ΠΑ.ΠΕΙ Τμήμα Πληροφορικής

Εργασία Μεταγλωττιστών: 2021 – 2022

3ο Εξάμηνο

<u>Ομάδα εργασίας:</u>

Αιμιλιανός Κουρπάς Δανάς Π20100,

Αποστόλης Σιαμπάνης Π20173,

Θοδωρής Κοξάνογλου Π20094

Περιεχόμενα

Θέμα 1ο:

1.1 Εκφώνηση:

Σχεδιάστε και υλοποιήστε μια γεννήτρια συμβολοσειρών για την παρακάτω γραμματική:

<E>::=(<Y>)

<Y>::=<A><B>

<A>::=v|<E>

<B>::=-<Y>|+<Y>|ε

Λάβετε μέριμνα ώστε η διαδικασία να τερματίζεται οπωσδήποτε. Το πρόγραμμά σας θα πρέπει να τυπώνει τα βήματα της παραγωγής.

1.2 Λύση:

1.2.1 Αλγοριθμική Περιγραφή:

Η γραμματική της εκφώνησης είναι σε μορφή **BNF**.

Τα μη-τερματικά σύμβολα της γλώσσας είναι: **E, Y, A, B** και περικλείονται από γωνιακές παρενθέσεις (<, >).

Τα τερματικά σύμβολα της γλώσσας είναι**: v, -, +, ε, (,)**

Έχουμε συνολικά 7 συντακτικούς κανόνες:

1. **<E>::=(<Y>)**
2. **<Y>::=<A><B>**
3. **<A>::=v**
4. **<A>::=<E>**
5. **<B>::=-<Y>**
6. **<B>::=+<Y>**
7. **<B>::=^**

Αρχικά, το πρόγραμμα θα ξεκινάει από το αρχικό σύμβολο **<E>**. Το αρχικό σύμβολο θα είναι τα **δεδομένα εισόδου** του προγράμματος.

Στη συνέχεια το πρόγραμμα θα λαμβάνει αυτή την συμβολοσειρά **"<E>"** και θα την αποθηκεύει σε μια **μεταβλητή.** Επιπλέον θα θέσουμε έναν **pointer** που θα πάρει όλες τις διευθύνσεις των συμβόλων της συμβολοσειράς με φορά από αριστερά προς τα δεξιά.

Σε κάθε σύμβολο που θα σταματάει ο pointer, το πρόγραμμα θα **αναγνωρίζει το είδος** του συμβόλου (τερματικό ή μη-τερματικό).

Αν το σύμβολο είναι τερματικό: θα αυξηθεί η τιμή του pointer κατά ένα ώστε να διαβαστεί το επόμενο σύμβολο της συμβολοσειράς.

Αν το σύμβολο είναι μη-τερματικό: ο αλγόριθμος θα εκτελέσει το σωστό κανόνα από την γραμματική που μας παρατέθηκε. Θα αντικαταστήσει το μη-τερματικό με τα σύμβολα του δεξιού μέλους του συντακτικού κανόνα που επιλέχθηκε. Στο τέλος της αντικατάστασης, ο pointer θα αυξηθεί κατά ένα ώστε να ελεγχθεί το επόμενο δεξιό σύμβολο.

Παρατηρήσαμε ότι η γραμματική έχει αριστερά αναδρομικούς κανόνες. Για να μην αποτύχει ο επαναληπτικός αλγόριθμος, λοιπόν, θα θέσουμε ένα μέγιστο αριθμό χρήσεων στους κανόνες που κάνουν την διαδικασία ατέρμονη.

Ο επαναληπτικός αλγόριθμος θα **τερματιστεί** όταν ο pointer αποκτήσει διεύθυνση μεγαλύτερη από την διεύθυνση του τελευταίου ψηφίου της συμβολοσειράς, καθώς θα έχει μετατρέψει την αρχική συμβολοσειρά σε μια συμβολοσειρά με μόνο τερματικά σύμβολα.

1.2.2 Υλοποίηση Προγράμματος

Γλώσσα Υλοποίησης: C++

Βιβλιοθήκη: <iostream>

Αρχικά δημιουργήσαμε 5 διαφορετικές μεταβλητές και έναν pointer χαρακτήρα:

Μεταβλητές:

1) **characters**, τύπου string, στην οποία θα θέσουμε την αρχική συμβολοσειρά “<E>”.
2) **symbolPointer**, τύπου int, που θα κάνει χρέη pointer για την συμβολοσειρά της characters.
3) **storePointer**, τύπου int, που θα αποθηκεύει την διεύθυνση του τελευταίου μη τερματικού συμβόλου, ώστε να συνεχίσει η παραγωγή από το τελευταίο μη τερματικό σύμβολο που σταμάτησε.
4) **counterY**, τύπου int, που θα αποθηκεύει τον μέγιστο αριθμό χρήσεων του κανόνα “**2.<Y>::=<A><B>**” ώστε να μην γίνει ατέρμονος ο αλγόριθμος του προγράμματος και το πρόγραμμα να αποτύχει.
5) **listOfSymbols**[10], τύπου const char*, στην οποία αποθηκεύουμε όλο το vocabulary της συμβολοσειράς. Θα μας χρησιμεύσει όταν θα δημιουργήσουμε την συμβολοσειρά.

Για να λειτουργήσει σωστά ο αλγόριθμος χρειάστηκε να δημιουργήσουμε και 4 συναρτήσεις.

Συναρτήσεις:

1) **production**, τύπου int, είναι η συνάρτηση, “κορμός”, της παραγωγής. Αποτελείται από μία επανάληψη while, η οποία θα ελέγχει κάθε φορά όλα τα σύμβολα, με την χρήση του **symbolPointer**. Όταν ο pointer δείχνει τερματικό χαρακτήρα,

μεγαλώνει κατά μία μονάδα και ελέγχεται το επόμενο σύμβολο. Όταν ο pointer δείχνει μη-τερματικό χαρακτήρα, ενεργοποιείται **switch-case** expression, όπου ανάλογα με το σύμβολο, καλείται η συνάρτηση **rule** με τα κατάλληλα ορίσματα.

2) **rule**, τύπου int, καλείται μόνο όταν ο αλγόριθμος έχει μη-τερματικό χαρακτήρα, και με την βοήθεια **switch-case** expression, καλείται η τελευταία συνάρτηση **changeSymbols** με τα απαραίτητα ορίσματα.
3) **changeSymbols**, τύπου int, είναι η συνάρτηση που χωρίζει το characters σε δύο μέλη (**leftString** με την βοήθεια της εντολής **.substr()**: το πρώτο μέλος από την αρχή του string μέχρι και πριν το μη-τερματικό σύμβολο του **symbolPointer**, και το **rightString**: το δεύτερο μέλος από το μη-τερματικό χαρακτήρα του μέχρι το τέλος του string **characters**). Στη θέση του μη τερματικού συμβόλου τοποθετούμε τον δεξιό μέλος του συντακτικού κανόνα που επιλέχθηκε και ενώνουμε το characters = **leftString** + δεξιό μέλος κανόνα + **rightString**, ώστε να ξεκινήσει η επόμενη επανάληψη.

Οι συναρτήσεις 1, 2 και 3 επαναλαμβάνονται με αυτή τη σειρά μέχρι ο **symbolPointer** να φτάσει στο τέλος της συμβολοσειράς **characters**, όπου και τερματίζεται ο κώδικας.

Για την αποφυγή δημιουργίας ατέρμονου κώδικα:

Κάθε φορά που εκτελείται η συνάρτηση int main, η μεταβλητή **counterY** λαμβάνει μία διαφορετική τιμή με την βοήθεια της **srand**(time(0)).

Μετράμε κάθε φορά πόσες φορές χρησιμοποιήθηκε ο κανόνας **“2. <Y>::=<A><B>”**, και τις αφαιρούμε από το **counterY**. Όταν ο **counterY** μηδενιστεί στη **changeSymbols**, απενεργοποιούνται οι κανόνες **4, 5 και 6** και το πρόγραμμα αποκτά πεπερασμένο χρόνο λειτουργίας.

Το πρόγραμμα δημιουργήθηκε ώστε να **μην** μπορεί να δημιουργηθεί κάποιο λάθος από τον χρήστη ή από τον αλγόριθμο.

