε και το ε (κενό).

### 3i) κανόνας

Μπορούμε να το εφαρμόσουμε στους κανόνες γραμματικής:

$Z \rightarrow *X$  όπου  $\alpha = *$ ,  $X = X$  και  $A = Z$

$X \rightarrow YZ$  όπου  $\alpha = Y$ ,  $X = Z$  και  $A = X$

$Y \rightarrow S$  όπου  $X = S$  και  $A = Y$

Άρα

$\text{FOLLOW}(Z) \supseteq \text{FOLLOW}(X) \rightarrow$  σχέση 4.

$\text{FOLLOW}(X) \supseteq \text{FOLLOW}(Z) \rightarrow$  σχέση 5.

$\text{FOLLOW}(S) \supseteq \text{FOLLOW}(Y) \rightarrow$  σχέση 6.

### 3ii) κανόνας

Μπορεί να εφαρμοστεί στον κανόνα γραμματικής:

$X \rightarrow YZ$  διότι στο  $\text{FIRST}(Z)$  περιέχετε το ε.

Άρα

$\text{FOLLOW}(Y) \supseteq \text{FOLLOW}(X) \rightarrow$  σχέση 7.

## Εύρεση των συνόλων FOLLOW

Κοιτάμε ποια σύνολα είναι ίσα, δηλαδή

$\text{FOLLOW}(\text{κανόνας } 1) \supseteq \text{FOLLOW}(\text{κανόνας } 2)$  και

$\text{FOLLOW}(\text{κανόνας } 2) \supseteq \text{FOLLOW}(\text{κανόνας } 1)$ .

Μετά κοιτάμε ποια σύνολα ποια σύνολα υπάρχουν μόνο σε μία σχέση, δηλαδή αυτά που είναι υπερσύνολα.

Άρα από τις σχέσεις 4 και 5:

**$\text{FOLLOW}(X) = \text{FOLLOW}(Z) \rightarrow$  σχέση 8.**

Τι ξέρουμε για το X; Ότι από την σχέση 3, η  $\{ ) \} \in \text{FOLLOW}(X)$ ,

Άρα από τις σχέσεις 3 και 8:

**$\text{FOLLOW}(X) = \text{FOLLOW}(Z) = \{ ) \} \rightarrow$  σχέση 9.**

Τι ξέρουμε για το Y; Ότι από την σχέση 2 και 7 είναι υπερσύνολο του  $\text{FIRST}(Z) - \{\epsilon\}$  και  $\text{FOLLOW}(S)$  και από την σχέση 9 ότι το  $\text{FOLLOW}(X) = \text{FOLLOW}(Y) = \{ ) \}$ .

Άρα από τις σχέσεις 2, 7 και 9:

**$\text{FOLLOW}(Y) = \{ *, -, +, ) \} \rightarrow$  σχέση 10.**

Εφόσον βρήκαμε σχεδόν όλα τα σύνολα, είναι εύκολο να βρούμε για το τελευταίο.

Άρα από τις σχέσεις 1, 6 και 10:

**$\text{FOLLOW}(S) = \{ *, -, +, ), \$ \} \rightarrow$  σχέση 11.**

### 3. Συνάρτηση EMPTY

Για τον υπολογισμό της συνάρτησης, πρέπει να κοιτάξουμε στους κανόνες της γραμματικής που υπάρχει το  $\epsilon$  (κενό), αν ένας κανόνας περιέχει το κενό,  $EMPTY(\text{κανόνας}) = TRUE$ , αλλιώς  $EMPTY(\text{κανόνας}) = FALSE$ .

Άρα

$EMPTY(S) = FALSE$

$EMPTY(X) = FALSE$

$EMPTY(Y) = FALSE$

$EMPTY(Z) = \mathbf{TRUE}$

### 4. Συνάρτηση LOOKAHEAD

Για την εύρεση της συνάρτησης LOOKAHEAD, υπάρχουν 2 κανόνες που επηρεάζονται από την συνάρτηση EMPTY.

Όταν  $EMPTY(A) \rightarrow X_1X_2...X_n = TRUE$ , τότε  
 $LOOKAHEAD(A) = FIRST(X_1) \cup FIRST(X_2) \cup FIRST(X_n)$   
 $\cup FOLLOW(A)$

Όταν  $EMPTY(A) \rightarrow X_1X_2...X_n = FALSE$ , τότε  
 $LOOKAHEAD(A) = FIRST(X_1) \cup FIRST(X_2) \cup FIRST(X_n)$

#### Εύρεση του LOOKAHEAD

$EMPTY(S) = FALSE$ , άρα

$LOOKAHEAD(S \rightarrow (X)) = FIRST('(') = \{ ( \}$

EMPTY(X)=FALSE, άρα

LOOKAHEAD(  $X \rightarrow YZ$  ) = FIRST( Y ) = {  $\alpha$ ,  $\beta$ , ( }

EMPTY(Y)=FALSE, άρα

LOOKAHEAD(  $Y \rightarrow \alpha$  ) = FIRST(  $\alpha$  ) = {  $\alpha$  }

EMPTY(Y)=FALSE, άρα

LOOKAHEAD(  $Y \rightarrow \beta$  ) = FIRST(  $\beta$  ) = {  $\beta$  }

EMPTY(Y)=FALSE, άρα

LOOKAHEAD(  $Y \rightarrow S$  ) = FIRST( S ) = { ( }

EMPTY(Z)=FALSE, άρα

LOOKAHEAD(  $Z \rightarrow *X$  ) = FIRST( \* ) = { \* }

EMPTY(Z)=FALSE, άρα

LOOKAHEAD(  $Z \rightarrow -X$  ) = FIRST( - ) = { - }

EMPTY(Z)=FALSE, άρα

LOOKAHEAD(  $Z \rightarrow +X$  ) = FIRST( + ) = { + }

EMPTY(Z)=**TRUE**, άρα

LOOKAHEAD(  $Z \rightarrow \epsilon$  ) = FIRST( $\epsilon$ )  $\cup$  FOLLOW(Z)={ $\epsilon$ ,)}

### Εξέταση Γραμματικής LL(1)

Εφόσον έχουμε βρει όλα τα σύνολα για την εξέταση της γραμματικής αν είναι LL(1), αυτό που μας μένει να κάνουμε, είναι να βρούμε από το σύνολο LOOKAHEAD.

Υπάρχουν 2 τρόποι.

#### 1<sup>ος</sup> τρόπος

Ελέγχουμε ποιοι κανόνες της γραμματικής είναι «σπασμένοι», πχ ο κανόνας  $Y \rightarrow \alpha \mid \beta \mid S$  είναι «σπασμένος» σε  $Y \rightarrow \alpha$  ή  $Y \rightarrow \beta$  ή  $Y \rightarrow S$  (το ίδιο και για τον κανόνα Z).

