Πανεπιστήμιο Πειραιώς
Τμήμα Πληροφορικής
Εργασία Μαθήματος **Μεταγλωττιστές 2023-24**

Η εργασία εκπονήθηκε από τους φοιτητές:

- Γεώργιος Αρβανίτης , Π22015
- Αντώνιος Ευάγγελος Λεμπέσης , Π22086
- Ιάκωβος Πρέκας , Π22147

Πίνακας Περιεχομένων:

Θέμα 1ο

1.1 Περιγραφή του προβλήματος

Καλούμαστε να υλοποιήσουμε ένα Ντετερμινιστικό αυτόματο στοίβας που αναγνωρίζει εκφράσεις αποτελούμενες από τους χαρακτήρες ‘x’ κα ‘y’ κατά τρόπον ώστε:

- 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’.

Στην συνέχεια να τυπώνεται η αλληλουχία βημάτων που οδηγήσαν στην αναγνώριση (ή στην απόρριψη) της έκφρασης.

1.2 Περιγραφή του προγράμματος επίλυσης και Επεξήγηση κώδικα

Ο συγκεκριμένος κώδικας ορίζει μια κλάση με όνομα 'DeterministicStackAutomaton', η οποία υλοποιεί ένα απλό ντετερμινιστικό αυτόματο στοίβας για να αναγνωρίσει ένα συγκεκριμένο πρότυπο σε μια δοσμένη συμβολοσειρά. Το ΝΑΣ έχει δύο καταστάσεις την q0 (αρχική κατάσταση) και την q1 (τελική κατάσταση) , την αλφάβητο ‘x’, ‘y’ και την στοίβα που εμείς την ονομάζουμε 'charStack'.

Η κλάση ορίζεται με ένα δημόσιο τμήμα.

Η συνάρτηση μέλους 'recognizeExpression' παίρνει μια συμβολοσειρά 'expression' ως είσοδο και επιστρέφει ένα boolean που υποδηλώνει εάν η είσοδος αναγνωρίζεται ή όχι

Η 'charStack' είναι μια έκδοση της κλάσης 'stack' που χρησιμοποιείται ως στοίβα για να κρατάει τους ‘x’ χαρακτήρες που δεν έχουν ακόμα συνδεθεί με κάποιο ‘y’

Η συνάρτηση επαναλαμβάνει για κάθε χαρακτήρα στην συμβολοσειρά εισόδου

Η εξεργασία χαρακτήρων γίνεται με 4 τρόπους:

1. Εάν ο τρέχον χαρακτήρας είναι 'x', τότε τον προσθέτει στην στοίβα και παραμένει (ή πηγαίνει) στην κατάσταση q0
2. Εάν ο τρέχον χαρακτήρας είναι 'y' και η στοίβα δεν είναι άδεια, τότε αφαιρεί έναν χαρακτήρα 'x' από αυτήν και ελέγχει αν η στοίβα είναι άδεια, αν είναι πηγαίνει στην κατάσταση q1
3. Εάν ο τρέχον χαρακτήρας είναι 'y' και η στοίβα είναι άδεια, τότε εκτυπώνει ένα μήνυμα σφάλματος και επιστρέφει 'false', υποδηλώνοντας ότι η ακολουθία εισόδου έχει από αριστερά προς τα δεξιά στο σημείο που σταματάει περισσότερα ‘y’ απ’ ότι ‘x’.
4. Σε διαφορετική περίπτωση, εκτυπώνει ένα μήνυμα σφάλματος και επιστρέφει 'false', υποδηλώνοντας ότι η ακολουθία εισόδου είναι μη έγκυρη καθώς βρέθηκε κάποιος μη αναγνωρίσιμος χαρακτήρας.

Αν καταλήξει η κατάσταση στο q1 τότε σημαίνει ότι η έκφρασή μας αναγνωρίστηκε επιτυχώς.

Διαφορετικά, εκτυπώνει ένα μήνυμα σφάλματος καθώς υπάρχουν περισσότερα ‘x’ απ’ ότι ‘y’. και επιστρέφει 'false'.

Τέλος στην main συνάρτηση, δημιουργείται ένα αντικείμενο της κλάσης DeterministicStackAutomaton με το όνομα automaton, και στη συνέχεια ελέγχεται εάν η συμβολοσειρά "xxxyyyxyx" αναγνωρίζεται από τον αυτόματο. Τυπώνονται αντίστοιχα μηνύματα επιτυχίας ή αποτυχίας αναγνώρισης (ή αν δεν δοθεί κάποια έκφραση επιστρέφει το αντίστοιχο μήνυμα).

1.3 Επίδειξη Επίλυσης

Αρχικά για να κάνουμε εκτελέσιμο το αρχείο μας πρέπει ,καθώς είναι εφαρμογή κονσόλα , να έχουμε εγκαταστημένο στην συσκευή μας έναν compiler για C/C++.
Εμείς σε αυτό το παράδειγμα χρησιμοποιούμε τον GCC compiler.

Για την δημιουργία του εκτελέσιμου αρχείου μέσα στον φάκελο που βρίσκεται το αρχείο main.cpp εκτελούμε τις παρακάτω εντολές:

- g++ main.cpp -o ex1.exe (ή ex1.out ανάλογα με το λειτουργικό σύστημα της κάθε συσκευής)

Αυτό θα δημιουργήσει ένα εκτελέσιμο αρχείο ex1.exe (ex1.out)
Με την εκτέλεση του θα έχουμε το παρακάτω αποτέλεσμα:
Σε αυτήν την περίπτωση η φράση μας είναι η ‘xxxyyyxyx’

```
Recognizing expression: xxxyyyxyx
Stack_Size  State  Input
   1         q0  xxyyyxyx
   2         q0  xyyyxyx
   3         q0  yyyxyx
   2         q0  yyxyx
   1         q0  yxyx
   0         q1  xyx
   1         q0  yx
   0         q1  x
   1         q0
Rejected - There are more x's than y's
Recognition Failed
```

Παράδειγμα 1.1 από εκτέλεση της εφαρμογής αλλά απόρριψης της έκφρασης

Εδώ βλέπουμε ότι σωστά απορρίφθηκε η έκφραση μας καθώς δεν έχουμε περισσότερα ‘x’ απ’ ότι ‘y’ με αποτέλεσμα στο τέλος της έκφρασής μας η στοίβα μας δεν είναι ούτε άδεια, ούτε έχει φτάσει στην τελική κατάσταση q1.