1.3 Το τεχνικό κομμάτι:

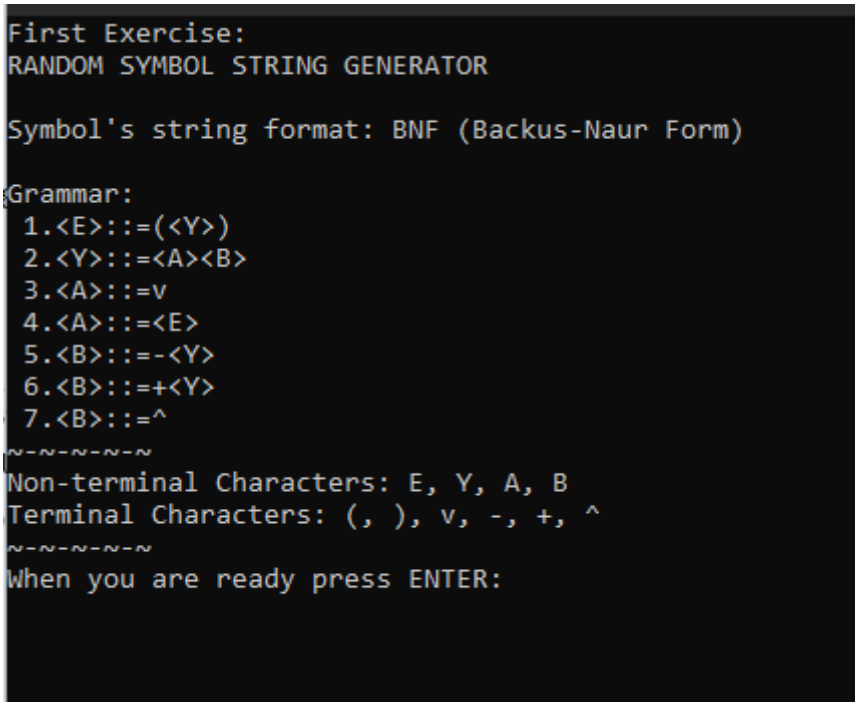
Για να τρέξει το πρόγραμμα αρκεί να κάνουμε compile το αρχείο thema1.cpp από Windows, Linux ή MacOS.

Αν έχετε MacOS ή κάποιο γνωστό Linux distro, υπάρχει μεγάλη πιθανότητα να μην χρειαστεί να κατεβάσετε c++ compiler, καθώς θα το περιλαμβάνει το λειτουργικό.

Οδηγίες για εκτέλεση του προγράμματος σε Windows:

1) Χρειάζεται να έχουμε εγκατεστημένο το MingW.
2) Κατά την εγκατάσταση του προγράμματος επιλέγετε το checkbox για την εγκατάσταση του cpp compiler του MingW, ώστε να λειτουργήσει η εντολή g++ που θα δείτε παρακάτω.
3) Όταν εγκαταστήσετε το MingW, προσθέτετε στο Advanced System Settings → Environment variables → User Variables for user → Path → New και βάζετε το path που βρίσκεται το bin του MingW (default path: C:\MinGW\bin).
4) Κάνετε επανεκκίνηση τον υπολογιστή σας.
5) Ανοίγετε το cmd στο path που βρίσκεται το thema1.cpp, και γράφετε g++ thema1.cpp.
6) Αν έχετε ακολουθήσει όλα τα βήματα και δεν υπήρξε κάποιο error, θα πρέπει κάτω από το ίδιο path του thema1.cpp να έχει δημιουργηθεί το εκτελέσιμο a.exe για συστήματα Windows (και αντίστοιχα θα δημιουργηθεί το εκτελέσιμο τύπου a.out για συστήματα MacOs και Linux).
7) Στο cmd για συστήματα Windows γράφετε: “a.exe”, ώστε να εκτελεστεί το πρόγραμμα (ενώ αντίστοιχα για τα συστήματα MacOs και Linux γράφετε: “./a.out”).

Όταν εκτελέσει ο χρήστης το εκτελέσιμο αρχείο a.exe, θα τον υποδεχθεί η οθόνη:



Σημείωση: για να φαίνεται το σύμβολο του κενού στο terminal, αποφασίσαμε να το ορίσουμε ως “^” αντί για “ε” ή “ ”.

Πατώντας το πλήκτρο enter, ξεκινάει ο αλγόριθμος παραγωγής ο οποίος εκτυπώνει στον terminal τα στάδια στα οποία βρίσκεται μέχρι η συμβολοσειρά να περιλαμβάνει μόνο τερματικά σύμβολα.

Παράδειγμα:

```
The starting symbol is: <E>

The <E> is a non-terminal character. Using the 1. Grammar Rule
=> <E>

The <Y> is a non-terminal character. Using the 2. Grammar Rule
=> (<Y>)

The <A> is a non-terminal character. Using the 3. Grammar Rule
=> (<A><B>)

The <B> is a non-terminal character. Using the 6. Grammar Rule
=> (v<B>)

The <Y> is a non-terminal character. Using the 2. Grammar Rule
=> (v+<Y>)

The <A> is a non-terminal character. Using the 3. Grammar Rule
=> (v+<A><B>)

The <B> is a non-terminal character. Using the 7. Grammar Rule
=> (v+v<B>)

---- Now all the characters are terminals. ----

The final symbol string is: (v+v^)

To close the program, press Enter:
```

Όταν τελειώσει η παραγωγή, για να τερματιστεί το πρόγραμμα ο χρήστης θα πρέπει να πατήσει το πλήκτρο Enter.

Την βιβλιογραφία για το πρώτο θέμα θα το βρείτε στο τέλος του τρίτου θέματος.

Θέμα 2ο:

2.1 Εκφώνηση:

Δίνεται η γραμματική:

S→(X)

X→YZ

Y→α | β | S

Z→*X | -X |+X| ε

Σχεδιάστε και υλοποιήστε συντακτικό αναλυτή top-down που αναγνωρίζει την εκάστοτε συμβολοσειρά ή απαντά αρνητικά ως προς τη συντακτική της ορθότητα. Να επιστρέφεται το σχετικό δέντρο και να εκτυπώνεται. Να γίνει επίδειξη για την έκφραση ((β-α)*(α+β)). Αποδεκτές γλώσσες προγραμματισμού: C/C++

2.2 Λύση:

2.2.1 Αλγοριθμική Περιγραφή

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**.

Έχουμε συνολικά 7 συντακτικούς κανόνες:

1. Y→α

2. Y→β

3. Y→S

4. Z→*X

5. Z→-X

6. Z→+X

7. Z→ε

Αρχικά διακρίνουμε τα **μη-τερματικά** και **τερματικά** σύμβολα της γλώσσας:

Μη-τερματικά: **N = { S, X, Y, Z }**

Τερματικά: **T = { a, b, *, +, -, ε, (,) }**

1) Σύνολα First

Σε κάθε κανόνα παρατηρούμε το αριστερότερο **τερματικό** σύμβολο.

Αλλά έχουμε δύο περιπτώσεις:

Αν το αριστερότερο σύμβολο είναι τερματικό:

τότε: **FIRST(**Κανόνα**)** = **{**Τερματικά Σύμβολα**}**

Αν το αριστερότερο σύμβολο ΔΕΝ ειναι τερματικο σύμβολο:

τότε: Πηγαίνουμε στον κανόνα του συμβόλου αυτού και παίρνουμε το αριστερό του μέρος (Αν πάλι δεν είναι τερματικό το σύμβολο αυτό η διαδικασία συνεχίζεται μέχρις ότου να βρεθεί τουλάχιστον ένα.)

Άρα :

i) **FIRST(S) = { (}**

ii) **FIRST(X) = FIRST(Y) = { α, β, (}**

iii) **FIRST(Y) = { α, β, (}**

iv) **FIRST(Z) = { *, -, +, ε }**

2) Σύνολα Follow

Ο υπολογισμός του συνόλου **Follow** πραγματοποιείται με την βοήθεια τριών **(3)** κανόνων

Οι 3 κανόνες είναι οι εξής:

1) **$ ∈ FOLLOW(S)**

όπου S αρχικό σύμβολο και **$** σύμβολο τέλους εισόδου

2) η παραγωγή A → αBβ τότε **FOLLOW(B) ⊇ FIRST(β) - { ε }**

3) Αν υπάρχει μια απο τις παρακάτω παραγωγές :

i) η παραγωγή A → αB ή

ii) η παραγωγή A → αBβ με ε ∈ **FIRST(β)**

Τότε **FOLLOW(X) ⊇ FOLLOW(A)**

Επομένως, σύμφωνα με τους παραπάνω κανόνες, τις παρακάτω σχέσεις:

Από τον 1ο κανόνα κατασκευής FOLLOW έχουμε :

Κανόνας γραμματικης 1:

Αφού S αρχικό σύμβολο **$ ∈ FOLLOW(S)** ***(1)***

Από τον 2ο κανόνα κατασκευής FOLLOW έχουμε :

Βρίσκουμε που μπορούμε να κάνουμε την αντικατάσταση αΒβ.