Αν τα σύνολά τους δεν περιέχουν κανένα σύμβολο κοινό, τότε η γραμματική είναι LL(1).

$$\text{LOOKAHEAD}(Y \rightarrow \alpha) \cap \text{LOOKAHEAD}(Y \rightarrow \beta) \cap \text{LOOKAHEAD}(Y \rightarrow S) = \emptyset$$

$$\text{LOOKAHEAD}(Z \rightarrow *X) \cap \text{LOOKAHEAD}(Z \rightarrow -X) \cap \text{LOOKAHEAD}(Z \rightarrow +X) \cap \text{LOOKAHEAD}(Z \rightarrow \varepsilon) = \emptyset$$

Άρα η γραμματική είναι LL(1)

2<sup>ος</sup> τρόπος

Μπορούμε να υλοποιήσουμε ένα συντακτικό πίνακα κοιτώντας στα LOOKAHEAD τα τερματικά σύμβολα της γραμματικής.

	(	)	$\alpha$	$\beta$	*	+	-
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		$Z \rightarrow \varepsilon$			$Z \rightarrow *X$	$Z \rightarrow +X$	$Z \rightarrow -X$

Βλέπουμε ότι ο πίνακας δεν έχει επικαλύψεις, άρα η γραμματική είναι LL(1).

Εφόσον δείξαμε ότι η γραμματική είναι LL(1), μπορούμε πλέον να σχεδιάσουμε τον **Top Down αναλυτή**.

Με την βοήθεια του συντακτικού πίνακα μέσω των κανόνων παραγωγής, μπορούμε να βρούμε την συντακτική ανάλυση για την έκφραση

**$((\beta-\alpha)^*(\alpha+\beta))$ .**

Αυτό που μας μένει να κάνουμε είναι να φτιάξουμε 4 στήλες που η κάθε μια θα περιέχει την **στοίβα**, την **είσοδο**, τα **στοιχεία του πίνακα** και την **παραγωγή κανόνων**.

Η στοίβα αρχικά περιέχει το \$ που είναι από τον 1<sup>ο</sup> κανόνα και αναλόγως την έκφραση που ακολουθεί στην είσοδο, χρησιμοποιούμε και τον ανάλογο κανόνα που μπαίνει στην στοίβα ανάποδα.

Στοίβα	Είσοδος	Στοιχεία Πίνακα	Παραγωγή
\$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-	$-\alpha)^*(\alpha+\beta))\$$		



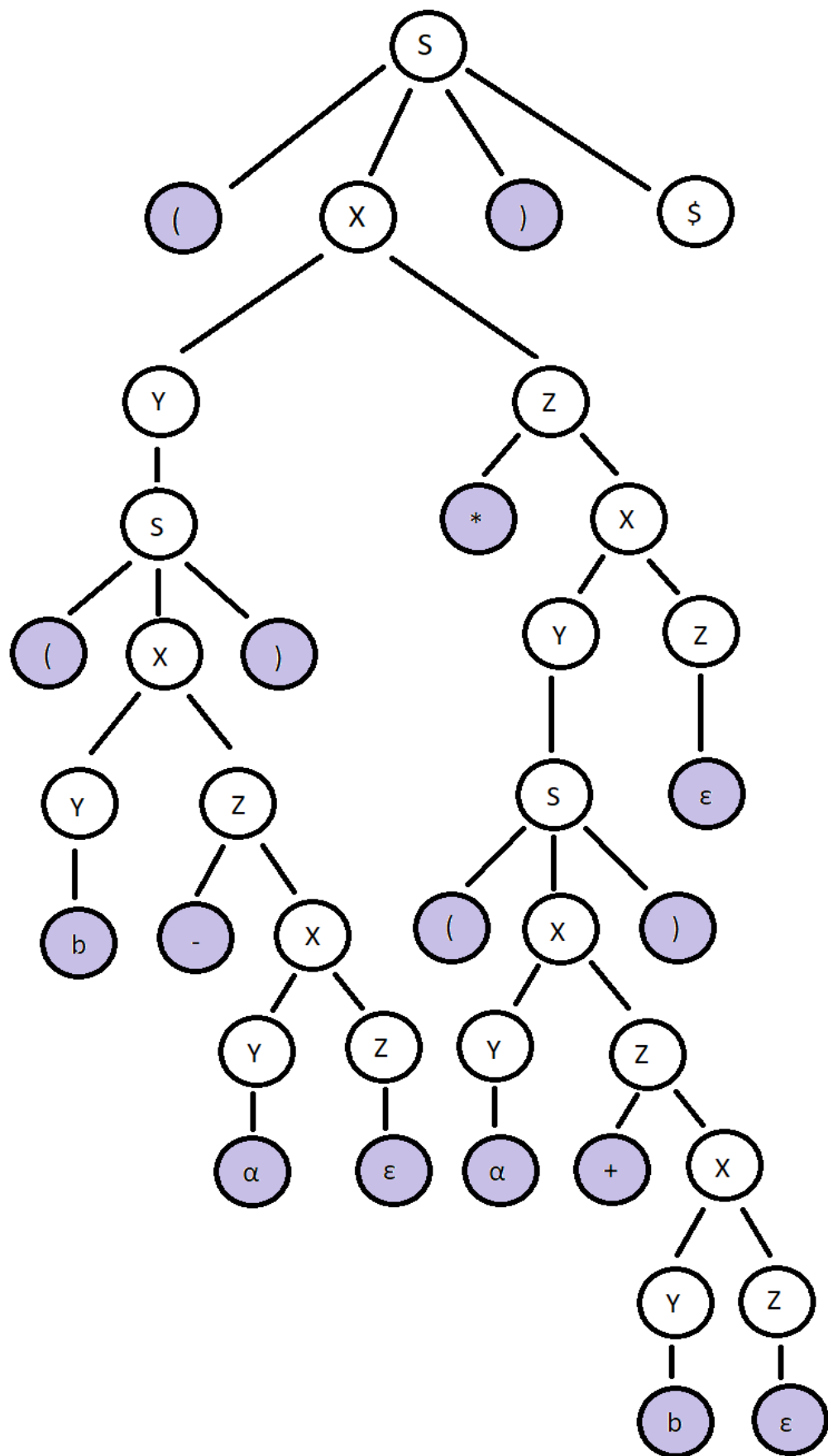
\$)Z)X	$\alpha)^*(\alpha+\beta))\$$	$M(X, \alpha)$	$X \rightarrow YZ$
\$)Z)ZY	$\alpha)^*(\alpha+\beta))\$$	$M(Y, \alpha)$	$Y \rightarrow \alpha$
\$)Z)Z $\alpha$	$\alpha)^*(\alpha+\beta))\$$		
\$)Z)Z	$)^*(\alpha+\beta))\$$	$M(Z, ' ')$	$Z \rightarrow \epsilon$
\$)Z)	$)^*(\alpha+\beta))\$$		
\$)Z	$*(\alpha+\beta))\$$	$M(Z, *)$	$Z \rightarrow *X$
\$)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	$)\$$	$M(Z, ' ')$	$Z \rightarrow \epsilon$
\$)	$)\$$		
\$	$\$$		

Εφόσον η στοίβα και η είσοδος άδειασαν, η έκφραση αναγνωρίζεται από τον συντακτικό αναλυτή.