Ένα δεύτερο παράδειγμα με την έκφραση ‘xxxxxyyyyyxyxyxyxy’ θα μας δώσει:

```
Recognizing expression: xxxxxyyyyyxyxyxyxy
Stack_Size  State  Input
   1         q0  xxxxyyyyyxyxyxyxy
   2         q0  xxxyyyyyxyxyxyxy
   3         q0  xxyyyyyxyxyxyxy
   4         q0  xyyyyyxyxyxyxy
   5         q0  yyyyyxyxyxyxy
   4         q0  yyyyxyxyxyxy
   3         q0  yyyxyxyxyxy
   2         q0  yyxyxyxyxy
   1         q0  yxyxyxyxy
   0         q1  xyxyxyxy
   1         q0  yxyxyxy
   0         q1  xyxyxy
   1         q0  yxyxy
   0         q1  xyxy
   1         q0  yxy
   0         q1  xy
   1         q0  y
   0         q1
Expression Accepted
Recognition Successful
```

Παράδειγμα 1.2 από εκτέλεση της εφαρμογής και αναγνώριση της έκφρασης

Σε αυτή την περίπτωση η έκφρασή μας αναγνωρίζεται καθώς ακολουθεί όλους τους κανόνες που έχουμε θέση και καταλήγει στο q1 με την στοίβα μας να είναι άδεια.

Άρα τελικά η εφαρμογή μας καλύπτει όλες τις περιπτώσεις που μας ζητήθηκαν.

1.4 Βιβλιογραφικές πηγές

1. Μ. Κ. Βίρβου
   ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ
   Πανεπιστήμιο Πειραιώς
   Εκδόσεις Βαρβαρήγου
   Φρεατύδος 4, 185 37, Πειραιάς
   ISBN: 978-960-7996-15-1

Θέμα 2ο

2.1 Περιγραφή του προβλήματος

Χρησιμοποιώντας την C++ καλούμαστε την παρακάτω γραμματική σε BNF

<Z>::=(<K>)

<K>::=<G><M>

<G>::=v|<Z>

<M>::=-<K>|+<K>|ε, όπου ε η κενή συμβολοσειρά

να την υλοποιήσουμε σε μια γεννήτρια συμβολοσειρών. Δηλαδή κάθε φορά που τρέχουμε το πρόγραμμά μας να έχουμε μια παραγωγή

δηλαδή ξεκινώντας κάθε φορά από το Αρχικό σύμβολο Z να επιλέγεται ένα μη τελικό σύμβολο (μη τελικά Z,K,G,M) τα οποία θα αντικαθίστανται ένα κάθε φορά σύμφωνα με τους κανόνες της γραμματικής μας δίνοντάς μας σε κάθε αντικατάσταση προστασιακούς τύπους μέχρι τελικά να φτάσουμε σε μια πρόταση (όπου πρόταση είναι η συμβολοσειρά που αναγνωρίζεται από την γλώσσα η οποία περιέχει μόνο τερματικά σύμβολα δηλ. (,),v,-,+ και το κενό.

Για ευκολία την γραμματική μας θα την γράψουμε ως κανόνες παραγωγής χωρίς την χρήση BNF δηλαδή:

Z -> (K)

K -> GM

G -> v | Z

M -> -K | +K | ε

Ή αλλιώς χωρίς ενοποίηση:

1. Z -> (K)
2. K -> GM
3. G -> v
4. G -> Z
5. M -> -K
6. M -> +K
7. M -> ε

Τις οποίες τις αριθμώ για να αναφέρομαι σε αυτές

2.2 Περιγραφή του προγράμματος επίλυσης

Η σκέψη που υλοποιήσαμε σε κώδικα είναι η επακόλουθη:

Έχουμε μια string μεταβλητή η οποία κάθε φορά θα περιέχει τον προτασιακό τύπο που έχουμε κάθε φορά για αντικατάσταση. Όμως υπάρχει περίπτωση καθώς θα επιλέγεται τυχαία ο κανόνας παραγωγής που θα εκτελεστεί σύμφωνα με τα σύμβολα που περιέχονται μέσα στον προτασιακό τύπο καθώς μπορεί να επιλέγεται κανόνας ο οποίος να αντικαθιστά το μη τερματικό μας σύμβολο με άλλα μη τερματικά συνέχεια και έτσι ο κώδικάς μας ή δεν θα τελειώσει ποτέ ή θα τελειώσει μετά από πάρα πολύ έχουμε έναν counter ο οποίος θα μετράει πόσες αντικαταστάσεις έχουν γίνει. Άμα οι αντικαταστάσεις δεν έχουν ξεπεράσει το όριο που θέσαμε (έχουμε επιλέξει το όριο να είναι 20 εμείς) τότε θα επιλέγεται κάθε φορά ένας από τους παραπάνω 7 κανόνες σύμφωνα με τα σύμβολα που έχουμε μέσα στον προτασιακό μας τύπο. Αν τώρα ξεπεράσει το όριο των 20 ο counter τότε αυτό που κάναμε είναι ενώ πριν το G το αφήναμε να αντικατασταθεί με v ή Z και το M με -K ή +K ή ε, τώρα το G θα αντικαθίσταται μόνο με v και το M μόνο με ε έτσι ώστε να επιλέγονται αντικαταστάσεις με μόνο τερματικά σύμβολα για να σταματήσει κάποια στιγμή η διαδικασία.

2.3 Επεξήγηση κώδικα

Εξήγηση custom functions που φτιάξαμε:

Έχουμε αρχικά φτιάξει 4 συναρτήσεις (checkForZ, checkForK, checkForG, checkForM) που ελέγχουν άμα υπάρχει τουλάχιστον ένα από τα αντίστοιχα μη τελικα σύμβολα στον προτασιακό τύπο που λαμβάνει ως όρισμα η κάθε μια συνάρτηση.

Έχουμε μετά την countCharacter που λαμβάνει 2 ορίσματα, τον προτασιακό τύπο και τον χαρακτήρα (που είναι ή Z ή K ή G ή M), για να μετρήσει μέσα στον προτασιακό τύπο πόσα από τα αντίστοιχα μη τελικά σύμβολα έχουμε.

Έχουμε την positionOfTheChracteeInMyString η οποία παίρνει 3 ορίσματα, η οποία ψάχνει στον προτασιακό τύπο που λαμβάνει ως όρισμα να βρει την θέση του μη τελικού συμβόλου Z ή K ή G ή M που επίσης λαμβάνει ως όρισμα για να βρει την ι-οστη εμφάνιση του χαρακτήρα αυτού ,που το ι το λαμβάνει επίσης ως όρισμα. Το σκεπτικό είναι ότι άμα είχαμε πχ string myString = "Hello"; cout << myString[k]; με το myString[k] μπορούμε να πάρουμε τον χαρακτήρα στο string μας που βρίσκεται στην k+1 θέση.

Τέλος έχουμε την chooseWhichProsuctionRuleToDo η οποία παίρνει σαν όρισμα τον προτασιακό μας τύπο, χρησιμοποιεί τις checkForZ, checkForK, checkForG, checkForM για να δει στον προτασιακό τύπο που έλαβε σαν όρισμα ποια μη τελικά σύμβολα υπάρχουν έτσι ώστε με την βοήθεια ενός vector άμα υπάρχει πχ Z να προσθέτει στο vector το νούμερο 1 (δηλαδή ότι μπορούμε να εφαρμόσουμε τον κανόνα νούμερο 1), άμα υπάρχει πχ M τότε προσθέτει 5,6,7 (που σημαίνει ότι μπορούμε να εφαρμόσουμε τους κανόνες 5 ή 6 ή 7 κτλ) όπου από αυτό το vector με την συνάντηση rand() θα επιλέξουμε έναν τυχαίο αριθμό από εκεί μέσα που θα είναι ποιος κανόνας θα εφαρμοστεί στον τωρινό προτασιακό τύπο. Αν επιστρέψει 0 σημαίνει ότι δεν έχουμε κάποιον κανόνα να επιλέξουμε και αυτό γιατί δεν υπάρχει στο προτασιακό τύπο κάποιο μη τελικό σύμβολο. Κανονικά αν υπήρχε θα επέστρεφε έναν αριθμό από το 1 μέχρι 7.