Κανόνας γραμματικής 2:

X→εYZ όπου Β = Y και β = Z

Άρα,

FOLLOW(Z) - { ε } ∈ FOLLOW(Y) ⇔ { * , - , + } ∈ FOLLOW(Y) *(2)*

Κανόνας γραμματικής 1:

S→ (X) όπου α = (, Β= X , β =)

Άρα,

FIRST( )) - { ε } ∈ FOLLOW(X) ⇔ {) } ∈ FOLLOW(X) *(3)*

Από τον 3ο κανόνα κατασκευής FOLLOW έχουμε :

Εδώ όπως αναφέραμε υπάρχουν δύο περιπτώσεις:

i) Θα εφαρμοστεί όπου δεν υπάρχει το β:

Κανόνας γραμματικής 6:

Z→*X όπου α = * Β = X και Α = Z

Άρα,

FOLLOW(X) ⊇ FOLLOW(Z) *(4)*

Κανόνας γραμματικής 2:

X→YZ όπου α = Y, Β = Z και Α = X

Άρα,

FOLLOW(Z) ⊇ FOLLOW(X) *(5)*

Κανόνας γραμματικής 5:

Υ→εS όπου Β = S και Α = Υ

Άρα,

FOLLOW(S) ⊇ **FOLLOW(Υ)** ***(6)***

ii) Θα εφαρμοστεί όπου περιέχεται και το **ε** (κενό):

Κανόνας γραμματικής 2 (γιατί στο First**(Z)** περιέχεται το **ε**):

Χ→ΥΖ

Άρα,

FOLLOW(Υ) ⊇ **FOLLOW(Χ)** ***(7)***

(4) , (5) => **FOLLOW(Χ)** = **FOLLOW(Z)** ***(8)***

Οπότε έχουμε:

(2) => **FOLLOW(Χ)** = **{) }**

(8) => **FOLLOW(Z)** = **FOLLOW(Χ)** = **{) }**

(2) , ***(7)*** => **FOLLOW(Υ)** = **{ * , - , + ,) }**

(1) , ***(6)*** => **FOLLOW(S)** = **{ $, * , + ,) }**

3) Συνάρτηση EMPTY

Για τον υπολογισμό της συνάρτησης, πρέπει να κοιτάξουμε στους κανόνες της γραμματικής που υπάρχει το **ε** , αν ένας κανόνας περιέχει το κενό, EMPTY(κανόνας) = TRUE, αλλιώς EMPTY(κανόνας) = FALSE.

Άρα,

EMPTY(S) = FALSE

EMPTY(X) = FALSE

EMPTY(Y) = FALSE

EMPTY(Z) = TRUE

4) Συνάρτηση LOOKAHEAD

Για την εύρεση της συνάρτησης LOOKAHEAD, θα χρειαστούμε την βοήθεια δύο **(2)** κανόνων:

i) Όταν EMPTY(A) X1X2...Xn = TRUE, τότε

LOOKAHEAD(A) = FIRST(X1) U FIRST(X2) U FIRST(Xn) U FOLLOW(A)

ii) Όταν EMPTY(A) X1X2...Xn = FALSE,

τότε LOOKAHEAD(A) = FIRST(X1) U FIRST(X2) U FIRST(Xn)

Άρα, έχουμε:

EMPTY(S) = FALSE, Άρα

LOOKAHEAD(S→ (X)) = FIRST(( ) = { (}

EMPTY(X) = FALSE, Άρα

LOOKAHEAD(X→ YZ) = FIRST(Y) = { α, β, (}

EMPTY(Y) = FALSE, Άρα

LOOKAHEAD(Y→ α) = FIRST(α) = { α }

EMPTY(Y) = FALSE, Άρα

LOOKAHEAD(Y→ β) = FIRST(β) = { β }

EMPTY(Y) = FALSE, Άρα

LOOKAHEAD(Y→ S) = FIRST(S) = { (}

EMPTY(Z) = FALSE, Άρα

LOOKAHEAD(Z→ *X) = FIRST(*) = { * }

EMPTY(Z) = FALSE, Άρα

LOOKAHEAD(Z→ -X) = FIRST(-) = { - }

EMPTY(Z) = FALSE, Άρα

LOOKAHEAD(Z→ +X) = FIRST(+) = { + }

EMPTY(Z) = TRUE, Άρα

LOOKAHEAD(Z→ ε) = FIRST(ε) U FOLLOW(Z) = { ε,) }

Τώρα που ολοκληρώσαμε την ανάλυση των **FIRST**, **FOLLOW**, **EMPTY**, και **LOOKAHEAD** θα μπορέσουμε να απαντήσουμε στο ερώτημα αν η γραμματική είναι **LL(1)**. Για να απαντηθεί αυτό το ερώτημα υπάρχουν δύο τρόποι τους οποίους θα αναλύσουμε.

1ος Τρόπος:

Ελέγχουμε στους κανόνες που έχουν περισσότερα του ενός δεξιά μέλη (Κανόνες 3, 4 και 5 Y→α|β|S και κανόνες 6, 7, 8, 9 Z→*X|-X|+X|ε) και ελέγχουμε αν τα σύνολα τους δεν περιέχουν κανένα κοινό σύμβολο.

Έπειτα αν και μόνο αν ισχύουν τα παρακάτω:

LOOKAHEAD(Y→ α) ∩**LOOKAHEAD**(Y→ β)∩ **LOOKAHEAD**(Y→ S) = Ø**(1)**

LOOKAHEAD(Z→ *X) ∩ **LOOKAHEAD**(Z→ -X) ∩ **LOOKAHEAD**(Z→ +X) ∩ **LOOKAHEAD**(Z→ ε) = Ø**(2)**

Τότε θα είναι η γραμματική μας **LL(1)**

Παρατηρούμε όμως ότι η σχέσεις **(1)** και **(2)** ισχύουν

Άρα η γραμματική είναι **LL(1)**!

2ος Τρόπος:

Θα μάθουμε αν η γραμματική είναι **LL(1)** με την κατασκευή του πίνακα Μ. Ο πίνακας Μ είναι ένας συντατικτός πίνακας όπου το κάθε στοιχείο θα είναι:

1) είτε **ένας κανόνας παραγωγής**

2)είτε **το κενό**

Θα χρειαστούμε την βοήθεια των συνόλων FIRST και FOLLOW:

FIRST(S) = **{** (**}**

FIRST(X) = **FIRST**(Y) = **{** α, β, (**}**

FIRST(Y) = **{** α, β, (**}**

FIRST(Z) = **{** *, -, +, ε **}**

ΚΑΙ

FOLLOW(X) = **{**) **}**

FOLLOW(Z) = **FOLLOW**(X) = **{**) **}**

FOLLOW(Y) = **{** * , -, +,) **}**

FOLLOW(S) = { $, *, +,) }

1.S→(D)

FIRST(S) = {(} Άρα **M(S,( )**= S→ (D)

2.X→YZ

FIRST(Y) = { α, β, (} Άρα **M(X, α)** = X→YZ ,

M(X,b) = X→YZ , **M(X,S)** = X→YZ

3.Y→α

FIRST(α) = {α} Άρα **M(Y,α)** = Y→α

4.Y→β

FIRST(β) = {β} Άρα **M(Y,β)** = Y→β

5.Y→S

FIRST(S) = {(} Άρα **M(Y,( )** = Y→S

6.Z→*X

FIRST(*) = {*} Άρα **M(Z,*)** = Z→*X

7.Z→-X

FIRST(-) = {-} Άρα **M(Z,-)** = Z→-X

8.Z→+X

FIRST(+) = {+} Άρα **M(Z,+)** = Z→+X

9.Z→ε

FIRST(ε) = {ε} => ε ∈ **FIRST(ε)**

Άρα εξετάζουμε το **FOLLOW(Z) = {)}**

Άρα **M(Z,))** = Z→ε

Ο συντακτικός πίνακας, λοιπόν, είναι ο παρακάτω:

	(	**)**	**α**	**β**	*****	**+**	**-**	**$**
S	S-> (X)							
X	X->YZ		X->YZ	X->YZ				
Y	Y->S		Y->α	Y->β				
Z		Z->ε			Z->*X	Z->+X	Z->-X	

Αφού ολοκληρώθηκε η κατασκευή του πίνακα τώρα θα μπορέσουμε να δούμε αν η γραμματική μας είναι **LL(1)**.