## Συντακτικό Δέντρο (Θεωρία)

Για την από πάνω προς τα κάτω κατασκευή του συντακτικού δέντρου της έκφρασης  $((\beta-\alpha)*(\alpha+\beta))$ , ακολουθούμε τα εξής βήματα:

- 1) Αρχίζουμε πάντα από τον 1<sup>ο</sup> κανόνα παραγωγής.
- 2) Για κάθε κανόνα παραγωγής, το χωρίζουμε σε **γονείς**, **παιδιά** και **φύλλα**. Γονέας είναι ο κανόνας παραγωγής (μη τερματικό σύμβολο), παιδί μπορεί να είναι μη τερματικό σύμβολο (δηλαδή ο κανόνας που ακολουθεί μετά) ή να είναι τερματικό σύμβολο, που τα τερματικά σύμβολα είναι τα φύλλα.
- 3) Αυτό που κοιτάμε είναι να βρούμε το πιο αριστερό μη τερματικό σύμβολο, ώστε να συνεχίσουμε την παραγωγή μέχρι να αντικατασταθούν οι κόμβοι με τερματικά σύμβολα.



*\*Οι κόμβοι με το μωβ χρώμα είναι τα τερματικά σύμβολα.*

Στο πρόγραμμα το συντακτικό δέντρο για την συγκεκριμένη έκφραση θα εμφανίζεται :

$$\begin{aligned} S \Rightarrow (X) \Rightarrow (YZ) \Rightarrow (SZ) \Rightarrow ((X)Z) \Rightarrow ((YZ)Z) \Rightarrow ((\beta Z)Z) \Rightarrow \\ ((\beta-X)Z) \Rightarrow ((\beta-YZ)Z) \Rightarrow ((\beta-\alpha Z)Z) \Rightarrow ((\beta-\alpha)Z) \Rightarrow \\ ((\beta-\alpha)*X) \Rightarrow ((\beta-\alpha)*YZ) \Rightarrow ((\beta-\alpha)*SZ) \Rightarrow ((\beta-\alpha)*((X)Z)) \Rightarrow \\ ((\beta-\alpha)*((YZ)Z)) \Rightarrow ((\beta-\alpha)*((\alpha Z)Z)) \Rightarrow ((\beta-\alpha)*((\alpha+X)Z)) \Rightarrow \\ ((\beta-\alpha)*((\alpha+YZ)Z)) \Rightarrow ((\beta-\alpha)*((\alpha+\beta Z)Z)) \Rightarrow ((\beta-\alpha)*((\alpha+\beta)Z)) \Rightarrow \\ ((\beta-\alpha)*(\alpha+\beta)) \end{aligned}$$

Με αυτό τον τρόπο, αρχίζουμε από τον πρώτο κανόνα, και κοιτάμε από αριστερά προς τα δεξιά να βρούμε το πρώτο μη τερματικό σύμβολο, ώστε να το αντικαταστήσουμε με ένα τερματικό σύμβολο και μετά πάμε στο επόμενο μη τερματικό σύμβολο ώστε να κάνουμε την αντικατάσταση με ένα τερματικό σύμβολο κ.ο.κ.

## Υλοποίηση προγράμματος

Γλώσσα Προγραμματισμού: C++

### Επεξήγηση

Ξεκινώντας, δηλώνουμε μια ουρά τύπου string και την ονομάζουμε prodtree και μέσα στην prodtree εισάγουμε το γράμμα S (το 1<sup>ο</sup> στοιχείο της γραμματικής). Δηλώνουμε μια μεταβλητή treetemp τύπου string (που θα μας χρειαστεί αργότερα) και μία στοίβα stanl ίδιου τύπου που θα χρησιμοποιηθεί για ανάλυση. Στη στοίβα βάζουμε το 1<sup>ο</sup> στοιχείο, δηλαδή το \$S. Ύστερα, για να έχουμε τον πίνακα γραμματικής χρησιμοποιούμε map τύπου string με όνομα table και βάζουμε τα στοιχεία του πίνακα. Έχουμε ένα string inp το οποίο παίρνει αυτό που θα εισάγει ο χρήστης και με ένα .append προσθέτουμε το \$S στο τέλος του input του χρήστη. Μετά δηλώνουμε δύο μεταβλητές a και x τύπου char που η χρήση τους θα φανεί αργότερα και μια μεταβλητή int pos = 0 η οποία θα λειτουργήσει σαν μετρητής για το input που έδωσε ο χρήστης πριν. Στη συνέχεια έχουμε τρεις μεταβλητές τύπου string, τις prod, tempS και tempS2. Η prod θα παίρνει τις παραγωγές και, οι tempS και tempS2 θα χρησιμεύσουν σε διάφορες περιπτώσεις που θα δούμε παρακάτω.

Με μια while, η οποία τρέχει όσο το input του χρήστη μείον 1 (μείον 1 διότι εμείς βάλαμε το σύμβολο \$S και

όχι ο χρήστης) είναι διαφορετικό του pos, κάνουμε τα εξής:

Αρχικά, το x παίρνει την τιμή του δεξιότερου χαρακτήρα της κορυφή της στοίβας και το a παίρνει την τιμή του αριστερότερου χαρακτήρα του input του χρήστη.

Με μία if ελέγχουμε αν το x (δηλαδή ο χαρακτήρας της στοίβας) είναι τερματικό σύμβολο.

Αν ναι τότε:

Πάλι με μια if ελέγχουμε αν ισχύει ο 1<sup>ος</sup> ( $x \neq '$' \ \&\& \ x == a$ ) ή ο 2<sup>ος</sup> ( $x == a \ \&\& \ a == '$'$ ) κανόνας.

Αν ισχύει ο 1<sup>ος</sup> κανόνας τότε το pos αυξάνεται κατά ένα (δηλαδή είναι σαν να διαγράφεται ένα στοιχείο από το input του χρήστη) και, με την μεταβλητή tempS σώζουμε το 1<sup>ο</sup> στοιχείο της στοίβας, αφαιρούμε το τελευταίο σύμβολο από το 1<sup>ο</sup> στοιχείο της στοίβας, ξαναπροσθέτουμε το updated στοιχείο στη στοίβα και τέλος, κάνουμε reset τη μεταβλητή tempS.

Αν ισχύει ο 2<sup>ος</sup> κανόνας τότε σημαίνει πως η στοίβα δεν έχει άλλα σύμβολα πέρα από το \$, επομένως εμφανίζεται το κατάλληλο μήνυμα και βγαίνουμε από την επανάληψη με ένα break.