Στην main μας ,η οποία είναι και η αφετηρία του προγράμματός μας,
(βλέπουμε: Γραμμή 135)
επειδή όπως αναφέραμε θα χρησιμοποιήσουμε τυχαιότητα ένας τρόπος είναι να χρησιμοποιήσουμε την rand(). Ο τρόπος λειτουργιάς της rand() στηρίζεται σε κάποια περιπλοκά μαθηματικά και κάθε φορά παράγει την ιδία ακολουθία τυχαίων αριθμών.

Για τον λόγο αυτό, λέμε ότι παράγει ψευδοτυχαίους αριθμούς. Προκειμένου κάθε φορά που τρέχουμε το πρόγραμμα να παράγει άλλη ακολουθία τυχαίων αριθμών, πρέπει αρχικά στο πρόγραμμά μας να τρέξουμε την συνάρτηση void srand(int seed) όπου ο πιο συνηθισμένος τρόπος χρήσης είναι στην αρχή του προγράμματός μας να γράψουμε την εντολή: srand(time(NULL)); έτσι ώστε να αρχικοποιηθεί η ακολουθία τυχαίων αριθμών με μία παράμετρο που εξαρτάται από την τρέχουσα ώρα. Θα απαιτηθεί να ενσωματώσουμε και το αρχείο κεφαλίδας time.h στο οποίο έχει οριστεί η συνάρτηση time() που χρησιμοποιεί το πρόγραμμά μας.

Και αυτό στην ουσία που κάνουμε, όσο ο counter δεν έχει ξεπεράσει το όριο των 20 αντικαταστάσεων (γραμμές 152 μέχρι 305), είναι επιλέγουμε όσο υπάρχει μη τελικό σύμβολο έναν κανόνα για να εφαρμόσουμε. Επειδή όμως το σύμβολο που θα αντικαταστήσουμε μπορεί να υπάρχει πολλές φορές πρέπει να μετρήσουμε πόσα από αυτά τα μη τελικά σύμβολα που βρίσκονται στο αριστερό μέρος του κανόνα υπάρχουν στον προτασιακό τύπο (με την countCharacter που ορίσαμε) και να επιλέξουμε ένα από αυτά για να αντικατασταθεί με το δεξί μέρος του κανόνα. Αφού έχουμε επιλέξει ποιο από τα 3 πχ K που βρίσκονται στον προτασιακό τύπο θα αντικαταστήσουμε χρησιμοποιούμε την positionOfTheChracteeInMyString για να μας πει στον προτασιακό μας τύπο το index αυτού του Occurrence του συμβόλου και ανάλογα με την θέση του έχουμε 3 περιπτώσεις.

1η περίπτωση: Άμα το position είναι 0 (που σημαίνει ότι είναι στην αρχή) τότε αλλάζουμε το πρώτο γράμμα του προτασιακό τύπου με αυτό που βρίσκεται δεξιά από τον κανόνα που επιλέξαμε και συναινούμε (concatenation) το υπόλοιπο κομμάτι του προτασιακού τύπου πλην του πρώτου γράμματος.

2η περίπτωση: Άμα το position είναι το length του προτασιακού μας τύπου -1 αυτό σημαίνει ότι το σύμβολο που θα αντικαταστήσουμε βρίσκεται στο τέλος άρα κάνουμε concatenation όλα τα γράμματα του προτασιακού τύπου εκτός του τελευταίου και στο τέλος προσθέτουμε το δεξί μέρος του κανόνα παραγωγής που επιλέξαμε.

3η περίπτωση: Για οποιαδήποτε άλλη τιμή του position σημαίνει ότι το σύμβολο που θα αντικαταστήσουμε βρίσκεται κάπου ανάμεσα στον προτασιακό τύπο άρα θα κάνουμε concatenation την αρχή μέχρι και ένα γράμμα πριν το σύμβολο αυτό με το δεξί μέρος

του κανόνα παραγωγής που θα αντικαταστήσουμε μαζί και από ένα γράμμα μετά του συμβόλου που θα αντικαταστήσουμε μέχρι το τέλος της λέξης.

Στην περίπτωση που ο counter έχει φτάσει την τιμή 20 (γραμμή 307 και παρακάτω), από τους αρχικούς 7 κανόνες, εκτελούνται μόνο οι 1, 2, 3, 7. Η λογική παραμένει παρόμοια με την προηγούμενη.

Σημαντικό είναι εδώ να αναφέρουμε ότι ο τρόπος που το υλοποιήσαμε είναι πρώτα να αντικαταστήσει όλα τα υπάρχοντα Z (άμα υπάρχουν), μετά όλα τα K (άμα υπάρχουν), μετά όλα τα G (άμα υπάρχουν) και τέλος όλα τα M (άμα υπάρχουν).

2.4 Επίδειξη Λύσης

Αρχικά για να κάνουμε εκτελέσιμο το αρχείο μας πρέπει ,καθώς είναι εφαρμογή κονσόλα , να έχουμε εγκαταστημένο στην συσκευή μας έναν compiler για C/C++. Εμείς σε αυτό το παράδειγμα χρησιμοποιούμε τον GCC compiler.

Για την δημιουργία του εκτελέσιμου αρχείου μέσα στον φάκελο που βρίσκεται το αρχείο main.cpp εκτελούμε τις παρακάτω εντολές:

- g++ main.cpp -o ex2.exe (ή ex2.out ανάλογα με το λειτουργικό σύστημα της κάθε συσκευής)

Αυτό θα δημιουργήσει ένα εκτελέσιμο αρχείο ex2.exe (ex2.out)
Με την εκτέλεση του θα έχουμε το παρακάτω αποτέλεσμα:

```
Arxiko symbolo Z
Z

Antikatashash toy 1ou Z me (K)
(K)

Antikatashash toy 1ou K me GM
(GM)

Antikatashash toy 1ou G me Z
(ZM)

Antikatashash toy 1ou Z me (K)
((K)M)

Antikatashash toy 1ou K me GM
((GM)M)

Antikatashash toy 1ou M me +K
((G+K)M)

Antikatashash toy 1ou K me GM
((G+GM)M)

Antikatashash toy 2ou G me v
((G+vM)M)

Antikatashash toy 2ou M me +K
((G+vM)+K)

Antikatashash toy 1ou M me keno
((G+v)+K)

Antikatashash toy 1ou G me Z
((Z+v)+K)

Antikatashash toy 1ou Z me (K)
(((K)+v)+K)

Antikatashash toy 1ou K me GM
(((GM)+v)+K)

Antikatashash toy 1ou K me GM
(((GM)+v)+GM)

Antikatashash toy 2ou G me v
(((GM)+v)+vM)

Antikatashash toy 1ou G me v
(((vM)+v)+vM)

Antikatashash toy 2ou M me keno
(((vM)+v)+v)

Antikatashash toy 1ou M me keno
(((v)+v)+v)
```

```
Arxiko symbolo Z
Z

Antikatashash toy 1ou Z me (K)
(K)

Antikatashash toy 1ou K me GM
(GM)

Antikatashash toy 1ou M me keno
(G)

Antikatashash toy 1ou G me v
(v)
```

Παράδειγμα 2.1 και 2.2 από εκτέλεση της εφαρμογής.