Θα είναι λοιπόν αν και μόνο αν στον πίνακα σε κάθε θέση **M(**χ,α**)**

να αντιστοιχεί το πολύ μια παραγωγή.

Παρατηρούμε λοιπόν ότι αυτό ισχύει οπότε η γραμματική μας

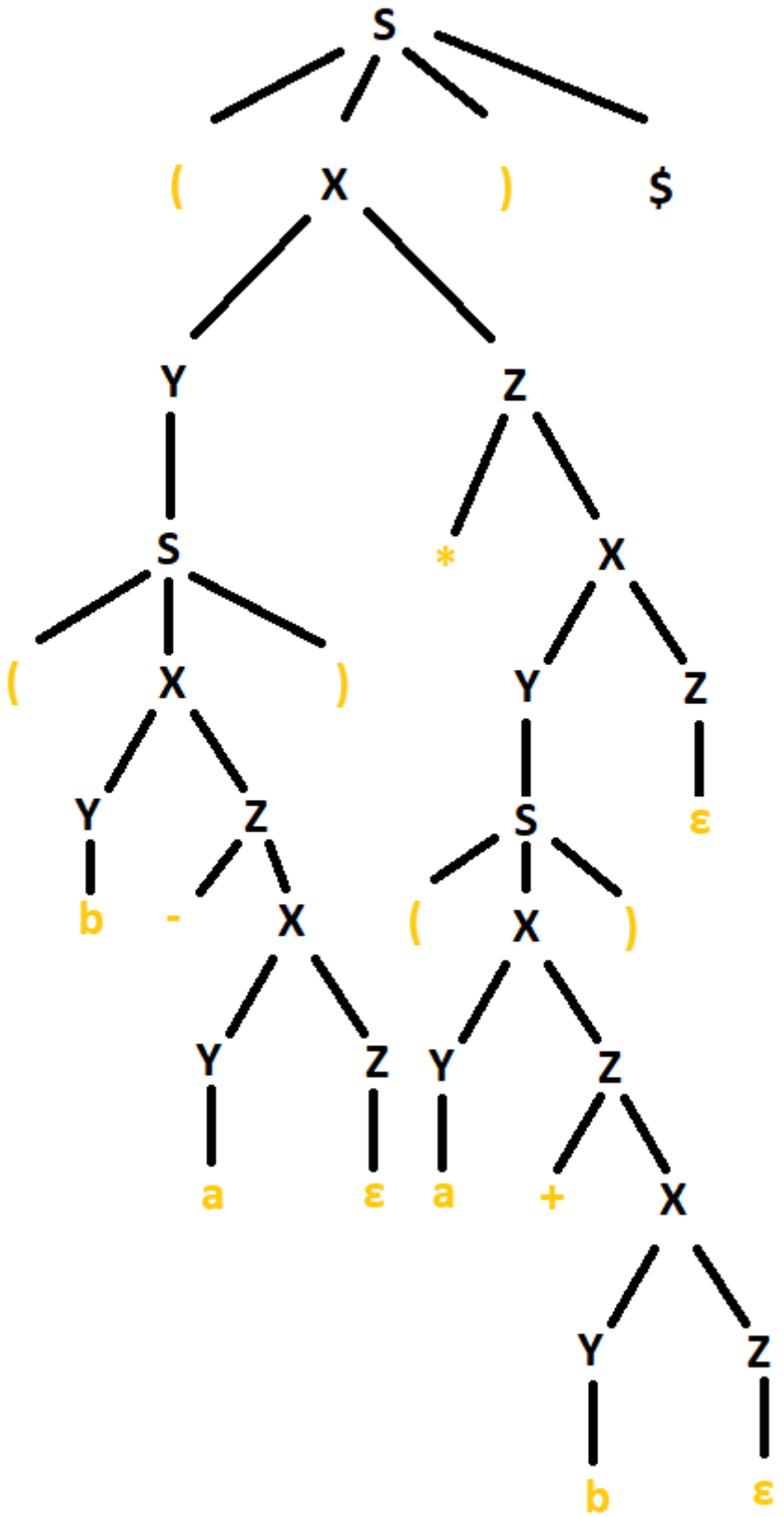
είναι **LL(1)**.

Ελέγχουμε την ενδεικτική συμβολοσειρά « ((b – a) * (a+b)) η οποία αποδεικνύεται ότι συμφωνεί με τους κανόνες παραγωγής και είναι αποδεκτή.

Στοίβα	**Είσοδος**	**Στοιχεία Πίνακα**	**Παραγωγή**
$S	((β-α)*(α+β))$	M(S, ( )	S-> (X)
$)X(	((β-α)*(α+β))$		
$)X	(β-α)*(α+β))$	M(X, ( )	X->YZ
$)ZY	(β-α)*(α+β))$	M(Y, ( )	Y->S
$)ZS	(β-α)*(α+β))$	M(S, ( )	S-> (X)
$)Z)X(	(β-α)*(α+β))$		
$)Z)X	β-α)*(α+β))$	M(X, β)	X->YZ
$)Z)ZY	β-α)*(α+β))$	M(Y, β)	Y->β
$)Z)Zβ	β-α)*(α+β))$		
$)Z)Z	-α)*(α+β))$	M(Z, -)	Z->-X
$)Z)X-	-α)*(α+β))$		
$)Z)X	α)*(α+β))$	M(X, α)	X->YZ
$)Z)ZY	α)*(α+β))$	M(Y, ( )	Y->S
$)Z)Zα	α)*(α+β))$		

$)Z)Z	)*(α+β))$	M(S, ( )	S-> (X)
$)Z)	)*(α+β))$		
$)Z	*(α+β))$	M(Z, *)	Z->*X
$)X*	*(α+β))$		
$)X	(α+β))$	M(X, ( )	X->YZ
$)ZY	(α+β))$	M(Y, ( )	Y->S
$)ZS	(α+β))$	M(S, ( )	S-> (X)
$)Z)X(	(α+β))$		
$)Z)X	α+β))$	M(X, α)	X->YZ
$)Z)ZY	α+β))$	M(Y, α)	Y->α
$)Z)Zα	α+β))$		
$)Z)Z	+β))$	M(Z, +)	Z->+X
$)Z)X+	+β))$		
$)Z)X	β))$	M(X, β)	X->YZ
$)Z)ZY	β))$	M(Y, β)	Y->β
$)Z)Zβ	β))$		
$)Z)Z	))$	M(Z,))	Z->ε
$)Z)	))$		

$)Z	)$	M(Z,))	Z->ε
$)	)$		
$	$		


S
(X) $
Y Z
S * X
(X) Y Z
Y Z S ε
b - X (X)
Y Z Y Z
a ε a + X
Y Z
b ε

2.2.2 Υλοποίηση Προγράμματος

Γλώσσες Υλοποίησης: C++

Βιβλιοθήκες: <iostream>, <list>, <vector>, <string>, <iomanip>

Αρχικά δημιουργήσαμε τις απαραίτητες μεταβλητές:

Μεταβλητές:

1) **listOfSymbols**, τύπου string, έχει αποθηκευμένο όλα τα μη-τερματικά και τερματικά σύμβολα.
2) **symbolString**, τύπου string, είναι η μεταβλητή που θα αποθηκεύσει το input-συμβολοσειρά του χρήστη και η είναι η μεταβλητή που θα επεξεργαστεί το πρόγραμμα στην αναγνώριση της συμβολοσειράς.
3) **stack**, τύπου int, κάνει χρέη stack στον πίνακα της αναγνώρισης.
4) **tableElement, syntaxRule**, τύπου string, εκτυπώνουν τα στοιχεία των στηλών "Στοιχεία Πίνακα" και "Παραγωγής" αντίστοιχα.

Για να λειτουργήσει σωστά ο αλγόριθμος χρειάστηκε να δημιουργήσουμε και 6 συναρτήσεις.

Συναρτήσεις:

1) **stringVocabularyResult** και **validateString**: ελέγχουν το **symbolString** του χρήστη είναι λεξιλογικά σωστό, δηλαδή περιέχει χαρακτήρες που ανήκουν στα τερματικά σύμβολα της γραμματικής.

Αν δεν γίνει αποδεκτή η συμβολοσειρά: το πρόγραμμα ζητάει από τον χρήστη να ξαναπροσπαθήσει εισάγοντας τα σωστά σύμβολα στο terminal.

Αν γίνει αποδεκτή η συμβολοσειρά: από το **validateString**, ενεργοποιείται η **parchingTable**, η οποία ξεκινάει την αναγνώριση της συμβολοσειράς.

