# Εισαγωγή

…

# Άσκηση Α

## Εισαγωγή

Η πρώτη άσκηση έχεις ως στόχο την υλοποίηση ενός προγράμματος που δέχεται ως είσοδο ένα αρχείο πηγαίου κώδικα και μέσω μιας συγκεκριμένης διαδικασίας αναγνωρίζει τον αριθμό των παρενθέσεων και αποφαίνεται για την ορθότητά τους.

Το πρόγραμμα σε αρχικό στάδιο δέχεται ως όρισμα τη διαδρομή του αρχείο που θα λάβει ως είσοδο ώστε να αρχίσει την επεξεργασία. Έπειτα, διαβάζει τα περιεχόμενά του και αφαιρεί όλους τους χαρακτήρες εκτός των παρενθέσεων. Το πρόβλημα που καλείται το πρόγραμμα να αντιμετωπίσει έγκειται στον τρόπο με τον οποίο θα αναγνωριστούν οι ανοιχτές και κλειστές παρενθέσεις λαμβάνοντας υπ’ όψη τη σειρά με την οποία βρίσκονται.

## Επίλυση

Με τη χρήση ενός Ντετερμινιστικού Αυτόματου Στοίβας (ΝΑΣ) το ζήτημα αντιμετωπίζεται αρκετά πιο εύκολα. Σύμφωνα με τον ορισμό ενός ΝΑΣ, στη δική μας περίπτωση ορίζεται ως εξής:

**Μ = (K, T, V, p, k1, A1, F)**

**Όπου:**

Σύνολο Καταστάσεων (K)  
{ **k0**, **k1**, **k2** }

Αλφάβητο Εισόδου (T)  
{ **(**, **)** }

Αλφάβητο Συμβόλων (V)  
{ **(**, **)**, **$** }

Συνάρτηση Στοίβας (p)  
f(**k0**, **ε**, **ε**) = { (**k1**, **$**) }  
f(**k1**, **ε**, **(**) = { (**k1**, **(**) }  
f(**k1**, **(**, **)**) = { (**k1**, **ε**) }  
f(**k1**, **$**, **ε**) = { (**k2**, **ε**) }

Αρχική Κατάσταση (k1)  
**k0**

Αρχικό Σύμβολο Στοίβας (A1)  
**$**

Σύνολο Τελικών Καταστάσεων (F)  
{ **t2** }

Με άλλα λόγια, κατά την έναρξη του προγράμματος, αφού διαβαστούν τα περιεχόμενα του αρχείου, η στοίβα σύμφωνα με τη συνάρτησή της, αρχικοποιείται και μεταβαίνει στην κατάσταση **k1** με το σύμβολο **$** στο πυθμένα της στοίβας. Στη συνέχεια, εάν δεχτεί η στοίβα το σύμβολο **(** εισάγει τη παρένθεση και παραμένει στην κατάσταση **k1** μέχρις ότου το σύμβολο **)** να διαβαστεί. Σε αυτή τη περίπτωση, αφαιρείται η προηγούμενη παρένθεση **(** από τη στοίβα. Συνεχίζουμε αυτή τη διαδικασία μέχρι να καταναλωθεί η συμβολοσειρά από παρενθέσεις και να μεταβούμε στη τελική κατάσταση **k2**.

Κατά την κατανάλωση της συμβολοσειράς, εάν διαπιστωθεί πως δεν υπάρχει καθορισμένη μετάβαση σύμφωνα με τη συνάρτηση της στοίβας, η διαδικασία σταματάει και οι παρενθέσεις δεν αναγνωρίζονται.

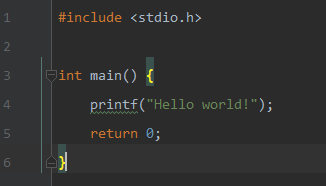
## Εκτέλεση

Το πρόγραμμα έχει υλοποιηθεί στη **Python 3** και δέχεται **1** όρισμα.