Βλέπουμε ότι η εφαρμογή μας δημιουργεί συμβολοσειρές πολλαπλών μεγεθών χρησιμοποιώντας την γραμματική που μας δόθηκε και καταλήγει πάντα στα τερματικά σύμβολα που ορίστηκαν από την γραμματική.

2.5 Βιβλιογραφικές πηγές

1. Μ. Κ. Βίρβου
   ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ
   Πανεπιστήμιο Πειραιώς
   Εκδόσεις Βαρβαρήγου
   Φρεατύδος 4, 185 37, Πειραιάς
   ISBN: 978-960-7996-15-1

Θέμα 3ο

3.1 Περιγραφή του προβλήματος

Χρησιμοποιώντας τις γλώσσες προγραμματισμού C/C++, καλούμαστε να σχεδιάσουμε και να υλοποιήσουμε ένα συντακτικό αναλυτή top-down, χρησιμοποιώντας την εξής γραμματική:
G→ (M)
M→ YZ
Y→ a | b | G
Z→ *M | -M | +M| ε, οπού ε η κενή συμβολοσειρά

Ο συντακτικός αναλυτής θα αναγνωρίζει ή θα απαντά αρνητικά ως προς την συντακτική ορθότητα της κάθε συμβολοσειράς. Θα επιστρέφει το αντίστοιχο συντακτικό δέντρο. Επίσης θα γίνει επίδειξη με την έκφραση ((a-b)*(a+b)).

Αφού θέλουμε έναν συντακτικό αναλυτή top-down (άρα μία LL γραμματική) εμείς θα πρέπει να χρησιμοποιήσουμε την γραμματική που μας δίνεται ώστε να δούμε τι είδος LL έχουμε.

3.2 Αναγνώριση είδους γραμματικής

Όπως εξηγήσαμε και στην περιγραφή του προβλήματος θα πρέπει να βρούμε τι είδους LL γραμματικής έχουμε.

Για να το πετύχουμε αυτό θα πρέπει πρώτα να ορίσουμε τα σύνολα FIRST , FOLLOW, EMPTY και LOOKAHEAD (ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ Μ.Κ.ΒΙΡΒΟΥ σελ 175-77, 180-181) :

- FIRST(a)
  Το σύνολο FIRST για κάποιο σύμβολο a της γραμματικής, είναι το σύνολο των τερματικών συμβόλων που εμφανίζονται στην αριστερή πλευρά των συμβολοσειρών που παράγονται με το a.
- FOLLOW(a)
  Το σύνολο FOLLOW για κάποιο σύμβολο a της γραμματικής, είναι το σύνολο των τερματικών συμβόλων που μπορούν να εμφανιστούν αμέσως στα δεξιά του α σε κάποια προτασιακή μορφή.
- EMPTY
  Η συνάρτηση EMPTY μας δείχνει για κάθε συντακτικό κανόνα αν αυτός παράγεται τον κενός χαρακτήρας (ε) (με κάποιους περιορισμούς) ή όχι.
- LOOKAHEAD
  Η συνάρτηση LOOKAHEAD μας δείχνει για κάθε συντακτικό κανόνα όλους τους πρώτους πραγματικούς τελικούς χαρακτήρες (δηλαδή χωρίς να υπολογίζουμε το κενό ως χαρακτήρα) που περιμένουμε από κάθε συντακτικό κανόνα.

Άρα εμείς τώρα πρέπει να βρούμε τα σύνολα FIRST , FOLLOW, EMPTY και LOOKAHEAD για την δικιά μας γραμματική.

Με βάση τους ορισμούς των κανόνων θα έχουμε:

1. G -> (M)
2. M -> YZ
3. Y -> a
4. Y -> b
5. Y -> G
6. Z -> *M
7. Z -> -M
8. Z -> +M
9. Z -> ε

FIRST:
(1) => FIRST(G) = { (}
(2) => FIRST(M) = FIRST(Y) = { a , b, (}
(3,4,5) => FIRST(Y) = { a b } and FIRST(G) = { (} = { a b (}
(6,7,8,9) => FIRST(Z) = { * - + ε }

FOLLOW:
(1) => $ ⊆ FOLLOW(G) (I)
(1) => FIRST()) – {ε} ⊆ FOLLOW(M) => {) } ⊆ FOLLOW(M) (II)
(2) => FIRST(Z) – {ε} ⊆ FOLLOW(Y) => { * - + } ⊆ FOLLOW(Y) (III)
(5) => FOLLOW(Y) ⊆ FOLLOW(G) (IV)
(6,7,8) => FOLLOW(Z) ⊆ FOLLOW(M) (V)
(2) => FOLLOW(M) ⊆ FOLLOW(Z) (VI)
(2) => FOLLOW(M) ⊆ FOLLOW(Y) αφού ε⊆ FIRST(Z) (VII)

(V,VI) => FOLLOW(M) = FOLLOW(Z) (VIII)
(VIII,II) => FOLLOW(Z) = FOLLOW(M) = {) }
(VII,III) => FOLLOW(Y) = { * - +) }
(IV,I) => FOLLOW(G) = { * - +) $ }

EMPTY:
(1) => EMPTY(S) = FALSE
(2) => EMPTY(M) = FALSE
(3,4,5) => EMPTY(Y) = FALSE
(6,7,8,9) => EMPTY(Z) = FALSE καθώς από τον (9) οδηγούμαστε το ε

LOOKAHEAD:
(1) => LOOKAHEAD(G -> (M)) = FIRST(( ) = { (}
(2) => LOOKAHEAD(M -> YZ) = FIRST(Y) = { a b (}
(3) => LOOKAHEAD(Y -> a) = FIRST(a) = { a }
(4) => LOOKAHEAD(Y -> b) = FIRST(b) = { b }
(5) => LOOKAHEAD(Y -> G) = FIRST(G) = { (}
(6) => LOOKAHEAD(Z -> *M) = FIRST(*) = { * }
(7) => LOOKAHEAD(Z -> -M) = FIRST(-) = { - }
(8) => LOOKAHEAD(Z -> +M) = FIRST(+) = { + }
(9) => LOOKAHEAD(Z -> ε) = FOLLOW(Z) = {) }

Τώρα για να βρούμε το είδος της LL γραμματικής έχουμε ορίσει έναν αριθμό κ, ο οποίος δείχνει πόσα το πολύ σύμβολα της συμβολοσειράς χρειαζόμαστε για να αποφασίσουμε ποια περισσότερο αριστερή παραγωγή να εφαρμόσουμε.