2) **parchingTable**: περιλαμβάνει μία επαναληπτικό αλγόριθμο ο οποίος τρέχει είτε μέχρι όλη η **symbolString** να αναγνωριστεί είτε όταν εντοπιστεί σφάλμα στην εύρεση στοιχείου στον συντακτικό πίνακα. Σε κάθε επανάληψη που γίνεται εκτελείται η **findProperSymbols**.
3) **findProperSymbols**: χρησιμοποιεί κάθε φορά το σύμβολο της κορυφής της στοίβας **stack** και το πρώτο σύμβολο της συμβολοσειράς **symbolString**. Με την χρήση δύο if παίρνουμε τις περιπτώσεις:

Αν τα σύμβολα είναι και τα δύο τερματικά: εκτελείται η συνάρτηση removeSymbols που διαγράφει τα σύμβολα αυτά και από τις δύο συναρτήσεις και εκτελείται η επόμενη επανάληψη στο **parchingTable.**

Αν το σύμβολο της στοίβας είναι μη-τερματικό και το σύμβολο της εισόδου είναι τερματικό: εκτελείται η συνάρτηση **findProperRule**.

4) **findProperRule**: βρίσκει το στοιχείο του συντακτικού πίνακα που θα αντικαταστήσει με το μη-τερματικό σύμβολο της στοίβας. Αν δεν βρεθεί κάποιο στοιχείο στον πίνακα, το πρόγραμμα εμφανίζει στον χρήστη το κατάλληλο μήνυμα error και στην συνέχεια τερματίζεται.

5) **printTree**: προσθέτει τις αλλάγες του δένδρου σε μία λίστα tree, η οποία στο τέλος του προγράμματος εκτυπώνεται.

Οι συναρτήσεις 2, 3 και 4 επαναλαμβάνονται με αυτή τη σειρά μέχρι να τελειώσει η αναγνώριση. Όταν τελειώσει η αναγνώριση και γίνει αποδεκτή η συμβολοσειρά, εκτυπώνεται το δέντρο και το κατάλληλο μήνυμα στον terminal και το πρόγραμμα τερματίζεται.

2.3 Το τεχνικό κομμάτι:

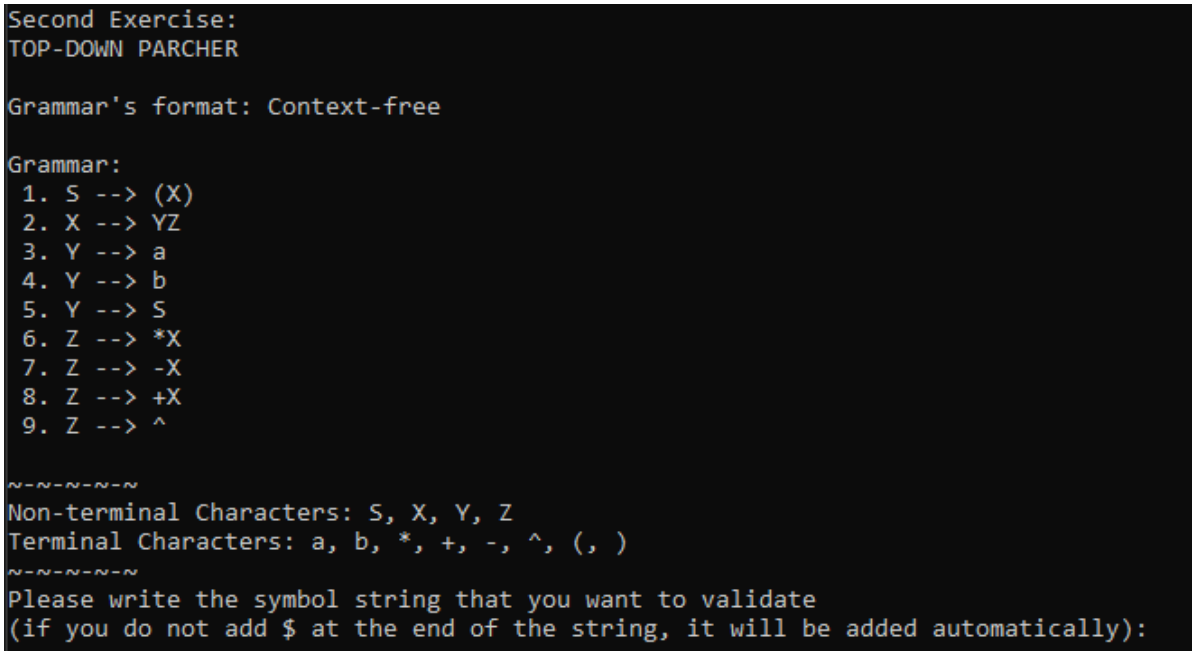
Για να τρέξει το πρόγραμμα αρκεί να κάνουμε compile το αρχείο thema2.cpp από Windows, Linux ή MacOS.

Αν έχετε MacOS ή κάποιο γνωστό Linux distro, υπάρχει μεγάλη πιθανότητα να μην χρειαστεί να κατεβάσετε c++ compiler, καθώς θα το περιλαμβάνει το λειτουργικό.

Οδηγίες για εκτέλεση του προγράμματος σε Windows:

8) Χρειάζεται να έχουμε εγκατεστημένο το MingW.
9) Κατά την εγκατάσταση του προγράμματος επιλέγετε το checkbox για την εγκατάσταση του cpp compiler του MingW, ώστε να λειτουργήσει η εντολή g++ που θα δείτε παρακάτω.
10) Όταν εγκαταστήσετε το MingW, προσθέτετε στο Advanced System Settings → Environment variables → User Variables for user → Path → New και βάζετε το path που βρίσκεται το bin του MingW (default path: C:\MinGW\bin).
11) Κάνετε επανεκκίνηση τον υπολογιστή σας.
12) Ανοίγετε το cmd στο path που βρίσκεται το thema2.cpp, και γράφετε g++ thema2.cpp.
13) Αν έχετε ακολουθήσει όλα τα βήματα και δεν υπήρξε κάποιο error, θα πρέπει κάτω από το ίδιο path του thema2.cpp να έχει δημιουργηθεί το εκτελέσιμο a.exe για συστήματα Windows (και αντίστοιχα θα δημιουργηθεί το εκτελέσιμο τύπου a.out για συστήματα MacOs και Linux).
14) Στο cmd για συστήματα Windows γράφετε: “a.exe”, ώστε να εκτελεστεί το πρόγραμμα (ενώ αντίστοιχα για τα συστήματα MacOs και Linux γράφετε: “./a.out”).