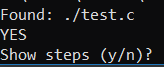
Ενδεικτικά, μέσω της γραμμής εντολών το πρόγραμμα εκτελείται ως εξής:

* Κάνουμε **cd compilers-project**
* Κάνουμε **cd exercise-1**
* Εκτελούμε το πρόγραμμα **python parentheses.py ./test.c**

Αν υποθέσουμε ότι το αρχείο **test.c** περιέχει τον πηγαίο κώδικα για ένα από hello world πρόγραμμα στη C και βρίσκεται στον ίδιο φάκελο με το αρχείο **parentheses.py**:



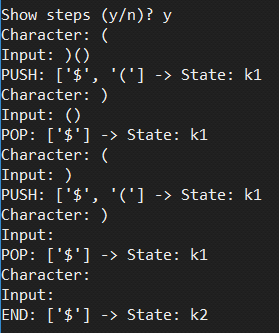
Τότε παρατηρούμε ότι το πρόγραμμά μας ανταποκρίνεται ως εξής:



Το πρόγραμμα βρήκε επιτυχώς το αρχείο **test.c** και αφού ανέλυσε τα περιεχόμενα και τις παρενθέσεις, τυπώνει «**YES**» στη γραμμή εντολών. Η ένδειξη αυτή σημαίνει πως οι παρενθέσεις που βρίσκονται στο πηγαίο κώδικα του test.c είναι σωστές.

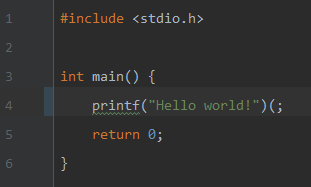
Πράγματι αφού έχουμε τις εξής παρενθέσεις: **()()**

Στη συνέχεια, παρατηρούμε πως το πρόγραμμα δίνει την επιλογή στο χρήστη να του εκτυπώσει αναλυτικά όλα τα βήματα που οδήγησαν σε αυτό το αποτέλεσμα.

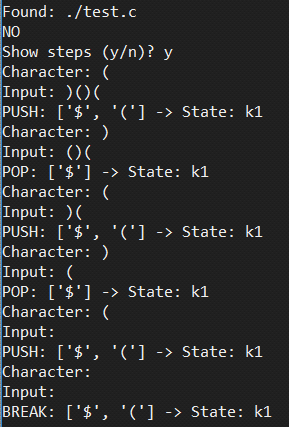


Στο τελευταίο βήμα, μπορούμε να διακρίνουμε πως η στοίβα περιέχει το σύμβολο **$** και βρίσκεται στην κατάσταση **k2**, άρα η συμβολοσειρά αναγνωρίζεται.

Σε μια άλλη περίπτωση που οι παρενθέσεις στο αρχείο του πηγαίου κώδικα δεν ήταν ισορροπημένες:



Το αποτέλεσμα θα ήταν «**ΝΟ**» αφού πράγματι θα είχαμε τις εξής παρενθέσεις: **()()(**



Έτσι, διαπιστώνουμε ότι η στοίβα παρέμεινε στη κατάσταση **k1** με μια εκκρεμής ανοιχτή παρένθεση.

# Άσκηση Β

## Εισαγωγή

Η δεύτερη άσκηση έχει ως θέμα την υλοποίηση ενός προγράμματος-γεννήτριας συμβολοσειρών σύμφωνα με την γραμματική που ορίζεται στην Ενότητα 3.2.4 του βιβλίου. Το πρόγραμμα όπως και στη πρώτη άσκηση έχει υλοποιηθεί στη γλώσσα προγραμματισμού Python, εφόσον είναι αρκετά ευέλικτη σε τέτοια θέματα παραγωγής συμβολοσειρών.

## Επίλυση

Ορίζουμε μια γραμματική G σύμφωνα με το παράδειγμα του βιβλίου.

