|  |
| --- |
|  |
| ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ |
|  |
| ΕΡΓΑΣΙΑ 2023-24 |

**πΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**

7 Ιανουαρίου 2024

Σύνταξη από: Δημήτριος Λαζάνας, Π22082

Αντώνιος Τσαλμπούρης, Π22272

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΕΡΓΑΣΙΑ 2023-24

# ΘΕΜΑ 1Ο

Μας ζητήθηκε να υλοποιήσουμε ένα Ντετερμινιστικό Αυτόματο Στοίβας το οποίο θα αναγνωρίζει εκφράσεις κατά τρόπον τον οποίον:

* Όσοι χαρακτήρες “x” εμφανίζονται συνολικά, να εμφανίζονται άλλοι τόσοι χαρακτήρες “y”.
* Κοιτάζοντας την έκφραση από τα αριστερά προς τα δεξιά οι χαρακτήρες των “y” να μην είναι ποτέ περισσότεροι από τους χαρακτήρες “x”.
* Να τυπώνεται η αλληλουχία βημάτων η οποία οδήγησε στην αναγνώριση ή στην απόρριψη της έκφρασης.

Για την υλοποίηση των ζητουμένων της εκφώνησης επιλέξαμε να χρησιμοποιήσουμε την γλώσσα προγραμματισμού C++.

##### ΥΛΟΠΟΙΗΣΗ

###### Class AutomatoStoivas :

Η κλάση AutomatoStoivas που δημιουργήσαμε, διαθέτει μία δομή στοίβας “char” και περιλαμβάνει όλες τις απαραίτητες μεθόδους για την επεξεργασία και αναγνώριση (ή απόρριψη) της έκφρασης που έδωσε ο χρήστης καθώς και για την εκτύπωση των βημάτων που ακολουθήθηκαν.

Η συνάρτηση **processExpression()** της κλάσης δέχεται σαν παράμετρο από την **main()** την έκφραση που πληκτρολόγησε ο χρήστης. Έπειτα, μέσω μιας επανάληψης, για κάθε ξεχωριστό χαρακτήρα της έκφρασης γίνονται ορισμένοι έλεγχοι. Εάν ο χαρακτήρας που διαβαστεί είναι “x” τότε προσθέτουμε ένα “x” στη στοίβα των χαρακτήρων και τυπώνουμε την ενέργεια που μόλις κάναμε καθώς και τα περιεχόμενα της στοίβας (μέσω της **printStack()**). Εάν ο χαρακτήρας που διαβαστεί είναι “y”,τότε πρώτα διασφαλίζουμε ότι η στοίβα δεν είναι κενή και ότι στην κορυφή βρίσκεται ένα “x”. Εφόσον ισχύουν αυτά, αφαιρούμε ένα “x” από την στοίβα ενημερώνοντας για την ενέργεια μας και δείχνοντας την στοίβα. Σε περίπτωση που δεν ισχύουν οι προϋποθέσεις για την αφαίρεση ενός “x” από την στοίβα τότε είναι σαφής ένδειξη ότι η έκφραση δεν ικανοποιεί τους περιορισμούς που τέθηκαν από την εκφώνηση και τυπώνει αντίστοιχο μήνυμα απόρριψης. Στη συνέχεια, αφού έχει τελειώσει η επανάληψη και έχουν γίνει όλοι οι απαραίτητοι έλεγχοι χωρίς να προκύψει απόρριψη λόγω κάποιου “y”, ελέγχουμε την κατάσταση της στοίβας. Εάν στη στοίβα υπάρχει ακόμα κάποιο “x” είναι σαφής ένδειξη πώς τα x ήτανε περισσότερα από τα “y”, οπότε οδηγούμαστε πάλι στην απόρριψη της έκφρασης τυπώνοντας το αντίστοιχο μήνυμα. Σε περίπτωση όμως που η στοίβα μετά το πέρας όλων των ενεργειών είναι κενή (βρίσκεται δηλαδή στην αρχική της κατάσταση) τότε σημαίνει πως ικανοποιήθηκαν όλες οι προϋποθέσεις και η έκφραση που λάβαμε από τον χρήστη αναγνωρίστηκε επιτυχώς.