Για να έχουμε κ=1 πρέπει να ισχύει ότι όλα τα σύνολα LOOKAHEAD για τους κανόνες παραγωγής που το αριστερό μέρος είναι το ίδιο δεν έχουν κοινά στοιχεία. Αυτό ισχύει καθώς έχουμε:

- LOOKAHEAD(Y -> a) ∩ LOOKAHEAD(Y -> b) ∩ LOOKAHEAD(Y -> G) = 0
- LOOKAHEAD(Z -> *M) ∩ LOOKAHEAD(Z -> +M) ∩ LOOKAHEAD(Z -> *M) ∩ LOOKAHEAD(Z -> E) = 0

Άρα η γραμματική μας είναι LL(1)

Ας δούμε τώρα αν αναγνωρίζεται πρέπει να αναγνωρίζεται η έκφραση που μας δόθηκε ((a-b)*(a+b)) από αυτή την γραμματική

ΣΤΟΊΒΑ	**ΕΊΣΟΔΟΣ**	**ΣΤΟΙΧΕΊΟ ΠΊΝΑΚΑ**	**ΠΑΡΑΓΩΓΉ**
$$	((a-b)*(a+b))$	M(G, ()	G -> (M)
$)M(	((a-b)*(a+b))$		
$)M	(a-b)*(a+b))$	M(M,()	M -> YZ
$)ZY	(a-b)*(a+b))$	M(Y,()	Y -> G
$)ZG	(a-b)*(a+b))$	M(G,()	G -> (M)
$)Z)M(	(a-b)*(a+b))$		
$)Z)M	a-b)*(a+b))$	M(M,a)	M -> YZ
$)Z)ZY	a-b)*(a+b))$	M(Y,a)	Y -> a
$)Z)Za	a-b)*(a+b))$		
$)Z)Z	-b)*(a+b))$	M(Z,-)	Z-> -M
$)Z)M-	-b)*(a+b))$		
$)Z)M	b)*(a+b))$	M(M,b)	M -> YZ
$)Z)ZY	b)*(a+b))$	M(Y,b)	Y -> b
$)Z)Zb	b)*(a+b))$		
$)Z)Z	)*(a+b))$	M(Z,))	Z -> ε
$)Z)	)*(a+b))$		
$)Z	*(a+b))$	M(Z,*)	Z -> *M
$)M*	*(a+b))$		
$)M	(a+b))$	M(M,()	M -> YZ
$)ZY	(a+b))$	M(Y,()	Y -> G
$)ZG	(a+b))$	M(G,()	G -> (M)
$)Z)M(	(a+b))$		
$)Z)M	a+b))$	M(M,a)	M -> YZ
$)Z)ZY	a+b))$	M(Y,a)	Y -> a
$)Z)Za	a+b))$		
$)Z)Z	+b))$	M(Z,+)	Z -> +M
$)Z)M+	+b))$		
$)Z)M	b))$	M(M,b)	M -> YZ
$)Z)ZY	b))$	M(Y,b)	Y -> b
$)Z)Zb	b))$		

$)Z)Z	))$	M(Z,))	F -> ε
$)Z)	))$		
$)Z	)$	M(Z,))	F -> ε
$)	)$		
$	$		

Άρα τελικά αφού η στοίβα και η είσοδος κατέληξαν στο $ η έκφραση ((a-b)*(a+b)) αναγνωρίζεται από την γραμματική μας.

Θα χρησιμοποιήσουμε αυτές τις πληροφορίες ώστε να μετατρέψουμε την θεωρία μας σε κώδικα.

3.3 Επεξήγηση Κώδικα

Επιλέξαμε να δημιουργήσουμε μία εφαρμογή κονσόλα, η οποία θα δέχεται μια hard-coded έκφραση θα δημιουργεί το συντακτικό δέντρο μέχρι είτε να εντοπίσει κάποιο λάθος όπου θα σταματάει και θα βγάζει το αντίστοιχο μήνυμα είτε να δημιούργει ολόκληρο το συντακτικό δέντρο όπου τότε θα κάνει κάποιους τελικούς ελέγχους ώστε να βγάλει μήνυμα πως αναγνωρίζει (ή δεν αναγνωρίζει) την έκφρασή μας.

Αρχικά έχουμε φτιάξει μία κλάση TopDownParser η οποία περιέχει έναν constructor που παίρνει ως όρισμα την έκφραση που θέλουμε να ελέγξουμε ως string και αρχικοποιεί κάποιες βασικές παραμέτρους:

- expression παίρνει την έκφραση που θέλουμε να ελέγξουμε
- current θα πάρει στην συνέχεια τον κάθε χαρακτήρα που θα ελέγχουμε
- position θα δείχνει την θέση του επόμενου χαρακτήρα που θα ελέγξουμε
- counter, pipe, pipecounter χρησιμοποιούνται για την πιο όμορφη εμφάνιση του δέντρου

Στην συνέχεια έχουμε μία μέθοδο parser() η οποία αρχίζει την διαδικασία ελέγχου της έκφρασης.
Ξεκινάει καλώντας την μέθοδο matchchar με όρισμα ‘/0’ (θα εξηγήσουμε στην συνέχεια τι ακριβώς κάνει αυτή η μέθοδος) μόλις τελειώσει θα καλέσει την μέθοδο G(). Μόλις ολοκληρωθεί και η μέθοδος G ελέγχει αν ο current είναι ίσος με τον τελικό χαρακτήρα ‘$’, αν αυτό ισχύει θα βγάλει μήνυμα ότι η έκφραση αναγνωρίστηκε αλλιώς ότι δεν αναγνωρίζεται.

Για να γίνονται οι έλεγχοι των χαρακτήρων χρησιμοποιούμε την μέθοδο matchchar.
Η matchchar παίρνει ως όρισμα έναν χαρακτήρα τον οποίο τον συγκρίνει με τον current που έχουμε αυτή την στιγμή. Αν αυτοί οι δύο είναι ίσοι σημαίνει ότι ο χαρακτήρας αναγνωρίστηκε επιτυχώς και μπορούμε να πάμε στον επόμενο current χαρακτήρα και η position να δείχνει την θέση του χαρακτήρα αμέσως μετά από τον επόμενο(αν δεν υπάρχει επόμενος θα δώσουμε στον current την τιμή του τελικού χαρακτήρα $)
Αν όμως ο current και ο χαρακτήρας που δώσαμε δεν είναι ίση σημαίνει ότι υπήρξε κάποιο λάθος αφού εμείς περιμέναμε τον χαρακτήρα που περάσαμε να βρίσκεται σε εκείνη την θέση και όχι ο μη αναγνωρίσιμος χαρακτήρας που υπάρχει στον current.