**G = (S, P, N, T)**

Όπου:

Σύμβολο Εκκίνησης (S)  
**<έκφραση>**

Σύνολο Κανόνων Παραγωγής (P)  
**<έκφραση>** ::= <όρος> | <έκφραση> + <όρος>  
**<όρος>** ::= <παράγοντας> | <όρος> \* <παράγοντας>  
**<παράγοντας>** ::= a | b | c

Σύνολο μη Τερματικών Συμβόλων (N)  
{ **<έκφραση>**, **<όρος>**, **<παράγοντας>** }

Σύνολο Τερματικών Συμβόλων (Τ)  
{ **a**, **b**, **c** }

Η γραμματική G έχει οριστεί με BNF (Backus-Naur Form), μεταγλώσσα που χρησιμοποιείται για τον ορισμό γλωσσών χωρίς συμφραζόμενα. Η παρούσα γραμματική δεν έχει συμφραζόμενα, γι’ αυτό με τη χρήση BNF μπορούμε εύκολα να υλοποιήσουμε ένα πρόγραμμα για τη προτάσεων αυτής της γλώσσας.

Το πρόγραμμα σε αρχικό στάδιο, αρχίζει τη παραγωγή μιας πρότασης από την **<έκφραση>**. Έπειτα, επιλέγει τυχαία και εφαρμόζει τον κατάλληλο κανόνα για την αντικατάσταση του μη τερματικού συμβόλου. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου να μην υπάρχει η <έκφραση> στη πρότασή μας. Η ίδια διαδικασία εφαρμόζεται και για τα σύμβολα **<όρος>** και **<παράγοντας>**. Όσο υπάρχουν μη τερματικά σύμβολα, επαναλαμβάνεται η εφαρμογή των κανόνων της γραμματικής.

Επειδή η εφαρμογή των κανόνων είναι τυχαία, μπορεί να καταλήξουμε σε μεγάλους βρόχους επανάληψης, δηλαδή σε μεγάλες προτάσεις που προέκυψαν έπειτα από την εφαρμογή πολλών κανόνων στη σειρά. Γι’ αυτό το λόγο, έχει τεθεί ένα όριο στον αριθμό των κανόνων που εφαρμόζονται. Έτσι αν έχουμε καταλήξει σε πρόταση που περιέχει μη τερματικά σύμβολα έπειτα από 50 εφαρμογές κανόνων, το πρόγραμμα σταματάει τη διαδικασία και εφαρμόζει τους πιο σύντομους κανόνες που θα μας οδηγήσουν σε τελικό αποτέλεσμα.

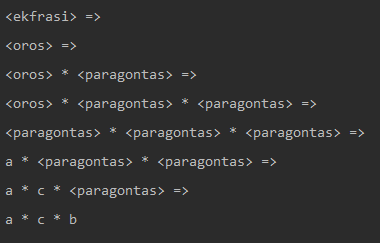
## Εκτέλεση

Το πρόγραμμα έχει υλοποιηθεί στη **Python 3** και δεν δέχεται ορίσματα.

Ενδεικτικά, μέσω της γραμμής εντολών το πρόγραμμα εκτελείται ως εξής:

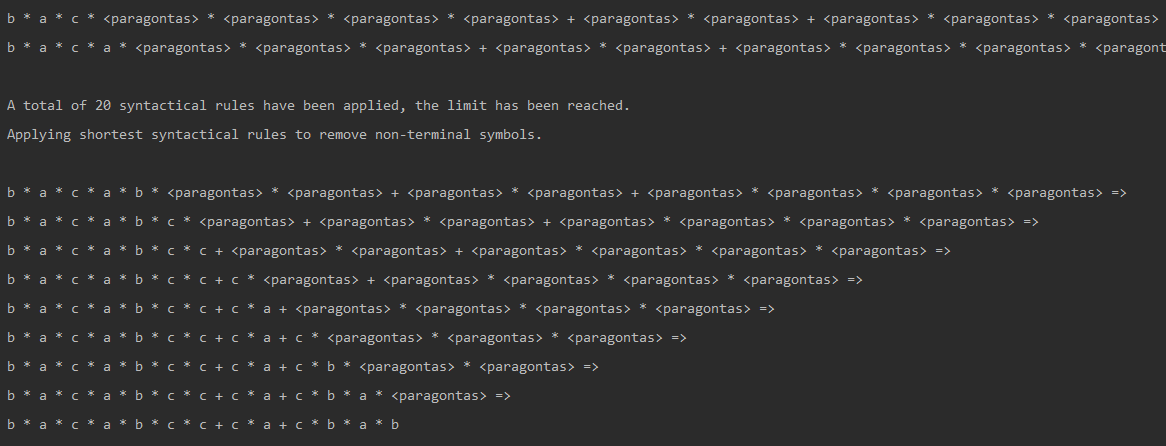
* Κάνουμε **cd compilers-project**
* Κάνουμε **cd exercise-2**
* Εκτελούμε το πρόγραμμα **python generator.py**