Αν το x δεν είναι τερματικό σύμβολο τότε:

Το tempS παίρνει την τιμή του δεξιότερου χαρακτήρα της κορυφή της στοίβας και την τιμή του αριστερότερου

χαρακτήρα του input του χρήστη. Με ένα try-catch τσεκάρουμε αν ισχύει ο 3<sup>ος</sup> ή ο 4<sup>ος</sup> κανόνας (Δηλαδή αν στον 3<sup>ο</sup> κανόνα βρεθεί exception τότε ισχύει ο 4<sup>ος</sup>).

3<sup>ος</sup> κανόνας:

Στην μεταβλητή prod τσεκάρουμε αν η tempS υπάρχει μέσα στο table και αν τη βρούμε τη σώζουμε στην ίδια μεταβλητή και με ένα reverse την αναποδογυρίζουμε για να είναι όπως πρέπει. Ύστερα στη μεταβλητή tempS2 βάζουμε το τελευταίο στοιχείο της στοίβας και αφαιρούμε το τελευταίο σύμβολο του τελευταίου στοιχείου της στοίβας. Με μια if ελέγχουμε αν η prod είναι διαφορετική του e (του ε, δηλαδή του κενού).

Αν είναι διαφορετική τότε:

Προσθέτουμε στην tempS2 την prod, με το reverse γυρνάμε πάλι την prod όπως ήταν πριν, στη treetemp βάζουμε το πίσω μέρος της ουράς prodtree και, τέλος, αλλάζουμε στο treetemp αυτό που άλλαξε στην παραγωγή και βάζουμε το updated treetemp στην ουρά prodtree.

Αν δεν είναι διαφορετική τότε:

Στη treetemp βάζουμε το πίσω μέρος της ουράς prodtree και, αφαιρούμε από το treetemp αυτό που είχε σαν παραγωγή το κενό και βάζουμε το updated treetemp στην ουρά prodtree.

Αφού τελειώσουμε με την if, βάζουμε στη στοίβα τη μεταβλητή tempS2 και κάνουμε reset τις μεταβλητές treetemp, prod, tempS και tempS2.

Αν για τον οποιοδήποτε λόγο βρεθεί exception τότε πάμε στον 4<sup>ο</sup> κανόνα, όπου εκεί εμφανίζεται το κατάλληλο μήνυμα και με ένα break βγαίνουμε από την επανάληψη.

Όταν βγούμε από τη μεγάλη if (που τσεκάρει αν το x είναι τερματικό ή όχι) εμφανίζουμε στον χρήστη πώς είναι τώρα η κορυφή της στοίβας.

Αν όλα πάνε καλά, παραπάνω διαδικασία θα επαναλαμβάνεται όσο η συνθήκη της while είναι αληθής (δηλαδή όσο υπάρχουν ακόμα στοιχεία από το input του χρήστη για να εξεταστούν).

Τέλος, μόλις βγούμε από τη while μπαίνουμε σε μια 2<sup>η</sup> while η οποία εκτελείται όσο η ουρά prodtree δεν είναι άδεια. Μέσα στη while υπάρχει μια if και:

Όταν τα στοιχεία της ουράς είναι περισσότερα από 1 τότε, εμφανίζουμε το μπροστινό μέρος της ουράς και προσθέτουμε και ένα βέλος (=>), αλλιώς εμφανίζουμε μόνο το μπροστινό μέρος της ουράς.

Όταν βγούμε από την if, αφαιρούμε από την ουρά αυτό που εκτυπώσαμε.

## **Άσκηση 4**

Η άσκηση μας ζητάει να φτιάξουμε ένα πρόγραμμα που θα αναγνωρίζει μια κανονική έκφραση που ορίζεται σε



μια γλώσσα προγραμματισμού αλλά τους κανόνες πρέπει να τους ορίσουμε εμείς με τη χρήση **BNF** και **EBNF**.

**BNF:** Είναι μια μεταγλώσσα που χρησιμοποιείται για τον ορισμό σύνταξης μιας γλώσσας προγραμματισμού.

Κύριο χαρακτηριστικό της είναι ότι χρησιμοποιεί την αναδρομικότητα.

**EBNF:** Είναι παρόμοια με την BNF αλλά με την διαφορά ότι γίνεται η χρήση μετασυμβόλων, δηλαδή συμβόλων που χρησιμοποιούνται στη μεταγλώσσα για να σημαίνουν κάτι συγκεκριμένο. Η EBNF αντικαθιστά την αναδρομικότητα της BNF με την επαναληπτικότητα.

Έχουν δοθεί οι συγκεκριμένες οδηγίες και παραδείγματα για την υλοποίηση των κανόνων:

- (1) Στην αρχή πρέπει να εμφανίζεται το όνομα μιας μεταβλητής.
- (2) Στη συνέχεια, θα πρέπει να ακολουθεί το σύμβολο “=”.
- (3) Μετά ακολουθεί το όνομα μιας μεταβλητής ή ένας αριθμός από 1 έως και 9, ύστερα ένα σύμβολο από τα εξής “+”, “-”, “\*”, “/”, “%” και έπειτα ξανά ακολουθεί το όνομα μιας μεταβλητής ή ένας αριθμός από 1 έως και 9.
- (4) Το (3) μπορεί να επαναληφθεί όσες φορές επιθυμούμε.

(5) Η κανονική έκφραση τελειώνει με τον χαρακτήρα “;”

Παραδείγματα έγκυρων κανονικών εκφράσεων:

$X = a + 3 * b;$

$Y = 5 \% 2;$

$Z = 1 / 2 - a * b;$

### Κανόνες σε BNF μορφή

- Μη τερματικά σύμβολα μπαίνουν σε  $\langle \rangle$ .
- Το  $::=$  αντιστοιχεί στο  $\rightarrow$ .
- Το  $|$  συμβολίζει το διαζευκτικό ή όταν σε έναν κανόνα μπορούν να αντιστοιχούν πάνω από ένα κανόνα.

$\langle \text{έκφραση} \rangle ::= \langle \text{γράμμα} \rangle = \langle \text{εκχώρηση} \rangle ;$

$\langle \text{εκχώρηση} \rangle ::= \langle \text{σύμβολο} \rangle \langle \text{τελεστής} \rangle \langle \text{σύμβολο} \rangle |$   
 $\langle \text{εκχώρηση} \rangle \langle \text{επανάληψη} \rangle$

$\langle \text{σύμβολο} \rangle ::= \langle \text{γράμμα} \rangle | \langle \text{αριθμός} \rangle$

$\langle \text{επανάληψη} \rangle ::= \langle \text{τελεστής} \rangle \langle \text{σύμβολο} \rangle |$   
 $\langle \text{πράξη} \rangle \langle \text{εκχώρηση} \rangle$

$\langle \text{γράμμα} \rangle ::= a | b | c | \dots | x | y | z | A | B | C | \dots | X | Y | Z$

$\langle \text{αριθμός} \rangle ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

$\langle \text{τελεστής} \rangle ::= + \mid - \mid * \mid / \mid \%$

Για να επιβεβαιώσουμε ότι οι κανόνες που φτιάξαμε είναι σωστοί, θα πάρουμε τα παραδείγματα των κανονικών εκφράσεων και θα αποδείξουμε ότι όντως βγαίνουν σωστές εκφράσεις.