Επίσης έχουμε την printspaces() που κάνει κάποιους απλούς ελέγχους ώστε να εκτυπώνονται σωστά τα κενά και κάποια “pipes” για να φαίνεται πιο όμορφο το τελικό αποτέλεσμα στην κονσόλα μας.

Τέλος έχουμε τις βασικές μεθόδους που απεικονίζουν κάθε συντακτικό κανόνα της γραμματικής μας. **G() M() Y() Z()**

Η κάθε μία πριν κάνει εκτύπωση στο δέντρο καλεί την printspaces ώστε να δημιουργηθούν τα σωστά κενά.
Στην συνεχεία ελέγχει αν υπάρχουν οι τερματικοί χαρακτήρες LOOKAHEAD κάθε έκφρασης ας πούμε για την G -> (M) η G() θα αρχίσει ελέγχοντας αν υπάρχει η (η οποία είναι ο μοναδικός χαρακτήρας που μπορεί να πάρει με βάση την LOOKAHEAD(G -> (M)) = { (}) και στην συνέχεια θα καλέσει την matchchar ώστε να πάρει τον επόμενο χαρακτήρα. Μόλις ολοκληρωθεί θα καλέσει την M(), η οποία όμως δεν δίνει τερματικούς χαρακτήρες άρα αυτή θα καλέσει αρχικά την Y() και στην συνέχεια την Z(). Η Y και η Z θα ελέγχουν και αυτές για τους LOOKAHEAD χαρακτήρες τους και στην συνέχεια θα κάνουν matchchar ώστε να πάνε στον επόμενο χαρακτήρα.
Η εκτύπωση για τους μη τερματικούς χαρακτήρες έχει δύο μορφές:

- Αν έχουμε ένα μη τερματικό το οποίο όμως θα δεν αναλυθεί σε άλλα τερματικά αντικαθιστάτε κατευθείαν και έχει την μορφή |--A(x) όπου A ο μη τερματικός χαρακτήρας και χ το παράγωγο του A.
  (π.χ αν το Y παράγει το a θα έχουμε |--Y(a))
- Αν έχουμε ένα μη τερματικό το οποίο όμως αναλύεται σε άλλα μη τερματικά θα έχει την μορφή |--A όπου A ο μη τερματικός χαρακτήρας που με pipes προς τα κάτω θα αναλυθεί σε περισσότερος
  (π.χ αν είχαμε το M αυτό θα είχε την μορφή |--M)

Σε οποιοδήποτε σημείο βρούμε κάποιο σύμβολο που δεν αναγνωρίζεται ή βρίσκεται σε λάθος θέση το πρόγραμμα μας θα βγάλει error και θα σταματήσει βγάζοντας το αντίστοιχο μήνυμα με το σύμβολο που περιμέναμε και το σύμβολο που βρήκαμε σε εκείνο το σημείο, αυτό ουσιαστικά θα σημαίνει ότι η έκφραση δεν αναγνωρίζεται.

Τέλος για να τρέξει το πρόγραμμά μας μέσα στην int main() θα δημιουργήσουμε ένα αντικείμενο της κλάσης TopDownParser της κλάσης με όνομα parse και όρισμα την έκφραση ως string ((a-b)*(a+b)). Και στην συνέχεια θα τρέξουμε τον parser καλώντας το μέσα από το αντικείμενο parse που μόλις δημιουργήσαμε.

3.4 Επίδειξη Λύσης

Αρχικά για να κάνουμε εκτελέσιμο το αρχείο μας πρέπει ,καθώς είναι εφαρμογή κονσόλα , να έχουμε εγκαταστημένο στην συσκευή μας έναν compiler για C/C++. Εμείς σε αυτό το παράδειγμα χρησιμοποιούμε τον GCC compiler.

Για την δημιουργία του εκτελέσιμου αρχείου μέσα στον φάκελο που βρίσκεται το αρχείο main.cpp εκτελούμε τις παρακάτω εντολές:

- g++ main.cpp -o ex3.exe (ή ex3.out ανάλογα με το λειτουργικό σύστημα της κάθε συσκευής)

Αυτό θα δημιουργήσει ένα εκτελέσιμο αρχείο ex3.exe (ex3.out)
Με την εκτέλεση του θα έχουμε το παρακάτω αποτέλεσμα:

```
G
|--(
|--M
   |--Y
   |  |--G
   |     |--(
   |     |--M
   |        |--Y(a)
   |        |--Z(-)
   |        |--M
   |           |--Y(b)
   |           |--Z(e)
   |           |--)
   |--Z(*)
   |--M
      |--Y
      |  |--G
      |     |--(
      |     |--M
      |        |--Y(a)
      |        |--Z(+)
      |        |--M
      |           |--Y(b)
      |           |--Z(e)
      |           |--)
      |--Z(e)
      |--)
Successfully recognized ((a-b)*(a+b))!
Process finished with exit code 0
```

Παράδειγμα 3.1 από εκτέλεση της εφαρμογής.

Απ’ ότι βλέπουμε η έκφραση μας δημιουργεί ένα συντακτικό δέντρο με pipes που και τα κενά που χρειάζονται ώστε να είναι ευανάγνωστη η αναγνώριση της έκφρασής μας. Τέλος βγάζει το απαραίτητο μήνυμα πως αναγνωρίστηκε επιτυχώς η έκφραση που το δόθηκε δείχνοντας έτσι την σωστή λειτουργία της εφαρμογής μας καθώς αναγνωρίζει τις εκφράσεις με την γραμματική που μας ζητήθηκε και με τον τρόπο που μας ζητήθηκε.

3.5 Βιβλιογραφικές πηγές

1. Μ. Κ. Βίρβου
   ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ
   Πανεπιστήμιο Πειραιώς
   Εκδόσεις Βαρβαρήγου
   Φρεατύδος 4, 185 37, Πειραιάς
   ISBN: 978-960-7996-15-1

Θέμα 4ο

4.1 Περιγραφή του προβλήματος

Χρησιμοποιώντας το εργαλείο Flex ,και σε συνδυασμό με τις γλώσσες προγραμματισμού C/C++ ,καλούμαστε να δημιουργήσουμε έναν λεκτικό αναλυτή ο οποίος θα αναγνωρίζει ένα υποσύνολο φυσικής γλώσσας όπου τα ονόματα σημείων ορίζονται ως η παράθεση ενός μονό συμβόλου, τα ονόματα ευθείων ορίζονται ως η παράθεση δυο συμβολών, τα ονόματα τρίγωνων ορίζονται ως η παράθεση τριών συμβολών, κ.ο.κ, έως και την περίπτωση οκτάγωνων. Δεν επιτρέπονται επαναλήψεις συμβόλων και δεν γίνεται να έχουμε λιγότερα σύμβολα από ότι χρειαζόμαστε.