Αμέσως, με την εκτέλεση του προγράμματος παρατηρούμε πως έχει δημιουργηθεί μια πρόταση έπειτα από την εφαρμογή των κανόνων γραμματικής:



Σε αυτή τη περίπτωση, το πρόγραμμα επέλεξε να αντικαταστήσει την **<έκφραση>** με **<όρος>**, έπειτα **<όρος>** με **<όρος> \* <παράγοντας>** κ.ο.κ. Έτσι, φτάσαμε στη συμβολοσειρά **a \* c \* b**.

Υπάρχουν και άλλες περιπτώσεις όπου το πρόγραμμα μπορεί να συνεχίσει τη εφαρμογή αναδρομικών κανόνων αρκετές φορές:



Γι’ αυτό το λόγο, θέτουμε ένα όριο στην εφαρμογή των κανόνων ώστε αν έχουμε μεγάλους βρόχους επανάληψης, σταματάει η διαδικασία και καταλήγουμε όσο το συντομότερο δυνατό σε επιθυμητό αποτέλεσμα.

Στο παράδειγμά μας, το όριο έχει τεθεί σε 20 εφαρμογές κανόνων για τη παρουσίαση του παραδείγματος, όμως ρεαλιστικά το πρόγραμμα σταματάει μετά από **50 εφαρμογές**, που είναι αρκετά πιο σπάνιο.

# Άσκηση Γ

Εισαγωγή

# Το θέμα της τρίτης άσκησης είναι να εξετάσουμε αν η παρακάτω γραμματική είναι LL(1).

**S [A]**

**A BE**

**B x | y | S**

**E :A | +A | ε**

Επιπλέον μας ζητήθηκε να κατασκευάσουμε ένα συντακτικό αναλυτή top-down που να αναγνωρίζει την εκάστοτε δοθείσα συμβολοσειρά ή να απαντά αρνητικά ως προς την συντακτική της ορθότητα. Η γλώσσα προγραμματισμού η οποία επιλέχθηκε είναι η Java.

## Εκτέλεση

Αρχικά θα δείξουμε ότι η γραμματική είναι LL(1).