Η συνάρτηση **printStack()** είναι υπεύθυνη για την εκτύπωση των περιεχομένων της στοίβας κάθε φορά που θέλουμε να τυπώσουμε ένα βήμα. Εκτυπώνει τα περιεχόμενα της στοίβας χρησιμοποιώντας μια βοηθητική στοίβα η οποία δημιουργείται τοπικά προκειμένου μέσα σε μια επαναληπτική διαδικασία να τυπώνει το στοιχείο της κορυφής και έπειτα να το αφαιρεί.

###### main() :

Στην συνάρτηση **main()** αρχικά δημιουργείται ένα αντικείμενο της κλάσης **AutomatoStoivas,** ζητείται από τον χρήστη η έκφραση η οποία θα υποβληθεί σε ελέγχους, καλείται η συνάρτηση επεξεργασίας που περιγράφηκε παραπάνω και τέλος τερματίζεται το πρόγραμμα.

*Παρακάτω δίνονται 3 παραδείγματα εκτέλεσης του προγράμματος (1 αναγνώρισης - 2 απόρριψης)*

# Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, τυπογραφία Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης Περιγραφή που δημιουργήθηκε αυτόματαΘΕΜΑ 2ο

Μας ζητήθηκε να σχεδιάσουμε και να υλοποιήσουμε μια γεννήτρια συμβολοσειρών για την παρακάτω γραμματική, λαμβάνοντας μέριμνα η διαδικασία να τερματίζεται οπωσδήποτε.

<Z>::=(<K>)

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

<G>::=ν|<Z>

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

##### ΥΛΟΠΟΙΗΣΗ

Για την ικανοποίηση των ζητουμένων (σε **C**) χρησιμοποιήθηκαν εκτός από την **main()** άλλες 4 συναρτήσεις (μία για κάθε κανόνα παραγωγής). Αναλυτικά, αυτές ήταν οι:

* **generate\_Z(**char \***result**, int \***pos**, int **depth**)
* **generate\_K**(char \*result, int \***pos**, int **depth**)
* **generate\_G**(char \***result**, int \***pos**, int **depth**)
* **generate\_M**(char \***result**, int \***pos**, int **depth**)

Κάθε μία από αυτές, δέχεται ως παράμετρο το buffer **result**, όπου αποθηκεύονται οι χαρακτήρες που δημιουργήθηκαν, την μεταβλητή **pos** που λειτουργεί ως index για το buffer των χαρακτήρων (και αυξάνεται κατά 1 κάθε φορά που γίνεται κάποια καταχώρηση στο result) και τέλος την μεταβλητή **depth** που μετράει πόσες φορές έγινε επιλογή μη-τερματικού χαρακτήρα αντί για τερματικού.

Οι 4 αυτές συναρτήσεις δηλώνονται σε global scope προκειμένου να μπορούν να κληθούν από οπουδήποτε στο πρόγραμμα. Βάσει του τρόπου υλοποίησης μας, η μία συνάρτηση καλεί την άλλη μέχρι ότου να φτάσουμε στην τελική συμβολοσειρά. Κατά αυτό το τρόπο, όταν παρακολουθούμε τα βήματα που ακολούθησε η τυχαία γεννήτρια στο τερματικό, φαίνονται σαν η συμβολοσειρά να δημιουργήθηκε από τα αριστερά προς τα δεξιά.

Στους κανόνες αντικατάστασης του **G** και του **M**, υπάρχουν διαφορετικές επιλογές για την αντικατάσταση αυτών των χαρακτήρων. Προκειμένου να διασφαλίσουμε την τυχαιότητα της επιλογής του κανόνα αντικατάστασης χρησιμοποιούμε μια γεννήτρια τυχαίων αριθμών στις αντίστοιχες συναρτήσεις **generate\_G** και **generate\_M**.

Κάθε φορά, αφού γίνει η επιλογή του κανόνα βάσει του οποίου θα γίνει αντικατάσταση του στοιχείου, εμφανίζουμε στον χρήστη τον κανόνα αυτό καθώς και το αποτέλεσμα της έκφρασης, ως έχει, εκείνη τη δεδομένη στιγμή. Ο τρόπος που επιλέξαμε να διαχειριστούμε το «ε» (κενό) ήτανε απλώς να τυπώσουμε στον χρήστη τον κανόνα που επιλέχθηκε και να του δείξουμε την έκφραση δίχως να κάνουμε κάποια αλλαγή σε αυτή.