4.2 Περιγραφή του προγράμματος επίλυσης

Επιλέξαμε να δημιουργήσουμε μία εφαρμογή κονσόλας η οποία θα παρουσιάζει με συγκεκριμένο τρόπο τους σωστούς και λάθους ορισμούς.
Για την αναγνώριση των συμβολοσειρών θα χρησιμοποιήσουμε κανονικές εκφράσεις
Ορίζουμε ότι ένας σωστός ορισμός μιας συμβολοσειράς είναι οι λέξεις:
point,line,triangle,rectangle,hexagon,heptagon,octagon
οι οποίες είναι όλες γραμμένες με πεζούς λατινικούς χαρακτήρες.
Η κάθε λέξη ακολουθείται από ένα κενό διάστημα και στη συνέχεια από συγκεκριμένο αριθμό (ο οποίος καθορίζεται από την κάθε λέξη) κεφαλαίων λατινικών χαρακτήρων. Ο κάθε κεφαλαίος λατινικός χαρακτήρας απεικονίζει το κάθε σημείο/κορυφή της γραμμής/σχήματος που έχουμε αρχικά ορίσει. Ώστε να εμποδίσουμε τον αναλυτή να αναγνωρίσει συμβολοσειρές με περισσότερα σημεία απ’ ότι χρειαζόμαστε, ορίζουμε ότι η συμβολοσειρά μας πρέπει να τελειώνει με:
\t (tab) ή \n (newline) ή κενό διάστημα ή κόμμα ή τελεία ή δολάριο (τελικός χαρακτήρας)
Για αυτές τις εκφράσεις στην συνέχεια θα γίνει έλεγχος εγκυρότητας ώστε να ελέγξουμε αν όλα τα σημεία που δημιουργούν τις γραμμές/σχήματα είναι διαφορετικά μεταξύ τους και θα τυπωθούν οι σωστοί ορισμοί με μπλε χρώμα και ένα μήνυμα σωστού ορισμού ενώ οι εσφαλμένοι ορισμοί θα τυπωθούν με κόκκινο με ένα μήνυμα για τους εσφαλμένους ορισμούς.

Αν δεν αναγνωριστεί καμία από τις παραπάνω συμβολοσειρές τότε μπορεί να αναγνωριστούν οι λέξεις που ορίσαμε αρχικά, τις οποίες θα ακολουθούσαν από 0 έως άπειροι κενοί χαρακτήρες και αυτούς από 0 έως άπειροι κεφαλαίοι λατινικοί χαρακτήρες. Αν αναγνωριστεί αυτή η έκφραση θα βγάλει μήνυμα εσφαλμένου ορισμού με κόκκινο χρώμα δίπλα στην αναγνωρισμένη έκφραση.

Αν δεν αναγνωριστεί κανένα από τα παραπάνω απλά θα ομαδοποιήσει τα σύμβολα σε συμβολοσειρές και θα τα παρουσιάσει ως είναι.

4.3 Επεξήγηση κώδικα

Τελικά ο κώδικάς μας χωρίζεται σε 3 μέρα στους:

- Ορισμούς για να χρησιμοποιήσουμε τον λεκτικό αναλυτή και μια μέθοδο validpoints η οποία θα ελέγχει αν γίνει έλεγχος εγκυρότητας των σημείων (ώστε να μην υπάρχουν τα ίδια σημεία πάνω από μία φορά)
- Κανόνες οι οποίοι μας αναγνωρίζουν συγκεκριμένες συμβολοσειρές όπως εξηγήσαμε στην περιγραφή του προγράμματος και καθώς αναγνωρίζονται εκτελούν συγκεκριμένα κώδικα.
- Int Main η οποία τρέχει τον λεκτικό

Αναλυτικότερα, για τους ορισμούς έχουμε την yywrap η οποία χρησιμοποιείται για την αναγνώριση του τέλους μιας εισόδου, την μεταβλητή arr που θα παίρνει μέσα στην μέθοδο την συμβολοσειρά με όλα τα σημεία, την μεταβλητή valid που θα χρησιμοποιήσουμε ως Boolean για το αν θα πρέπει να συνεχίσουμε την αναζήτηση μέσα στην επόμενη μέθοδο ή όχι, και τέλος την μέθοδο validpoints την οποία εξηγούμε αναλυτικά στην συνέχεια.

Η μέθοδος validpoints παίρνει 3 ορίσματα ,έναν ακέραιο αριθμό letters που δείχνει πόσα σημεία χρειαζόμαστε (π.χ αν είχαμε “line AB“αυτός ο αριθμός θα ήταν 2 αφού θέλουμε να αναγνωρίσει μόνο 2 σημεία) ,έναν ακέραιο αριθμό start ο οποίος δείχνει την θέση του πρώτου σημείου μέσα στην συμβολοσειρά
(π.χ αν είχαμε “line AB“ αυτός ο αριθμός θα ήταν 5 αφού από αριστερά προς τα δεξιά αν μετρήσουμε από το 0 ,συμπεριλαμβανομένου του κενού, το πρώτο σημείο το βρίσκουμε στην θέση 5) και ένα array από χαρακτήρες που παίρνει το όνομα της γραμμής/σχήματος (π.χ αν είχαμε “line AB” το array από χαρακτήρες θα ήταν ίσο με “line”)
Αρχικά κάθε φορά που καλούμε αυτή την μέθοδο καθαρίζει την arr και θέτει την τιμή 1 στο valid ώστε να ώστε να ξεκινήσουμε τον έλεγχο. Στην συνέχεια θέλουμε να αποθηκεύσουμε στο arr μόνο τα σημεία που αναγνωρίστηκαν. Αυτό το καταφέρνουμε αφού η yytext περιέχει μέσα της όλη την συμβολοσειρά και καθώς γνωρίζουμε την θέση των σημείων μέσα σε αυτήν μπορούμε με ένα loop να αποθηκεύσουμε μόνο τα σημεία αυτά μέσα στην arr. Για να συγκρίνουμε τα σημεία μεταξύ τους χρησιμοποιούμε μία εμφωλευμένη επανάληψη με δύο loop όπου συγκρίνουμε το πρώτο σημείο με όλα τα επόμενα, το δεύτερο σημείο με όλα τα επόμενα (εκτός από τον ίδιον τους τον εαυτό) κ.ο.κ η επανάληψη συνεχίζει μέχρι να τελειώσει χωρίς κανένα κοινό σημείο ή σταματάει (κάνοντας το valid=0)όταν βρει δύο σημεία που ορίζονται από το ίδιο σύμβολο οπότε σε αυτή την περίπτωση βγάζει το αντίστοιχο μήνυμα με κόκκινο χρώμα. Αν όμως δεν βρει κάποιον εσφαλμένο ορισμό όταν τελειώσει η επανάληψη ελέγχει ότι έχουμε έναν σωστό ορισμό (valid=1) και βγάζει το αντίστοιχο μήνυμα με μπλε χρώμα.