**FIRST**

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

FIRST(**A**) = { x, y, [ }

FIRST(**B**) = { x, y, [ }

FIRST(**E**) = { :, +, ε }

**FOLLOW**

1. **$ FOLLOW(S)**
2. **S [A]**

S [A]  **FIRST(]) – {ε} FOLLOW(A) = {]} FOLLOW(A)**

1. **A BE**

**A εΒΕ FIRST(E) – {ε} FOLLOW(B) = {:, +} FOLLOW(B)**

**A εΒΕ όπου ε FIRST(E) FOLLOW(A) FOLLOW(B)**

**A ΒΕ FOLLOW(A) FOLLOW(E)**

1. **E :A**

**Ε :Α FOLLOW(E) FOLLOW(A)**

1. **E +A**

**Ε +Α FOLLOW(E) FOLLOW(A)**

1. **B S**

**B S FOLLOW(B) FOLLOW(S)**

**Επομένως έχουμε:**

FOLLOW(**S**) = {$, :, +, } FOLLOW(**Α**) = {]}

FOLLOW(**B**) = {:, +, ]} FOLLOW(**Ε**) = {]}

**EMPTY**

EMPTY(**[A]**) = FALSE

EMPTY(**x**) = FALSE

EMPTY(**y**) = FALSE

EMPTY(**S**) = FALSE

EMPTY(**BE**) = EMPTY(**B**) AND EMPTY(**E**) = FALSE AND TRUE = FALSE

EMPTY(**:A**) = EMPTY(**:**) AND EMPTY(**A**) = FALSE

EMPTY(**+A**) = EMPTY(**+**) AND EMPTY(**A**) = FALSE

EMPTY(**ε**) = TRUE

**LOOKAHEAD**

LOOKAHEAD(**S [A]**) = FIRST(**[**) FIRST(**A]**) = {[, x, y}

LOOKAHEAD(**A ΒΕ**) = FIRST(**B**) FIRST(**E**) = {[, x, y, :, +, ε}

LOOKAHEAD(**B x**) = FIRST(**x**) = {x}

LOOKAHEAD(**B y**) = FIRST(**y**) = {y}

LOOKAHEAD(**B S**) = FIRST(**S**) = {[}

LOOKAHEAD(**Ε :Α**) = FIRST(**:A**) = {:}

LOOKAHEAD(**Ε +Α**) = FIRST(**+A**) = {+}

LOOKAHEAD(**Ε ε**) = FIRST(**ε**) FOLLOW(**E**) = {ε, [}

Αρά έχουμε:

LOOKAHEAD(**B x**) LOOKAHEAD(**B y**) LOOKAHEAD(**B S**) =

LOOKAHEAD(**Ε :Α**) LOOKAHEAD(**Ε +Α**) LOOKAHEAD(**Ε +Α**) =

**Επομένως η γραμματική που δόθηκε είναι LL(1).**

Στη συνέχεια θα κατασκευάσουμε τον συντακτικό πίνακα, ο οποίος θα χρησιμοποιηθεί για την κατασκευή ενός αποδοτικού συντακτικού αναλυτή από πάνω προς τα κάτω.

1. **S [A]**

FIRST(**[**) = [

Άρα:

Μ(**S**, **[**) = S [A]

1. **A ΒΕ**

FIRST(**BE**) = FIRST(**B**) = {x, y, [}

Άρα:

Μ(**Α, x**) = **A ΒΕ**

Μ(**Α, y**) = **A ΒΕ**

Μ(**Α, [**) = **A ΒΕ**

1. **B -> x | y | S**
   1. **B x**

**FIRST(x) = {x} Άρα Μ(Β, x) = B x**

* 1. **B y**

**FIRST(y) = {y} Άρα Μ(Β, y) = B y**

* 1. **B S**

**FIRST(S) = {[} Άρα Μ(Β, [) = B S**

1. **E -> :A | +A | ε**
   1. **Ε :Α**

**FIRST(:) = {:} Άρα Μ(E, :) = Ε :Α**

* 1. **Ε +Α**

**FIRST(+) = {+} Άρα Μ(Β, +) = Ε +Α**

* 1. **Ε ε**

**FIRST(ε) = {ε} ε FIRST(ε)**

**Αρά εξετάζουμε το FOLLOW(E) = {]}**

**Οπότε Μ(E, ]) = Ε ε**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| V/T | [ | ] | : | + | x | y | $ |
| S | S [A] |  |  |  |  |  |  |
| A | **A ΒΕ** |  |  |  | **A ΒΕ** | **A ΒΕ** |  |
| B | **B S** |  |  |  | **B x** | **B y** |  |
| E |  | **Ε ε** | **Ε :Α** | **Ε +Α** |  |  |  |

**Πίνακας 1** Συντακτικός Πίνακας

**Λειτουργία LL(1) Parser**

Ο LL(1) parser είναι υλοποιημένος σε γλώσσα προγραμματισμού JAVA. Συγκεκριμένα υλοποιήσαμε μια τάξη με όνομα LL1\_Parser και μέσα σε αυτή περιέχονται οι μέθοδοι με τις απαραίτητες λειτουργίες. Το αρχείο java με τον κώδικα του συντακτικού αναλυτή βρίσκεται μέσα στο φάκελο **exercise-3/ll1\_parser**.