**$X = a + 3 * b;$**

$\langle \text{έκφραση} \rangle \Rightarrow \langle \text{γράμμα} \rangle = \langle \text{εκχώρηση} \rangle ; \Rightarrow$

**$X = \langle \text{εκχώρηση} \rangle \langle \text{επανάληψη} \rangle ; \Rightarrow$**

**$X = \langle \text{σύμβολο} \rangle \langle \text{τελεστής} \rangle \langle \text{σύμβολο} \rangle \langle \text{τελεστής} \rangle$**   
 **$\langle \text{σύμβολο} \rangle ; \Rightarrow X = a + 3 * b;$**

**$Y = 5 \% 2;$**

$\langle \text{έκφραση} \rangle \Rightarrow \langle \text{γράμμα} \rangle = \langle \text{εκχώρηση} \rangle ; \Rightarrow$

**$Y = \langle \text{σύμβολο} \rangle \langle \text{τελεστής} \rangle \langle \text{σύμβολο} \rangle ; \Rightarrow Y = 5 \% 2;$**

**$Z = 1 / 2 - a * b;$**

$\langle \text{έκφραση} \rangle \Rightarrow \langle \text{γράμμα} \rangle = \langle \text{εκχώρηση} \rangle ; \Rightarrow$

**$Z = \langle \text{εκχώρηση} \rangle \langle \text{επανάληψη} \rangle ; \Rightarrow$**

**$Z = \langle \text{σύμβολο} \rangle \langle \text{τελεστής} \rangle \langle \text{σύμβολο} \rangle \langle \text{πράξη} \rangle$**   
 **$\langle \text{εκχώρηση} \rangle ; \Rightarrow$**

**$Z = 1 / 2 - \langle \text{σύμβολο} \rangle \langle \text{τελεστής} \rangle \langle \text{σύμβολο} \rangle ; \Rightarrow Z = 1 / 2 - a * b;$**

## Κανόνες σε EBNF μορφή

Προηγουμένως είπαμε ότι η EBNF αντικαθιστά την αναδρομικότητα της BNF με την επαναληπτικότητα, άρα αυτό που πρέπει να κάνουμε είναι να τροποποιήσουμε τους κανόνες BNF ώστε να γίνει η μετατροπή τους σε EBNF.

Τα τερματικά σύμβολα μπαίνουν σε “ ”, οπότε φεύγουν τα < > στα μη τερματικά.

Η ::= μπορούμε να το αντικαταστήσουμε με το = .

Στο τέλος του κάθε συντακτικού κανόνα προσθέτουμε μια τελεία.

έκφραση = γράμμα “=” εκχώρηση “;”.

εκχώρηση = σύμβολο τελεστής σύμβολο  
{τελεστής σύμβολο}.

σύμβολο = γράμμα | αριθμός.

γράμμα = “a” | “b” | “c” | ... | “x” | “y” | “z” | “A” | “B” |  
“C” | ... | “X” | “Y” | “Z”.

αριθμός = “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”.


τελεστής = “+” | “-” | “\*” | “/” | “%”.

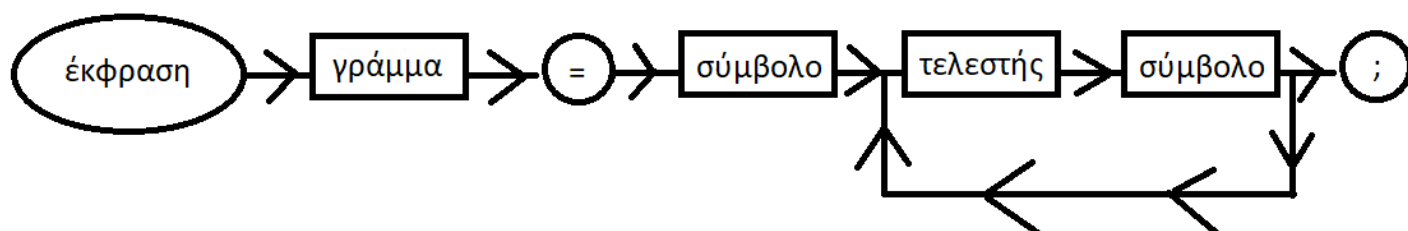
Διώξαμε τον κανόνα της επανάληψης γιατί ο κανόνας αυτός αντικαταστάθηκε με τα { } στον κανόνα εκχώρησης.

Τα { } σημαίνουν επανάληψη από 0 - πολλές φορές.