Επιπλέον, όπως αναφέραμε και προηγουμένως, έπρεπε να διασφαλίσουμε πως ο αλγόριθμος μας, θα επέλεγε κάποια στιγμή έναν κανόνα που οδηγεί σε τερματικό σύμβολο παρά την τυχαιότητα του. Έτσι, κάθε φορά που επιλεγόταν τυχαία κάποιος μη τερματικός κανόνας αυξάναμε την τιμή της μεταβλητής **depth** κατά 1. Κατ’ επέκταση σε κάθε επιλογή κανόνα προβλέψαμε εκτός από την τυχαιότητα, εάν η τιμή της μεταβλητής depth είναι μεγαλύτερη από 10, τότε να γίνεται απευθείας η επιλογή του κανόνα που οδηγεί σε τερματικό σύμβολο.

Τέλος, στην κύρια συνάρτηση **main()** δεν γίνεται τίποτα παραπάνω παρά αρχικοποιήσεις μεταβλητών (result, pos, γεννήτριας αριθμών) καθώς και κλήση της πρώτης συνάρτησης **generate\_Z** που ανταποκρίνεται στον πρώτο κανόνα αντικατάστασης. Αφού ολοκληρωθεί η δημιουργία της συμβολοσειράς και το πρόγραμμα επιστρέψει στην **main()**, τυπώνουμε για την χρήστη το τελικό αποτέλεσμα και το πρόγραμμα τερματίζεται.

*Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, τυπογραφία

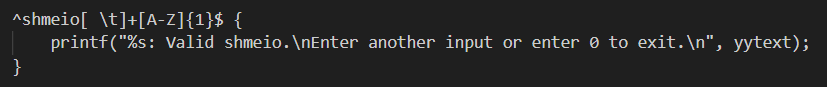
Περιγραφή που δημιουργήθηκε αυτόματαΠαράδειγμα κώδικα καθώς και εκτέλεσης του προγράμματος:*

# ΘΕΜΑ 4Ο

Μας ζητήθηκε να υλοποιήσουμε ένα πρόγραμμα FLEX (σε συνδυασμό με C) το οποίο θα αναγνωρίζει τα ονόματα σημείων ως την παράθεση ενός συμβόλου, τα ονόματα τριγώνων ως τη παράθεση 3 συμβόλων κ.ο.κ, έως και την περίπτωση οκταγώνων. Σημαντική παράμετρος επίσης είναι ότι δεν επιτρέπονται οι επαναλήψεις συμβόλων όταν δίνεται ένα σχήμα.

##### ΥΛΟΠΟΙΗΣΗ

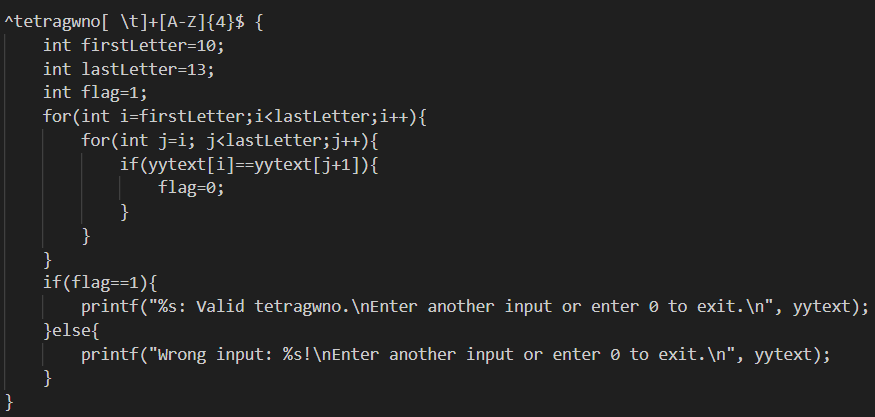
Για την υλοποίηση των ζητουμένων, και για την αποφυγή προβλημάτων encoding αποφασίσαμε το πρόγραμμα μας να χρησιμοποιεί **μόνο λατινικούς χαρακτήρες**. Το λεξιλόγιο που αναγνωρίζει το πρόγραμμα μας περιγράφεται αναλυτικά στο χρήστη με την εκκίνηση του προγράμματος, για πιο εύκολη εμπειρία χρήσης.