**Μεταβλητές**

* **private String input**Είναι private μεταβλητή τύπου string στην οποία γίνεται ανάθεση της συμβολοσειράς που είναι προς αναγνώριση.
* **private int indexOfInput**

Είναι private ακέραια μεταβλητή και χρησιμοποιείται για να κρατάει κάθε φορά το index του input στο οποίο βρισκόμαστε. Η αρχική της τιμή είναι -1.

* **private Stack<String> pareserStack**

Εδώ έχουμε μια private μεταβλητή η οποία έχει το ρόλο της στοίβας και δέχεται string.

* **String[][]** **syntaxArray**

Η μεταβλητή syntax Array αποτελεί ένα δυσδιάστατο πίνακα ο οποίος αντιπροσωπεύει τον συντακτικό πίνακα που κατασκευάσαμε παραπάνω.

* **private String[] termSymbols**

Είναι private μεταβλητή, με την οποία υλοποιούμε ένα πίνακα όπου δέχεται string. Ο συγκείμενος πίνακας περιέχει τα τερματικά σύμβολα.

* **private String[] nonTermSymbols**

Είναι private μεταβλητή, με την οποία υλοποιούμε ένα πίνακα όπου δέχεται string. Ο συγκείμενος πίνακας περιέχει τα μη τερματικά σύμβολα.

**Μέθοδοι**

* **public LL1\_Parser(String input)**

Αποτελεί τον κατασκευαστή της τάξης και δέχεται μία παράμετρο τύπου string. Έτσι όταν δημιουργείται ένα αντικείμενο της τάξης LL1\_Parser, μέσω της παραμέτρου αυτής, μπορούμε να περάσουμε την συμβολοσειρά στον Parser μας. Επιπλέον γίνεται και η δημιουργία της στοίβας.

* **public void parsingAlgorithm()**

Καλώντας αυτή τη μέθοδο, η οποία είναι public και τύπου void, ξεκινάει η διαδικασία της ανάλυσης.   
Μέσα σε αυτή περιέχεται όλη η βασική λειτουργία του parser.

Αρχικά εισάγεται στη στοίβα, κάτω, το σύμβολο $ και στη κορυφή το αρχικό σύμβολο S. Ο συντακτικός αναλυτής διαβάζει τα σύμβολα από την είσοδο ένα προς ένα.

Οι ενέργειες του αναλυτή καθορίζονται από το τρέχον σύμβολο εισόδου (*έστω a*) και το σύμβολο όπου βρίσκεται στη κορυφή της στοίβας (*έστω x*).

Υπάρχουν **4 περιπτώσεις** για τον συντακτικό αναλυτή:

* + Εάν το **x = a** και **a $** τότε το σύμβολο εισόδου είχε αναγνωριστεί και το **x** βγαίνει από την στοίβα.
  + Εάν **x = a = $** τότε ο αναλυτής έχει δεχθεί τη συμβολοσειρά εισόδου και έχει τελειώσει την εργασία αναγνώρισης.
  + Εάν το **x** είναι μη τερματικό τότε ο αναλυτής ανατρέχει στο **syntaxArray(x, a)** του συντακτικού πίνακα.
  + Εάν στη θέση αυτή υπάρχει κανόνας παραγωγής τότε το x στη στοίβα αντικαθίσταται με τα σύμβολα στο δεξί μέλος της παραγωγής.

Τα σύμβολα μπαίνουν στη στοίβα με αντιστροφή σειρά. Δηλαδή το αριστερότερο σύμβολο παραγωγής καταλήγει στη κορυφή της λίστας.