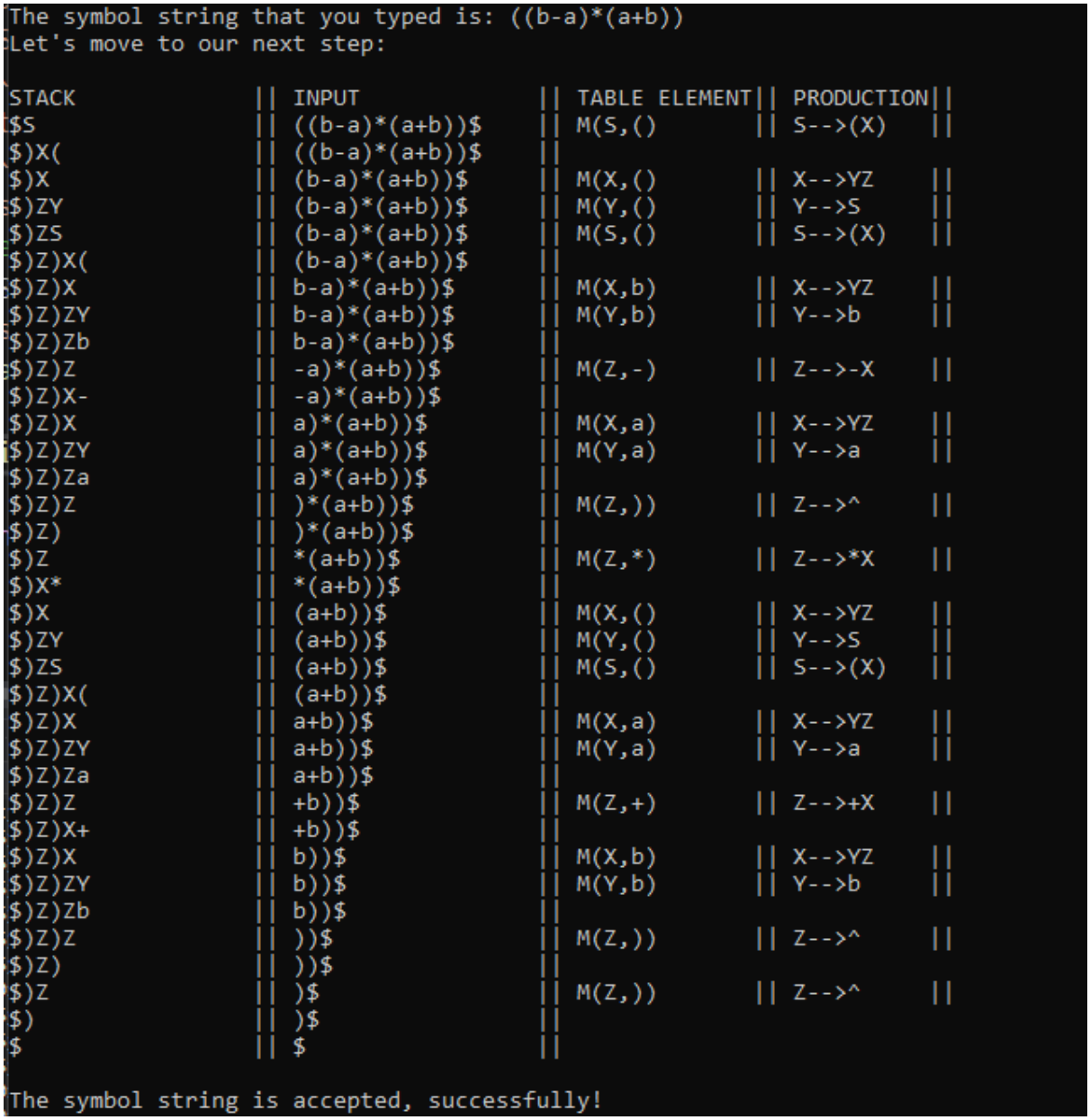
Όταν εκτελέσει ο χρήστης το εκτελέσιμο αρχείο a.exe, θα τον υποδεχθεί η οθόνη:



Σημείωση: για να φαίνεται το σύμβολο του κενού στο terminal, αποφασίσαμε να το ορίσουμε ως “^” αντί για “ε” ή “ ”.

Προτρέπει τον χρήστη να γράψει μία συμβολοσειρά προς αναγνώριση:

Παράδειγμα: Έστω ότι ο χρήστης εισάγει την συμβολοσειρά **((b-a)*(a+b))**



```
The symbol string that you typed is: ((b-a)*(a+b))
Let's move to our next step:

STACK            || INPUT            || TABLE ELEMENT|| PRODUCTION||
$S               || ((b-a)*(a+b))$   || M(S,()       || S-->(X)   ||
$)X(             || ((b-a)*(a+b))$   ||
$)X              || (b-a)*(a+b))$    || M(X,()       || X-->YZ    ||
$)ZY             || (b-a)*(a+b))$    || M(Y,()       || Y-->S     ||
$)ZS             || (b-a)*(a+b))$    || M(S,()       || S-->(X)   ||
$)Z)X(           || (b-a)*(a+b))$    ||
$)Z)X            || b-a)*(a+b))$     || M(X,b)       || X-->YZ    ||
$)Z)ZY           || b-a)*(a+b))$     || M(Y,b)       || Y-->b     ||
$)Z)Zb           || b-a)*(a+b))$     ||
$)Z)Z            || -a)*(a+b))$      || M(Z,-)       || Z-->-X    ||
$)Z)X-           || -a)*(a+b))$      ||
$)Z)X            || a)*(a+b))$       || M(X,a)       || X-->YZ    ||
$)Z)ZY           || a)*(a+b))$       || M(Y,a)       || Y-->a     ||
$)Z)Za           || a)*(a+b))$       ||
$)Z)Z            || )*(a+b))$        || M(Z,))       || Z-->^     ||
$)Z)             || )*(a+b))$        ||
$)Z              || *(a+b))$         || M(Z,*)       || Z-->*X    ||
$)X*             || *(a+b))$         ||
$)X              || (a+b))$          || M(X,()       || X-->YZ    ||
$)ZY             || (a+b))$          || M(Y,()       || Y-->S     ||
$)ZS             || (a+b))$          || M(S,()       || S-->(X)   ||
$)Z)X(           || (a+b))$          ||
$)Z)X            || a+b))$           || M(X,a)       || X-->YZ    ||
$)Z)ZY           || a+b))$           || M(Y,a)       || Y-->a     ||
$)Z)Za           || a+b))$           ||
$)Z)Z            || +b))$            || M(Z,+)       || Z-->+X    ||
$)Z)X+           || +b))$            ||
$)Z)X            || b))$             || M(X,b)       || X-->YZ    ||
$)Z)ZY           || b))$             || M(Y,b)       || Y-->b     ||
$)Z)Zb           || b))$             ||
$)Z)Z            || ))$              || M(Z,))       || Z-->^     ||
$)Z)             || ))$              ||
$)Z              || )$               || M(Z,))       || Z-->^     ||
$)               || )$               ||
$                || $                ||

The symbol string is accepted, successfully!
```

Τον ενημερώνει αν η συμβολοσειρά είναι αποδεκτή ή όχι και εκτυπώνει το συντακτικό πίνακα και το αντίστοιχο δέντρο.

```
The symbol string is accepted, successfully!

Here is the tree:
S
(X)
(YZ)
(aZ)
(a+X)
(a+YZ)
(a+bZ)
(a+b^)
To close the program, press Enter:
```

Για να τερματιστεί το πρόγραμμα ο χρήστης θα πρέπει να πατήσει το πλήκτρο Enter.

Αν ο χρήστης εισάγει την λάθος συμβολοσειρά:

```
The symbol string that you typed is: asd

The symbol string does not have the right vocabulary.
Accepted Symbols: a, b, *, +, -, ^, (, ), $

Please try again:
```

Ή αν γράψει συμβολοσειρά με αποδεκτά σύμβολα άλλα όχι συντακτικά ορθά:

```
(aa)
Let's move to our next step:

STACK    || INPUT    || TABLE ELEMENT|| PRODUCTION||
$S       || (aa)$    || M(S,()       || S-->(X)   ||
$)X(     || (aa)$    ||
$)X      || aa)$     || M(X,a)       || X-->YZ    ||
$)ZY     || aa)$     || M(Y,a)       || Y-->a     ||
$)Za     || aa)$     ||
$)Z      || a)$      ||

The symbol syntax is not correct.
To close the program, press Enter:
```

Την βιβλιογραφία για όλες τις λύσεις των θεμάτων θα τις βρείτε στο τέλος του τρίτου θέματος.

Θέμα 3ο:

3.1 Εκφώνηση:

Σε ένα υποσύνολο φυσικής γλώσσας, τα ονόματα σημείων ορίζονται ως η παράθεση ενός μόνο συμβόλου, τα ονόματα ευθειών ορίζονται ως η παράθεση δύο συμβόλων, τα ονόματα τριγώνων ορίζονται ως η παράθεση τριών συμβόλων, κ.ο.κ, έως και την περίπτωση οκταγώνων. Δεν επιτρέπονται επαναλήψεις συμβόλων. Σχεδιάστε και υλοποιήστε πρόγραμμα Flex που θα αναλύει προτάσεις της μορφής ≪τρίγωνο BCD≫, ≪τετράγωνο BCDA≫, κ.ο.κ. και θα αποδέχεται μόνο τους σωστούς ορισμούς. Παραδείγματα εσφαλμένων ορισμών είναι ≪τετράγωνο AB≫, ≪τρίγωνο AAD≫, ≪γωνία BC≫. Αποδεκτές γλώσσες προγραμματισμού: FLEX (σε συνδυασμό με C/C++).

3.2 Λύση:

3.2.1 Αλγοριθμική Περιγραφή

Θα χρειαστούμε να ορίσουμε τις κανονικές εκφράσεις που θα δέχεται το πρόγραμμα από τον χρήστη. Ο χρήστης θα πρέπει να εισάγει δύο λέξεις. Η **πρώτη λέξη** θα είναι το σχήμα που θέλει να αναγνωρίσει (σημείο, ευθεία, τρίγωνο έως και την περίπτωση του οκταγώνου) και η **δεύτερη λέξη** θα είναι οι κορυφες του σχήματος, οι οποίες θα **πρέπει** να είναι διαφορετικές μεταξύ τους.

Στη συνέχεια, με τη χρήση του λεκτικού αναλυτή, το πρόγραμμα θα αναγνωρίζει την μορφή των λέξεων που έχει εισάγει ο χρήστης.