Στους κανόνες μετάφρασης του προγράμματος έχουμε δηλώσει όλα τα αποδεκτά μοτίβα προτάσεων. Κάθε φορά που το πρόγραμμα αναγνωρίζει το μοτίβο μίας έκφρασης, όπως αυτή περιεγράφηκε στους κανόνες, προβαίνει στις αντίστοιχες ενέργειες που προγραμματίστηκαν. Για παράδειγμα, στην πιο εύκολη υλοποίηση, στο «σημείο», κάθε φορά που αναγνωρίζεται είσοδος λέξης «σημείο» ακολουθούμενη από κενό και ένα κεφαλαίο γράμμα της λατινικής αλφαβήτου, εκτυπώνεται μήνυμα αποδοχής της έκφρασης. *Δίνεται επίσης η επιλογή ο χρήστης να δώσει νέα είσοδο ή να εισάγει το ‘0’ προκειμένου να τερματιστεί το πρόγραμμα (υπάρχει αντίστοιχος κανόνας μετάφρασης του ‘0’ σε εντολή exit(0);)*

**^**:αρχή γραμμής, **$**:τέλος γραμμής

Η διαδικασία αναγνώρισης έκφρασης περιπλέκεται περισσότερο όταν ο χρήστης εισάγει μεγαλύτερα σχήματα. Αυτό συμβαίνει διότι, πρέπει να κάνουμε τους απαραίτητους ελέγχους προκειμένου να διαπιστώσουμε εάν έχει επαναληφθεί κάποιο σύμβολο. Έτσι, για παράδειγμα όταν ο χρήστης εισάγει το σωστό πρότυπο έκφρασης για ένα τετράγωνο (δηλαδή η λέξη τετράγωνο ακολουθούμενη από κενό και πέντε κεφαλαίους χαρακτήρες), τότε εμείς γνωρίζοντας σε ποια θέση βρίσκεται ο πρώτος και ο τελευταίος χαρακτήρας ξεκινάμε μία διαδικασία 2 επαναλήψεων, όπου ελέγχεται κάθε κεφαλαίο σύμβολο εάν είναι ίδιο με κάποιο από τα επόμενα του. Σε περίπτωση που είναι, μέσω μιας μεταβλητής flag τυπώνουμε κατάλληλο μήνυμα (η flag μεταβλητή αρχικοποιείται ως αληθής και αν προκύψει ταύτιση συμβόλου με κάποιο άλλο αλλάζει σε ψευδής).

*Ακολουθεί παράδειγμα κανόνα μετάφρασης για την περίπτωση του τετραγώνου:*



*Αντιστοίχως λειτουργούν και οι υπόλοιποι κανόνες μετάφρασης για τα άλλα σχήματα με μόνες διαφοροποιήσεις στο όνομα του σχήματος, στον αριθμό κεφαλαίων χαρακτήρων που αναμένονται, και στις μεταβλητές ‘firstLetter’ και ‘lastLetter’ που μαρτυρούν σε ποια θέση του πίνακα ‘yytext’ βρίσκονται το πρώτο και το τελευταίο σύμβολο κάθε σχήματος*

Συμπληρωματικά, έχουμε προσθέσει έναν κανόνα μετάφρασης “.+” για να εκτυπώνει μόνο 1 μήνυμα απόρριψης αντί για περισσότερα κάθε φορά που βλέπει μια «αλλόκοτη» έκφραση.

Τέλος, στις «βοηθητικές διαδικασίες», έχουμε τη συνάρτηση “yywrap” που χρειάζεται ο flex για να καταλάβει το τέλος μιας εισόδου, καθώς και την main(), η οποία τυπώνει στον χρήστη τα αρχικά μηνύματα (μεταξύ των οποίων και το «λεξικό»), καλεί την απαραίτητη συνάρτηση yylex() και τερματίζεται.

*Παρακάτω δίνεται παράδειγμα εκτέλεσης του προγράμματος:*