Αν η θέση του πίνακα είναι κενή, τότε ο αναλυτής βγάζει μνήμα λάθους. Η συμβολοσειρά δεν αναγνωρίζεται από τη γραμματική σε αυτή την περίπτωση.

* **private void pushRuleInStack(String rule)**

Αυτή η μέθοδος είναι private τύπου void και δέχεται μια παράμετρο τύπου string. Η μέθοδος αυτή καλείται για την εισαγωγή του κανόνα στη στοίβα.

* **private boolean isTerminal(String inputString)**

Αυτή η μέθοδος είναι private, τύπου Boolean. Δέχεται μια παράμετρο τύπου string, η οποία ελέγχεται αν είναι τερματικό σύμβολο. Αν είναι η παράμετρος που δόθηκε είναι τερματικό σύμβολο επιστρέφεται η τιμή true, αλλιώς επιστρέφεται η τιμή false.

* **private String read()**

Η μέθοδος αυτή είναι private, τύπου string. Όταν καλείται αυξάνει τη μεταβλητή indexOfInput κατά ένα και στη συνέχεια επιστρέφει, από το Input, το χαρακτήρα που βρίσκεται στη συγκεκριμένη θέση.

* **private void errorLogger(String errorMsg)**

Η συγκεκριμένη μέθοδος είναι private, τύπου void, και στην περίπτωση που δεν αναγνωριστεί η συμβολοσειρά που δόθηκε εμφανίζει το κατάλληλο μήνυμα.

* **private String getRuleFromArray(String nonTerminal, String terminal)**

Η μέθοδος αυτή είναι private και τύπου string. Επιπλέον δέχεται δυο παραμέτρους τύπους string. Όταν καλείται, δίνονται ως πρώτη παράμετρος ένα μη τερματικό σύμβολο, και ως δεύτερη ένα τερματικό σύμβολο. Με βάση αυτά ελέγχεται αν υπάρχει ο κανόνας στο πίνακα και επιστρέφεται από τη μέθοδο. Αλλιώς εμφανίζει το κατάλληλο μήνυμα στο χρήστη.

* **private int getTermSymbolIndex(String inputString)**

Η μέθοδος αυτή είναι private, ακέραιου τύπου και δέχεται μια παράμετρο τύπου string. Στη συγκρίνεται η παράμετρος που δίνεται συγκρίνεται με τα στοιχεία που περιέχονται στον πίνακα με τα τερματικά σύμβολα. Αν η παράμετρος είναι ιδιά με κάποια από τα σύμβολα, επιστρέφεται το index στο οποίο βρέθηκε. Αλλιώς εμφανίζεται το κατάλληλο μήνυμα και επιστρέφεται η τιμή -1.

* **private int getNonTermSymbolIndex(String inputString)**

Η μέθοδος αυτή είναι private, ακέραιου τύπου και δέχεται μια παράμετρο τύπου string. Στη συγκρίνεται η παράμετρος που δίνεται συγκρίνεται με τα στοιχεία που περιέχονται στον πίνακα με τα μη τερματικά σύμβολα. Αν η παράμετρος είναι ιδιά με κάποια από τα σύμβολα, επιστρέφεται το index στο οποίο βρέθηκε. Αλλιώς εμφανίζεται το κατάλληλο μήνυμα και επιστρέφεται η τιμή -1.

* **private String peekOfStack()**

Η μέθοδος αυτή είναι Private, τύπου String και επιστρέφει το στοιχείο που είναι στην κορυφή της στοίβας, χωρίς να το αφαιρεί από αυτήν.

* **private String popOutOfStack()**

Η μέθοδος αυτή είναι Private, τύπου String και επιστρέφει το στοιχείο που είναι στην κορυφή της στοίβας, αφαιρώντας το από αυτήν.

* **private void pushInStack(String string)**

Η μέθοδος αυτή είναι Private, τύπου void και δέχεται μια παράμετρο τύπου string. Η παράμετρος αυτή εισάγεται στη κορυφή της στοίβας.