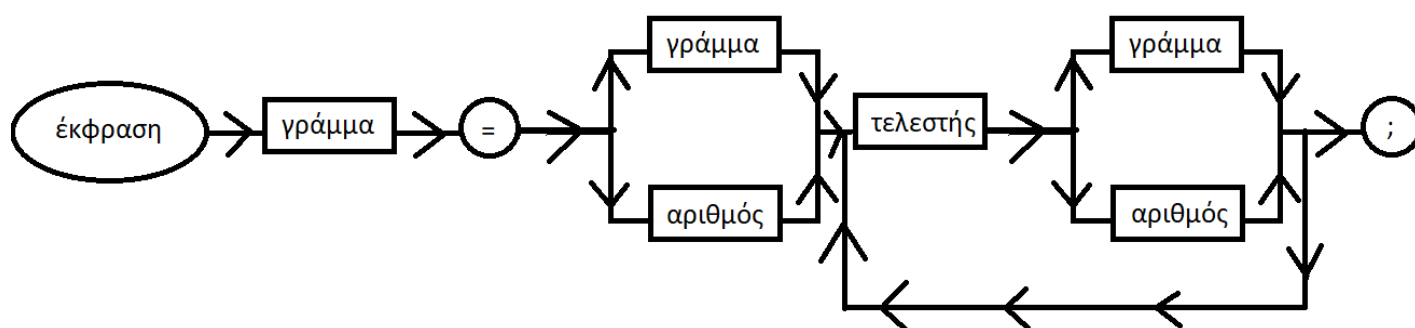
### Συντακτικό Διάγραμμα EBNF

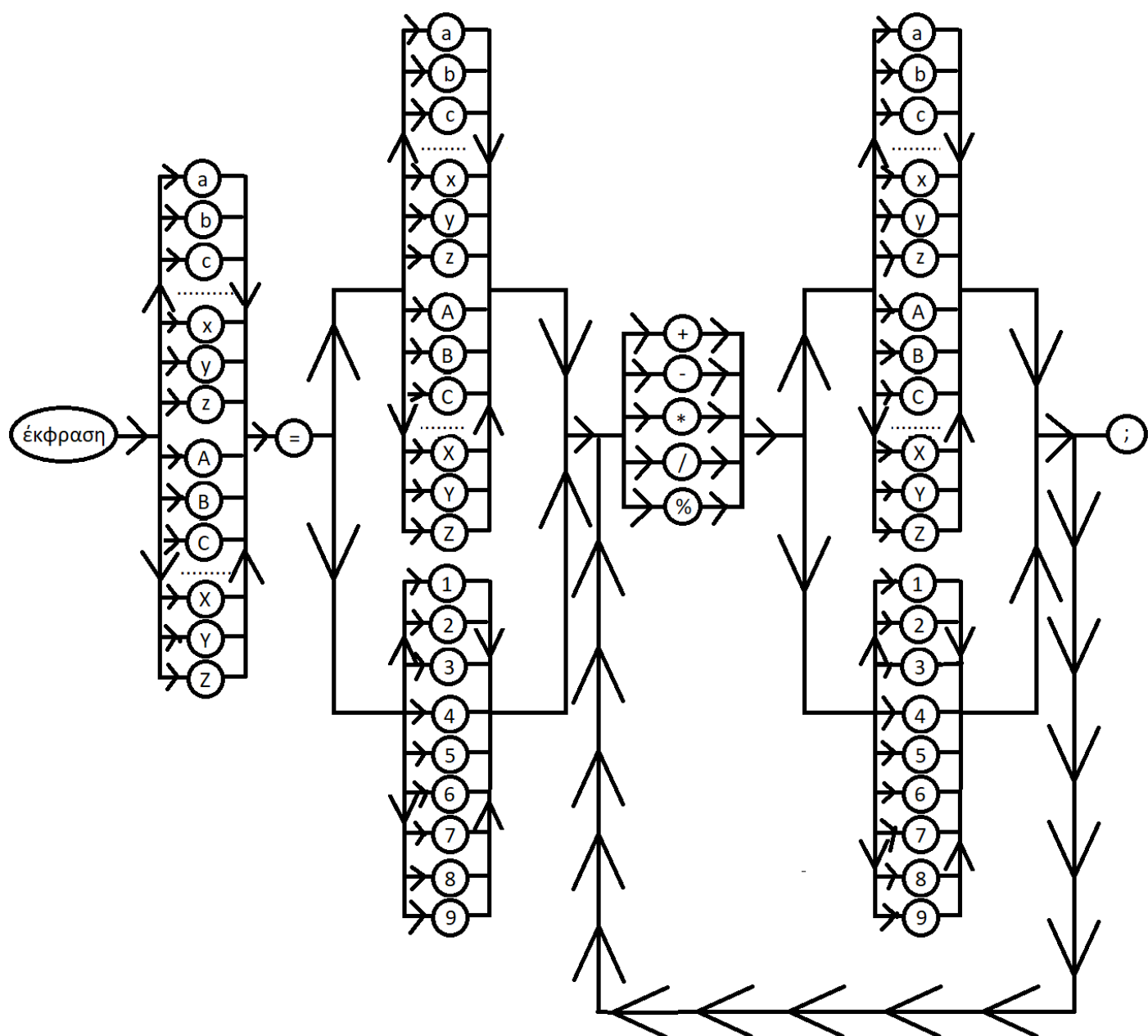
Σε  μπαίνουν τα τερματικά σύμβολα.

Σε  μπαίνουν τα μη τερματικά σύμβολα.



Μπορούμε να αναλύσουμε κι άλλο τα μη τερματικά σύμβολα.





Υλοποίηση προγράμματος

Γλώσσα Προγραμματισμού: Flex (Fast Lexical Analyzer generator)

## Επεξήγηση

Ορίζουμε στο τμήμα δηλώσεων τους συντακτικούς κανόνες έναν-έναν.

Από το flex manual, για να ορίσουμε τους κανόνες γράμμα, αριθμός και τελεστής, θα χρησιμοποιήσουμε τα [ ] (είναι το διαζευκτικό ή)

Και του ορίζουμε έτσι:

- Letters [από το a έως το z ή από το A έως το Z]

Παίρνει όλη την αλφάβητο με τα πεζά και κεφαλαία γράμματα.

- Digits [από το 1 έως το 9]

Παίρνει τους αριθμούς 1,2,3,4,5,6,7,8 ή 9.

- Operators [+\*-/ %]

Παίρνει τους συγκεκριμένους τελεστές.

Πάλι στο τμήμα δηλώσεων, ορίζουμε τους κανόνες Σύμβολο, Εκχώρηση και Έκφραση χρησιμοποιώντας τα {} ώστε να πάρουμε τις ήδη υπάρχουσες εντολές (Letters, Digits και Operators).

Πρώτα ορίζουμε τον κανόνα Symbol που παίρνει τους κανόνες Digits ή Letters :

- $\text{Symbol} \{ \text{Digits} \} | \{ \text{Letters} \}$

Δεύτερο ορίζουμε τον κανόνα Assign που παίρνει τους κανόνες Operators και Symbol:

- $\text{Assign} \{ \text{Operators} \} \{ \text{Symbol} \}$

Τρίτο, και τελευταίο, ορίζουμε τον κανόνα Expression που περιέχει τους κανόνες Letters, μετά βάζουμε τον χαρακτήρα “=” που μετά ακολουθούν οι κανόνες Symbol και Assign. Στο τέλος του κανόνα Expression, βάζουμε το + που σημαίνει 1 έως άπειρες φορές, δηλαδή τον κανόνα Expression μπορούμε να τον επαναλάβουμε από 1 έως άπειρες φορές. Στο τέλος μπαίνει ο χαρακτήρας “;”:

- $\text{Expression} \{ \text{Letters} \} “=” \{ \text{Symbol} \} \{ \text{Assign} \} + “;”$

Τώρα πάμε στο τμήμα κανόνων που εκεί κάνουμε τους ελέγχους.

Παίρνουμε τον κανόνα Expression περικλείνοντας τον με τα { } (δηλαδή παίρνουμε όλο τον κανόνα χωρίς να τον γράψουμε πάλι αναλυτικά) και ανοίγουμε καινούργια { } ώστε, όταν τρέξουμε το πρόγραμμα, και ο χρήστης βάλει input που ακολουθεί τους συντακτικούς κανόνες, να εμφανίζει μήνυμα ότι η έκφραση που έδωσε αναγνωρίζεται από τον συντακτικό αναλυτή.



Έξω από τα { } βάλουμε μία κανονική έκφραση `[^\\n]+` που σημαίνει “οποιοσδήποτε χαρακτήρας εκτός της εναλλαγής γραμμής από μία έως άπειρες φορές”. Αυτό γιατί, όταν ξεκινήσει η εκτέλεση του προγράμματος, δώσουμε ένα input και πατήσουμε enter, αναγκαστικά θα γίνει εναλλαγή γραμμής. Επομένως, με αυτή την έκφραση, το πρόγραμμα θα δει ένα-ένα τους χαρακτήρες που έχουμε δώσει, εκτός της καινούργιας γραμμής.