Στην περίπτωση που η έκφραση του χρήστη είναι λάθος (γραμματικά και συντακτικά): θα εμφανιστεί ένα μήνυμα error που θα προτρέπει τον χρήστη να ξαναγράψει σωστά το σχήμα και τις κορυφές που θέλει να αναγνωριστούν.

Στην περίπτωση που το Input του χρήστη είναι σωστό (γραμματικά και συντακτικά): θα κληθεί μία συνάρτηση η οποία θα ελέγξει την μοναδικότητα των κορυφών.

Η συνάρτηση θα εκτελεί μία επανάληψη η οποία θα ελέγχει μία κορυφή κάθε φορά και θα με την χρήση μιας εμφωλευμένης επανάληψης, θα συγκρίνει την κορυφή της με τις υπόλοιπες κορυφές. Κάθε φορά που εντοπίζεται κοινή κορυφή στο ίδιο σχήμα (μετράει και η περίπτωση όπου η ίδια κορυφή θα ελέγχεται με τον εαυτό της) θα αποθηκεύεται η τιμή της σε έναν μετρητή.

Όταν ελεγχθούν όλες οι κορυφές στο πρόγραμμα θα συγκρίνεται το πλήθος των κορυφών του σχήματος με το πλήθος των κοινών κορυφών του.

Αν το πλήθος τους είναι ίσο: σημαίνει ότι ο χρήστης έθεσε σωστά το σχήμα του, θα ενημερωθεί από το πρόγραμμα και θα τερματιστεί.

Αν δεν είναι ίσο το πλήθος τους: τότε ο χρήστης ενημερώνεται ότι το υπάρχει λάθος στο θέσιμο των κορυφών του και τον προτρέπει να ξαναπροσπαθήσει εισάγοντας μία σωστή κανονική έκφραση.

3.2.2 Υλοποίηση Προγράμματος

Γλώσσες Υλοποίησης: Flex, C

Βιβλιοθήκη: <stdio.h>

Αρχικά γράφουμε την εντολή:

%option noyywrap, που λέει στην Flex να αναγνώσει μόνο ένα αρχείο εισαγωγής τιμών.

Ένα πρόγραμμα Flex χωρίζεται σε τρία μέρη:

Δηλώσεις

%%

Κανόνες Μετάφρασης

%%

Βοηθητικές Διαδικασίες

Δηλώσεις

Μεταξύ των %{ %} προσθέτουμε την βιβλιοθήκη και ορίζουμε 3 διαφορετικές μεταβλητές και έναν χαρακτήρα που παίρνει την τιμή του κενού (" ").

Μεταβλητές:

1) i, τύπου int, που χρησιμοποιείται για συνθήκη επανάληψης for.
2) j, τύπου int, που χρησιμοποιείται για συνθήκη εμφωλευμένης επανάληψης for.
3) peaks_counter, τύπου int, που αποθηκεύει το πλήθος των συμβόλων των διαφορετικών κορυφών.
4) *s, τύπου char, που με την δήλωση του δημιουργεί ένα κυριολεκτικό string και στη δική μας περίπτωση ορίζει το κενό " ".

Μετά το πρώτο τμήμα του κώδικα, περνάμε στο Τμήμα Δηλώσεων, στο οποίο ορίζουμε τον κανόνα **PEAK** [A-Z], που δηλώνει στο λεκτικό αναλυτή ότι γίνονται δεκτοί οι χαρακτήρες από το A έως το Z στο όρισμα **PEAK**.

Κανόνες Μετάφρασης

Στο τμήμα αυτό ορίζουμε τους **9 Κανόνες Μετάφρασης** που θα χρειαστούμε.

Οι πρώτοι 8 ορίζονται ως παράθεση:

Σχήμα + "κενό" + {κορυφές}{μέγιστου πλήθους ανάλογα με το σχήμα}$ {validation(αριθμός);}

Η δομή της παράθεσης είναι:

1) **Σχήμα**: το όνομα σχήματος που ορίζει ο χρήστης.

Διευκρίνιση: Για λόγους ευκολίας στην υλοποίηση της διαδικασίας, πήραμε την παραδοχή ότι το πρόγραμμα θα αναγνωρίζει μόνο τις αγγλικές ορολογίες των σχημάτων.

Το όνομα σχήματος έχει τις επιλογές να οριστεί ως:

- ➔ point
- ➔ line
- ➔ triangle
- ➔ square
- ➔ pentagon
- ➔ hexagon
- ➔ heptagon
- ➔ octagon

2) το **κενό** ως: " "
3) τις **κορυφές**: **{PEAK}{number}**, όπου το **{PEAK}** ορίζει τον κανόνα από το Τμήμα Δηλώσεων και το **{number}** ορίζει το πλήθος των κορυφών που ανταποκρίνεται στο ανάλογο σχήμα.
4) το **$**: ορίζει την κανονική έκφραση που προηγείται, αλλά μόνο στο τέλος μιας γραμμής.

5) {validation**(αριθμός);**}: καλείται η συνάρτηση εγκυρότητας εισαγωγής των κορυφών η οποία θα αναλυθεί στο τμήμα των βοηθητικών διαδικασιών.

Το {validation**(αριθμός);**} χρησιμοποιείται από το σχήμα των ευθειών και μετέπειτα. Εξαίρεση στην παραπάνω παράθεση αποτελεί μόνο ο πρώτος κανόνας μετάφρασης, όπου είναι:

```
point+" "+{PEAK}{1}$ {printf("Accepted!!!\n");}
```

Το **{printf("Accepted!!!\n");}** υπάρχει για την εκτύπωση του μηνύματος, όταν ο χρήστης έχει εισάγει σωστά την κανονική έκφραση. Δεν χρειάζεται να κληθεί η validation συνάρτηση σε αυτή την περίπτωση γιατί δεν υπάρχει σύγκριση κορυφών.

Ο ένατος και τελευταίος κανόνας μετάφρασης ορίζεται ως:

```
.* printf("Please try again with the proper regular expression\n");
```

“Οποιοσδήποτε άλλος χαρακτήρας δοθεί από το χρήστη ως input θα του εκτυπωθεί μήνυμα να δοκιμάσει ξανά την εισαγωγή μιας αποδεκτής από το πρόγραμμα κανονικής έκφρασης.”

Παραδοχή στην συγκεκριμένη περίπτωση του παραπάνω κανόνα είναι το: **.*** . Όπου η τελεία **(.)** εκφράζει τον οποιοδήποτε χαρακτήρα εκτός της νέας γραμμής (new line), ενώ το αστεράκι **(*)** εκφράζει καμία ή περισσότερες εμφανίσεις της κανονικής έκφρασης, που προηγείται (δηλαδή ότι περιλαμβάνει ο χαρακτήρας τελεία).

Βοηθητικές Διαδικασίες

Κάθε φορά που εκτελείται η συνάρτηση int main, εκτυπώνει στο χρήστη ένα μήνυμα για το απαραίτητο input που χρειάζεται το πρόγραμμα για να ξεκινήσει τον αλγόριθμο. Αμέσως μετά, γίνεται η κλήση του συντακτικού αναλυτή **Flex** με την εντολή:

yylex**()**: όπου αποτελεί την κύρια συνάρτηση του λεκτικού αναλυτή που παράγεται βάσει του αρχείου περιγραφής. Επίσης επιστρέφει έναν ακέραιο αριθμό, που συνήθως αντιστοιχεί σε μια λεκτική μονάδα.

Συνάρτηση C:

Validation**:** Η κλήση της συνάρτησης Validation γίνεται από το συντακτικό αναλυτή για την υλοποίηση μιας λειτουργίας που δεν είναι σε θέση ο ίδιος να υλοποιήσει. Συγκεκριμένα η κλήση του γίνεται στο τέλος κάθε κανονικής έκφρασης **($).**

Αναλυτικότερα, με την κλήση της η συνάρτηση χρησιμοποιεί τον μετρητή peaks_counter και αντιγράφει το input του χρήστη με την χρήση της εντολής yytext - η οποία διαβάζει το κείμενο που εισήγαγε ο χρήστης εισάγωντας την στην δήλωση του *yycopy, τύπου char, κάνοντας δεκτό, αυτούσιο, το string που εισήγαγε ο χρήστης στον terminal.