# Άσκηση Δ

…

# Άσκηση Ε

Εισαγωγή

Στη άσκηση αυτή μας ζητήθηκε να γράψουμε ένα πρόγραμμα FLEX το οποίο θα αναλύει προτάσεις της μορφής «Dinetai trigwno ABC». Έτσι λοιπόν στην προηγουμένη πρόταση θα πρέπει να λάβουμε ως έξοδο τα εξής:

***Dinetai: einai rhma***

***trigwno: einai gewmetrikh ontothta***

***ABC: einai onoma gewmetrikhs ontothtas***

Στο πρόγραμμα μας λοιπόν έχουμε υλοποιήσει τις περίπτωσης για την γωνιά, την ευθεία, το τρίγωνο, το τετράγωνο και το πεντάγωνο.

Επίσης, το πρόγραμμα είδη είναι δέχεται greeklish μπορεί να αναγνωρίζει της ακόλουθες παραλλαγές εκφράσεων:

* Για το «Dinetai» που είναι το ρήμα έχουμε την κανονική έκφραση

**(D|d)inetai** , ώστε να αναγνωρίζεται και το «dinetai» αλλά και το «Dinetai» , διότι βρίσκεται στην αρχή της πρότασης.

* Για το «euthia» που είναι γεωμετρική οντότητα έχουμε την κανονική έκφραση

**e(y|u)thia**, ώστε να αναγνωρίζεται και το «eythia» αλλά και το «euthia».

* Για το «trigwno» που είναι γεωμετρική οντότητα έχουμε την κανονική έκφραση **trig(w|o)no**, ώστε να αναγνωρίζεται και το «trigwno» αλλά και το «trigono».
* Για το «tetragwno» που είναι γεωμετρική οντότητα έχουμε την κανονική έκφραση **tetrag(w|o)no**, ώστε να αναγνωρίζεται και το «tetragwno» αλλά και το «tetragono».
* Για το «gwnia» που είναι γεωμετρική οντότητα έχουμε την κανονική έκφραση

**g(w|o)nia**, ώστε να αναγνωρίζεται και το «gwnia» αλλά και το «gonia».

* Για το «polygwno» που είναι γεωμετρική οντότητα έχουμε την κανονική έκφραση **pentag(w|o)no**, ώστε να αναγνωρίζεται και το «polygwno» αλλά και το «polygono».

Τέλος για τα 5 γράμματα της αλφάβητου (A, B, C, D, E) χρησιμοποιήσαμε την κανονική έκφραση [A-E]

Σε αυτό το σημείο να σημειώσουμε ότι έχει γίνει και η υλοποίηση του ερωτήματος Ε1 αλλά και του Ε2. Συνεπώς ο αναλυτής μας δεν θα αναγνώριση για παράδειγμα κάποια είσοδο της μορφής *«Dinetai trigwno AAB»* γιατι το όνομα της γεωμετρικής οντότητας περιέχει δυο φορές το γράμμα A. Επίσης δεν θα αναγνωρίσει για παράδειγμα την είσοδο *«Dinetai eythia AECD»* γιατι το όνομα της ευθείας πρέπει να αποτελείται μόνο από 2 γράμματα. Στη συνέχεια ακολουθών στιγμιότυπα από την εκτέλεση του προγράμματος, με διαφορά παραδείγματα.

**Εκτέλεση προγράμματος FLEX**

Το αρχείο που περιέχει το πρόγραμμα flex βρίσκεται στο φάκελο **exercise-5** και έχει όνομα **exercise5.l**

Για να δημιουργήσουμε το εκτελέσιμο αρχείο ακολουθούμε τα παρακάτω βήματα.

1. Ανοίγουμε το terminal και πάμε στο φάκελο **exercise-5** όπου περιέχεται το αρχείο **exercise5.l**
2. Γράφουμε την εντολή **flex exercise5.l**
3. Γράφουμε την εντολή **gcc lex.yy.c -lfl**
4. Εκτελούμε το αρχείο με την εντολή **./a.out**

**Στιγμιότυπα από την εκτέλεση του προγράμματος**