Αν το input δεν αντιστοιχεί με τους συντακτικούς κανόνες θα εμφανίζει το αντίστοιχο μήνυμα.

Το πρόγραμμα ξεκινά στο τμήμα μεταεπεξεργασίας όπου μέσα στην συνάρτηση `main` καλούμε την συνάρτηση `yylex()`;

Η συνάρτηση αυτή περιέχει τον λεκτικό αναλυτή και το input που επιστρέφει.

## Άσκηση 5

### Υλοποίηση προγράμματος

Γλώσσα Προγραμματισμού: Flex (Fast Lexical Analyzer generator) και C

## Επεξήγηση

Ανοίγουμε `%{ %}` και στο εσωτερικό αφού κάνουμε `#include <stdio.h>` για τη C, δηλώνουμε τις μεταβλητές. Χρησιμοποιούμε τις μεταβλητές `flag`, `i`, `j` και `ret` τύπου `int` που είναι ίσες με το 0 και έναν `pointer *s` τύπου `char` που είναι ίσος με `" "`. Η `flag` μετράει σημεία (A, B, C, κ.ο.κ.), τα `i` και `j` είναι μετρητές, η `ret` παίρνει αυτό που επιστρέφει η συνάρτηση `Shape(int x)` (θα το δούμε στη συνέχεια), και ο `pointer *s` θα μας χρησιμεύσει παρακάτω σε συγκρίσεις για να ελέγξουμε αν υπάρχουν κενά στο `input` του χρήστη. Επίσης, στη συνάρτηση χρησιμοποιείται και μια μεταβλητή `int x` της οποίας η τιμή αλλάζει σε κάθε κανόνα καθώς, προσδιορίζει τον αριθμό των σημείων των σχημάτων.

Ύστερα, περνάμε στο Τμήμα Δηλώσεων.

Αρχικά έχουμε τον κανόνα `Alphabet [A-H]`, δηλαδή τα κεφαλαία γράμματα από A έως H.

Μετά, έχουμε τους κανόνες `Point`, `Straight_Line`, `Triangle`, `Square`, `Pentagon`, `Hexagon`, `Heptagon` και `Octagon` που έχουν τον ίδιο τρόπο δόμησης. Ας πάρουμε ως παράδειγμα τον κανόνα `Point` ο οποίος είναι: `"point" ""*{Alphabet}{1}`

Αυτό σημαίνει ότι:

Η 1<sup>η</sup> λέξη στο input του χρήστη θα πρέπει να είναι "point" και, ουσιαστικά, προσδιορίζει αυτό που θέλει να φτιάξει ο χρήστης. Παρακάτω μπορούν να ακολουθούν από 1 έως άπειρα κενά (" "+: το συν σημαίνει από 1 έως άπειρο και, εφόσον είναι μετά από το κενό, σημαίνει 1 έως άπειρα κενά). Και, τέλος, τα κενά πρέπει να τα ακολουθεί **MONO 1** στοιχείο του κανόνα Alphabet αφού, δίπλα από το { Alphabet } έχουμε βάλει {1}. Αυτό σημαίνει ότι το point θα είναι ένα σημείο (όπως λέει και η ονομασία του).

Το ίδιο ισχύει και για τους υπόλοιπους κανόνες.

Δηλαδή, με βάση τον κανόνα Triangle ο χρήστης πρώτα θα πρέπει να ονοματίσει το σχήμα (triangle), μετά να εισάγει 1 ή παραπάνω κενά και, τελικά, να βάλει τα γράμματα 3<sup>ων</sup> σημείων (δηλαδή όσα σημεία έχει ένα τρίγωνο) που περιλαμβάνονται στον κανόνα Alphabet.

Ακολουθεί το Τμήμα Κανόνων.

Παίρνουμε έναν κανόνα περικλείοντάς τον με τα { } (δηλαδή παίρνουμε όλο τον κανόνα χωρίς να τον γράψουμε πάλι αναλυτικά) και ανοίγουμε καινούργια { } ώστε, όταν τρέξουμε το πρόγραμμα και ο χρήστης βάλει input που ακολουθεί τους συντακτικούς κανόνες, να εκτελούνται τα εξής:

Καλούμε μία συνάρτηση int Shape(int x) δίνοντάς της τον αριθμό των σημείων που πρέπει να έχει το σχήμα.

Μέσα στην συνάρτηση:

Βάζουμε το `flag = 0` έτσι ώστε η μεταβλητή να μην κρατάει προηγούμενες τιμές (σε περίπτωση που η συνάρτηση έχει καλεστεί ξανά). Αντιγράφουμε το `yytext` σε έναν `char pointer *yycopy` για ευκολία, διότι με βάση το `manual`, το `yytext` έχει περιορισμένες δυνατότητες ως προς τις συγκρίσεις. Στη συνθήκη της 1<sup>ης</sup> επανάληψης βάζουμε `i = yyleng - 1` (επειδή αν πχ το `yyleng` είναι από 1-12, το `yycopy` (δηλαδή το `yytext`) το θέλουμε από 0-11 αφού αυτό ξεκινά από 0. Επομένως αν δεν βάζαμε το -1 θα υπήρχε θέμα όταν θα παίρναμε το `yycopy[i]` γιατί θα «βγαίναμε» εκτός του `array`) με την επανάληψη να τρέχει όσο `i >= yyleng - x` (όπου `x` ο αριθμός που δώσαμε στην `Shape(int x)` παραπάνω) και `--i`.

Ουσιαστικά, τρέχουμε το `input` του χρήστη από πίσω προς τα μπρος και περιορίζουμε τον αριθμό των επαναλήψεων με βάση τον αριθμό των σημείων, δηλαδή με άλλα λόγια, ελέγχουμε **μόνο** τα σημεία που έδωσε ο χρήστης και όχι ολόκληρο το `input`.

Το ίδιο κάνουμε και στην εμφωλευμένη επανάληψη με τη μεταβλητή `j`.

Στο εσωτερικό της 2<sup>ης</sup> επανάληψης υπάρχει μία `if` η οποία συγκρίνει το `input` με τον εαυτό του μετρώντας με ένα `flag` πόσες φορές υπάρχουν όμοια γράμματα (χωρίς κενά). Όταν τελειώσουν οι επαναλήψεις ελέγχουμε με μία `if` αν το `flag` (δηλαδή οι φορές που ήταν όμοια τα γράμματα) είναι ίσο με το `x` (τον αριθμό

που δώσαμε στην `Shape(int x)` παραπάνω). Αν ναι τότε η συνάρτηση επιστρέφει το `flag` αλλιώς επιστρέφει 0.