Αμέσως μετά γίνεται η κλήση της πρώτης επανάληψης for της οποίας ο μετρητής είναι το **i** και έχει ως αρχική τιμή το μήκος του input του χρήστη με την χρήση της μεταβλητής yyleng (η οποία περιέχει το πλήθος των χαρακτήρων που περιέχονται στο yytext) ελαττωμένης κατά ένα - για να μην συμπεριληφθεί ο χαρακτήρας **$** - και θα ελαττώνεται κατά ένα την φορά μέχρι να ισχύσει η συνθήκη yyleng **-** **1** **==** yyleng **-** peaks. Στο εσωτερικό της η επανάληψη

περιλαμβάνει την **yycopy** με όρισμα το μετρητή **i** για τη διατήρηση ενός και μόνο χαρακτήρα στην συγκεκριμένη επανάληψη.

Στη συνέχεια περιέχει μια εμφωλευμένη επανάληψη for με μετρητή το **j** και ίδια συνθήκη. Η εμφωλευμένη τώρα επανάληψη με την σειρά της περιέχει έναν έλεγχο τύπου **if** που ως προϋπόθεση έχει την **yycopy** με όρισμα το **i** να ισούται με την **yycopy** με όρισμα το **j** και ταυτόχρονα να ισχύει πως η **yycopy** με όρισμα το **i** να είναι διάφορο με τον ακριβή χαρακτήρα ***s** που όπως προαναφέραμε συμβολίζει το κενό " ". Στην περίπτωση που η συνθήκη οριστεί σωστή ο μετρητής peaks_counter αυξάνεται κατά 1. Με το πέρας όλων των επαναλήψεων δημιουργούμε έναν νέο έλεγχο τύπου **if** που ως προϋπόθεση έχει ο μετρητής peaks_counter να ισούται με την μεταβλητή **peaks** (που δημιουργείται ως όρισμα με την έναρξη της συνάρτησης **validation** και περιέχει το μέγιστο πλήθος κορυφών ανάλογα με το σχήμα) και ταυτόχρονα αυτή (η **peaks** δηλαδή) να είναι διάφορη του 1.

Στην περίπτωση που η συνθήκη οριστεί σωστή: εκτυπώνεται το μήνυμα **{printf("Accepted!!!\n");}** και επιστρέφεται ο μετρητής peaks_counter με την εντολή **return** peaks_counter;.

Ενώ στην αντίθετη περίπτωση όπου δεν θα ισχύουν οι παραπάνω συνθήκες: θα τυπώνεται το μήνυμα **{printf("You have insert one or more times the same letter for the peaks.\n");}** και τερματίζει με το **return 0;**.

Στο τέλος της η main τελειώνει με το **return 0;** το οποίο σημαίνει πως το πρόγραμμα θα εκτελέσει επιτυχημένα αυτό που του έχει ζητηθεί.

3.2.3 Το τεχνικό κομμάτι:

Για να τρέξει το πρόγραμμα αρκεί να κάνουμε compile το αρχείο thema3.l και στη συνέχεια το αρχείο lex.yy.c από Windows, Linux ή MacOS.

Αν έχετε MacOS ή κάποιο γνωστό Linux distro, χρειαστεί να κατεβάσετε flex compiler ακολουθώντας τα παρακάτω βήματα.

Οδηγίες για την εγκατάσταση του Flex compiler για MacOs:

1) Ανοίγετε το terminal και γράφετε:

```
ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install
/master/install)" < /dev/null 2> /dev/null
```

2) Τέλος γράφεται:

```
brew install flex
```

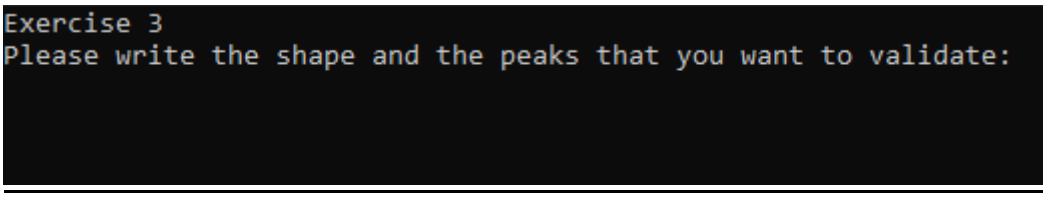
Αν έχετε Windows:

Οδηγίες για εκτέλεση του προγράμματος σε Windows:

1) Χρειάζεται να έχουμε εγκατεστημένο το Flex.
2) Κατά την εγκατάσταση του προγράμματος επιλέγετε το checkbox για την εγκατάσταση του flex compiler του Flex, ώστε να λειτουργήσει η εντολή flex που θα δείτε παρακάτω.
3) Όταν εγκαταστήσετε το Flex, προσθέτετε στο Advanced System Settings → Environment variables → User Variables for user → Path → New και βάζετε το path που βρίσκεται το bin του GnuWin32 (default path: C:\GnuWin32\bin).
4) Επίσης χρειάζεται να έχουμε εγκατεστημένο το MingW.

5) Κατά την εγκατάσταση του προγράμματος επιλέγετε το checkbox για την εγκατάσταση του cpp compiler του MingW, ώστε να λειτουργήσει η εντολή g++ που θα δείτε παρακάτω.
6) Όταν εγκαταστήσετε το MingW, προσθέτετε στο Advanced System Settings → Environment variables → User Variables for user → Path → New και βάζετε το path που βρίσκεται το bin του MingW (default path: C:\MinGW\bin).
7) Κάνετε επανεκκίνηση τον υπολογιστή σας.
8) Ανοίγετε το cmd στο path που βρίσκεται το thema3.l, και γράφετε: flex thema3.l.
9) Στη συνέχεια γράφετε: gcc lex.yy.c
10) Αν έχετε ακολουθήσει όλα τα βήματα και δεν υπήρξε κάποιο error, θα πρέπει κάτω από το ίδιο path του lex.yy.c να έχει δημιουργηθεί το εκτελέσιμο a.exe για συστήματα Windows (και αντίστοιχα θα δημιουργηθεί το εκτελέσιμο τύπου a.out για συστήματα MacOs και Linux).
11) Στο cmd για συστήματα Windows γράφετε: “a.exe”, ώστε να εκτελεστεί το πρόγραμμα (ενώ αντίστοιχα για τα συστήματα MacOs και Linux γράφετε: “./a.out”).

Όταν εκτελεστεί το αρχείο a.exe, θα εμφανιστεί στον terminal:



Παραδείγματα:

- Αν ο χρήστης κάνει λάθος εντελώς την εισαγωγή των δεδομένων, το πρόγραμμα του εμφανίζει μήνυμα για την εισαγωγή μιας άλλης κανονικής έκφρασης.

```
Exercise 3
Please write the shape and the peaks that you want to validate:
points A
Please try again with the proper regural expression
```

- Στη περίπτωση όπου ο χρήστης κάνει εισαγωγή 2 ή περισσοτέρων κορυφών με την την ίδια ονομασία των κορυφών.

```
Exercise 3
Please write the shape and the peaks that you want to validate:
triangle AAD
You have insert one or more times the same letter for the peaks.
```

- Στην περίπτωση που ο χρήστης κάνει σωστή την εισαγωγή και το πρόγραμμα εκτελεστεί κανονικά.

```
Exercise 3
Please write the shape and the peaks that you want to validate:
square ABCD
Accepted!!!
```

Για να τερματιστεί το πρόγραμμα ο χρήστης αρκεί να κλείσει το παράθυρο του terminal.

Βιβλιογραφία Θεμάτων:

Θέμα 1ο:

1)StackoverFlow , Retrieve 12/01/2022 from link:

i)for random:

How do I create a random alpha-numeric string in C++? - Stack Overflow

ii)for Delimiter:

Parse (split) a string in C++ using string delimiter (standard C++) - Stack Overflow

2) Μεταγλωττιστές, Μ. Κ. Βίρβου (Κεφαλαιο 3.2 Σελιδες 92-98)

Θέμα 2ο:

1)Μεταγλωττιστές, Μ. Κ. Βίρβου (Κεφαλαιο 5.2.3 Σελιδες 198-205)

Θέμα 3ο:

1)Μεταγλωττιστές, Μ. Κ. Βίρβου (Κεφαλαιο 4.7 Σελιδες 143-154)

2)Dinosaur , Retrieve 17/01/2022 from link:
http://dinosaur.compilertools.net/flex/manpage.html

3)Flex manual under “ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ” from Gunet2 link:
https://gunet2.cs.unipi.gr/courses/TMB100/index.php

4)Examples (lex1 , lex2 & lex3) from Gunet2 link:
https://gunet2.cs.unipi.gr/modules/document/document.php?course=TMB100&openDir=/20091111162132gzkdd89q/20091111162257862drsj1