Για τους κανόνες εξηγήσαμε ήδη πως θέλουμε να αναγνωρίζονται στην περιγραφή μας. Η περιγραφή μας σε κώδικα μεταφράζεται ως κανονικές εκφράσεις της μορφής:

1. Θέλουμε η συμβολοσειρά να ξεκινάει από μία από τις λέξεις κλειδιά (π.χ “triangle”)
2. Να ακολουθείται από κενό (“triangle “)
3. Να ακολουθείται από κατάλληλο αριθμό σημείων ώστε να φτιάχνεται το σχήμα (στην δικιά μας περίπτωση πρέπει να είναι 3 άρα θα έχουμε “triangle ”[A-Z]{3})
4. Έχουμε ορίσει ότι η αναγνώριση της έκφρασης θέλουμε να τελειώνει με συγκεκριμένο τρόπο (Άρα τελικά θα έχουμε “triangle “[A-Z]{3}[\t\n ,.$])

Αφού αναγνωριστεί αυτή η έκφραση τότε θα καλέσουμε την μέθοδο με τα κατάλληλα ορίσματα για κάθε περίπτωση (π.χ αν αναγνωριστεί ένα “triangle “[A-Z]{3}[\t\n ,.$] θα έχουμε validpoints(3,9,”triangle”);)

Επίσης αναγνωρίζουμε έναν άλλο τύπο κανονικής έκφρασης όπως αναλύσαμε στην περιγραφή η οποία είναι της μορφής:

1. Οποιοδήποτε από τις 8 λέξεις (“point”|”line”|…|”octagon)
2. Από 0 έως άπειρα κενά διαστήματα (“point”|”line”|…|”octagon) [ ]*)
3. Από 0 έως άπειρα κεφαλαίοι λατινικοί χαρακτήρες (“point”|”line”|…|”octagon) [ ]*[A-Z]*)

Και αυτό βγάζει το αντίστοιχο μήνυμα με κόκκινο χρώμα.

Όλα τα υπόλοιπα (συμβολίζονται με την τελεία) τυπώνονται στην κονσόλα όπως τα λαμβάνει

Τέλος στην Int main τρέχει τον λεκτικό αναλυτή μέχρι να κλείσουμε οι ίδιοι την εφαρμογή ή αν δώσουμε δεδομένα από αρχείο σταματάει μόλις φτάσει στο EOF του αρχείου.

4.4 Επίδειξη Λύσης

Αρχικά για να κάνουμε εκτελέσιμο το αρχείο μας πρέπει ,καθώς είναι εφαρμογή κονσόλα , να έχουμε εγκατεστημένο στην συσκευή μας την εφαρμογή Flex , αλλά και έναν compiler για C/C++. Εμείς σε αυτό το παράδειγμα χρησιμοποιούμε τον GCC compiler.

Για την δημιουργία του εκτελέσιμου αρχείου μέσα στον φάκελο που βρίσκεται το αρχείο ex4.l εκτελούμε τις παρακάτω εντολές:

1. flex ex4.l (θα δημιουργηθεί ένα αρχείο lex.yy.c)
2. gcc lex.yy.c -lfl (θα δημιουργήσει ένα εκτελέσιμο αρχείο με το όνομα “a.exe” ή “a.out” ανάλογα με το λειτουργικό σύστημα του κάθε υπολογιστή)

Εκτελώντας το αρχείο και χρησιμοποιώντας ως εισόδους κάποιες προτάσεις όπως
“We want to connect point A to point B”
“and we want to make line AB”
“but we can’t make line AA”
“we can make pentagon ABCDE”
“but if we try pentagon ABCD it doesn’t recognize it as valid since it doesn’t have enough points”
Θα έχουμε το παρακάτω αποτέλεσμα:

```
[anto@archlinux [18:48:55] ~/Documents/Flex]$ ./a.out
We want to connect point A to point B
We want to connect point A <Your point A is valid>
to point B <Your point B is valid>

and we want to make line AB
and we want to make line AB
<Your line AB is valid>
but we can't make line AA
but we can't make line AA
<Invalid line AA you have at least two matching points>
we can make a pentagon ABCDE
we can make a pentagon ABCDE
<Your pentagon ABCDE is valid>
but if we try pentagon ABCD it doesn't recognize it as valid since it doesn't have enough points
but if we try pentagon ABCD <Invalid pentagon ABCD>
 it doesn't recognize it as valid since it doesn't have enough point <Invalid point>
```

Παράδειγμα 4.1 από εκτέλεση της εφαρμογής εισάγοντας εκφράσεις από το πληκτρολόγιο.

Το οποίο είναι και το επιθυμητό αποτέλεσμα καθώς σωστές εκφράσεις είναι μόνο αυτές με μπλε χρώμα ενώ αυτές με κόκκινο έχουν κάποιο λάθος.

Τρέχοντας την ίδια εφαρμογή με ένα αρχείο που έχουμε δημιουργήσει θα δούμε το ίδιο αποτέλεσμα:

```
[anto@archlinux [18:51:06] ~/Documents/Flex]$ ./a.out < test.txt

This is a test for our Flex program.

We want to connect point A <Your point A is valid>
to point B.<Your point B is valid>
 To achieve this we will connect the two point <Invalid point>
s and make line AB
<Your line AB is valid>

we can also make a rectangle TOPI.<Your rectangle TOPI is valid>

but point ABCDE <Invalid point ABCDE>
 doesn't exist as well as triangle ACA <Invalid triangle ACA you have at least two matching points>
or triangle LTT
<Invalid triangle LTT you have at least two matching points>

But what about octagon KLOPQWER <Your octagon KLOPQWER is valid>
, well we can recognize that as well.
But we don't recognize things like Octagon ABCDEFGH or LINE AB or even line  <Invalid line >
ab.
We only recognize point  <Invalid point >
line  <Invalid line >
triangle  <Invalid triangle >
rectangle  <Invalid rectangle >
pentagon  <Invalid pentagon >
hexagon  <Invalid hexagon >
heptagon  <Invalid heptagon >
octagon  <Invalid octagon >
followed by the right ammount
of point <Invalid point>
s written only with upper case latin characters!
```

Παράδειγμα 4.2 από εκτέλεση της εφαρμογής με τη χρήση αρχείου.

Άρα τελικά η εφαρμογή μας λύνει το πρόβλημα που μας τέθηκε καθώς αναγνωρίζει όλα τα σημεία/γραμμές/σχήματα που μας ζητήθηκαν μόνο όταν δεν υπάρχουν επαναλήψεις συμβόλων και τα σύμβολα είναι ακριβώς όσα χρειαζόμαστε.

4.5 Βιβλιογραφικές πηγές

1. M. E. Lesk and E. Schmidt,
   Lex – A Lexical Analyzer Generator
   Bell Laboratories
   Murray Hill, New Jersey 07974
   (1975)
2. Vern Paxson
   Flex, version 2.5
   A fast scanner generator
   Edition 2.5, March 1995,
   This product includes
   software developed by the University of California, Berkeley and its contributors
   (1990)
3. Μ. Κ. Βίρβου
   ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ
   Πανεπιστήμιο Πειραιώς
   Εκδόσεις Βαρβαρήγου
   Φρεατύδος 4, 185 37, Πειραιάς
   ISBN: 978-960-7996-15-1