Θα χρησιμοποιήσουμε τον κανόνα `Triangle` ως παράδειγμα:

Υποθέτουμε ότι το `input` του χρήστη ακολουθεί τους συντακτικούς κανόνες, δηλαδή το `input` του είναι πχ: `triangle ABC` επομένως και το `yleng = 12`. Καλούμε τη συνάρτηση `Shape(int x)` και της δίνουμε τον αριθμό 3, δηλαδή τον αριθμό των σημείων που έχουν τα τρίγωνα, άρα  $x = 3$ . Το `flag = 0` και το `input` του χρήστη αντιγράφεται στο `*ycopy`.

Στη συνθήκη της επανάληψης βάζουμε το  $i = \text{yleng} - 1$  για τον λόγο που εξηγήσαμε παραπάνω, άρα ουσιαστικά  $i = 11$ . Η επανάληψη θα τρέχει όσο το  $i \geq \text{yleng} - x$ . Πρακτικά αυτό σημαίνει  $i \geq 9$ , αφού  $12 - 3 = 9$ . Αφού το  $i$  είναι 11, το όριο **μέχρι και** 9 και το βήμα -1, η επανάληψη θα τρέξει 3 φορές (από το 11 έως και το 9), δηλαδή όσα τα σημεία του τριγώνου και είναι προφανές ότι το `input` εξετάζεται από πίσω προς τα μπρος. Το ίδιο ισχύει και για την εμφωλευμένη επανάληψη.

Εφόσον το `input` είναι `triangle ABC`, όταν μπούμε στη 2<sup>η</sup> επανάληψη γίνεται το εξής:

Όπως παρατηρήσαμε παραπάνω, αν θέλαμε να γράψουμε από τα αριστερά προς τα δεξιά πώς φαίνεται το `input` με τον τρόπο που το πήραμε, θα το γράφαμε

έτσι: CBA elgnairt (με τους δείκτες τους να είναι: 11, 10, 9, 8, κ.ο.κ.)

Όμως περιοριζόμαστε μόνο στα σημεία, άρα αυτό που μένει είναι: CBA (με δείκτες: 11, 10, 9)

Άρα όταν φτάσουμε στην if γίνεται το εξής:

Τα  $i, j = 11$ .

Αν `ggcopy[i]` (δηλαδή το C) `== ggcopy[j]` (δηλαδή C) `&& ggcopy[i] != *s` (και C διάφορο του κενού) (το `ggcopy` είναι pointer οπότε έπρεπε να συγκριθεί με pointer που είναι ίσος με το κενό και όχι να συγκριθεί κατευθείαν με το κενό γιατί το κενό είναι char οπότε δεν γίνεται σύγκριση), τότε αύξησε το flag κατά 1. Όντως το flag αυξάνεται κατά 1 γιατί τη συνθήκη είναι αληθής.

Μετά το  $i$  παραμένει 11 και το  $j$  γίνεται 10.

Άρα αν `ggcopy[i]` (δηλαδή το C) `== ggcopy[j]` (δηλαδή B) `&& ggcopy[i] != *s` (και C διάφορο του κενού), τότε αύξησε το flag κατά 1. Δεν αυξάνεται γιατί η συνθήκη είναι ψευδής.

Το ίδιο γίνεται και όταν το `ggcopy[j]` δείχνει στο A.

Ύστερα το  $i$  γίνεται 10, δηλαδή το `ggcopy[i]` δείχνει στο B και γίνεται η ίδια διαδικασία.

Αυτό συνεχίζεται μέχρι να συγκριθούν όλα τα γράμματα μεταξύ τους.

Όταν τελειώσει αυτό, καταλαβαίνουμε ότι θα καταλήξουμε στο ότι `flag = 3` αφού τα γράμματα ABC είναι διαφορετικά μεταξύ τους (άρα και αδύνατο να είναι ίδια με τον εαυτό τους πάνω από 3 φορές) και, προφανώς, βγαίνουμε από την επανάληψη.

Μετά πάμε σε μία `if` και ελέγχουμε αν οι φορές που ήταν ίδια τα γράμματα (`flag`) είναι ίσες με τον αριθμό των σημείων που πρέπει να έχει το σχήμα (`x`). Στη συγκεκριμένη περίπτωση αυτό ισχύει άρα η συνάρτηση θα επιστρέψει 3.

Μέσα στα `brackets` του κανόνα, όταν η `Shape(int x)` επιστρέψει την τιμή του `flag`, τη μεταβιβάζει στη μεταβλητή `ret`. Αν η `ret` ισούται με 3 (ο αριθμός των σημείων που πρέπει να έχει ένα τρίγωνο) τότε εμφανίζεται το κατάλληλο μήνυμα. Αν αυτό δεν ήταν αληθές τότε θα εμφανιζόταν διαφορετικό μήνυμα.

Γενικά, ο τρόπος αυτός βοηθά στο να βρίσκουμε αν ο χρήστης έχει δώσει έγκυρα ονόματα σημείων. Πχ το τρίγωνο ABB δεν είναι έγκυρο και το `flag` θα ισούται με 5 αντί 3. Ή το τρίγωνο AB C (AB κενό C) πάλι δεν είναι έγκυρο αφού το κενό μετρίεται ως χαρακτήρας. Άρα με βάση τον κώδικα, το `input` που θα εξεταστεί θα είναι το B C (B κενό C) και συνεπώς το `flag` θα ισούται με 2 αντί 3.

Η συνάρτηση `Shape(int x)` καλείται από όλους τους κανόνες (εκτός του κανόνα `point`) και με βάση αυτούς, της δίνεται ο κατάλληλος αριθμός σημείων, ενώ παράλληλα, και η μεταβλητή `ret` συγκρίνεται με τον ίδιο κατάλληλο αριθμό για να εμφανιστούν τα κατάλληλα μηνύματα.

Έξω από τα `{ }` όλων των κανόνων βάλαμε μία κανονική έκφραση `[^\\n]+` που σημαίνει “οποιοσδήποτε χαρακτήρας εκτός της εναλλαγής γραμμής από μία έως άπειρες φορές”. Αυτό γιατί, όταν ξεκινήσει η εκτέλεση του προγράμματος, δώσουμε ένα `input` και πατήσουμε `enter`, αναγκαστικά θα γίνει εναλλαγή γραμμής. Επομένως, με αυτή την έκφραση, το πρόγραμμα θα δει ένα-ένα τους χαρακτήρες που έχουμε δώσει, εκτός της καινούργιας γραμμής.

Αν το `input` δεν αντιστοιχεί με τους συντακτικούς κανόνες θα εμφανίζει το αντίστοιχο μήνυμα.

Το πρόγραμμα ξεκινά στο τμήμα μεταεπεξεργασίας, όπου μέσα στην συνάρτηση `main` καλούμε την συνάρτηση `yylex()`; . Η συνάρτηση αυτή περιέχει τον λεκτικό αναλυτή και το `input` που επιστρέφει.