Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Σημειώσεις Μαθήματος

Εαρινού Εξαμήνου 2021-22

COMPILERS INTRO

Ονοματεπώνυμό Φοιτητή:
Κούμανης Βασίλειος

Καθηγητής:
Χρήστος Τρούσσας

Abstract

- Το παρών έγγραφο πρόκειται για τις σημειώσεις του μαθήματος ‘COMPILERS’ για την πλατφόρμα για την εργασία Ηλεκτρονικής Μάθησης. Θα περιέχει υλικό για κανονικές εκφράσεις, λεκτική ανάλυση με το εργαλείο FLEX και τη συντακτική ανάλυση.

Περιεχόμενα

Page:

Κανονικές Εκφράσεις 4

Λεκτική Ανάλυση 5

Συντακτική Ανάλυση 6-7

Task Μαθήματος 7

Κανονικές Εκφράσεις [Regex]

- Οι κανονικές εκφράσεις είναι ένας τρόπος αναγνώρισης συμβολοσειρών.
- Με τη χρήση συγκεκριμένων συμβόλων μπορούμε να αναγνωρίσουμε διάφορα patterns κειμένου. Τα σύμβολα αυτά είναι:

  - `[]` Λίστα αντικειμένων
  - **`[A-Z]`** Κεφαλαία γράμματα
  - **`[a-z]`** Πεζά γράμματα
  - `[0-9]` Αριθμός
  - | Λογικό **'η' (`OR`)**
  - **`*`** Το σύμβολο που ακολούθησε μπορεί να υπάρχει **μηδέν** (0) ή **περισσότερες** φορές
  - **`+`** Το σύμβολο που ακολούθησε μπορεί να υπάρχει **μια** (1) ή **περισσότερες** φορές
  - **`?`** προαιρετική ύπαρξη του συμβόλου δηλαδή **μηδέν** (0) ή **μια** (1) φορές
  - `x{n}` Δείχνει το πόσες φορές (`'n'`) πρόκειται να εμφανιστεί ο χαρακτήρας (`'x'`)
  - `x{n, }` Δείχνει τις ελάχιστες φορές (`'n, '`) πρόκειται να εμφανιστεί ο χαρακτήρας (`'x'`)
  - `x{n,m}` Ο χαρακτήρας (`'x'`) πρόκειται να εμφανιστεί από (`'n'`) έως (`'m'`) φορές

- Για παράδειγμα, το παρακάτω regex αναγνωρίζει όλα τα γράμματα (κεφαλαία και πεζά) όσες φορές υπάρχουν που μετά έχουν μόνο από 2 έως 3 αριθμούς.

  - **`([A-Z]|[a-z])+[0-9]{2,3}`**
    Για παράδειγμα, αναγνωρίζεται η παρακάτω συμβολοσειρά:

    " abcThisisAstring647 "

Σημείωση: Εμείς καλύψαμε μόνο τα βασικά των regex βεβαίως και υπάρχουν πολύ δυνατότερες λειτουργίες.

Λεκτική Ανάλυση

Αφού μάθαμε τι είναι τα regex και τις βασικές υλοποιήσεις τους ήρθε η ώρα να προχωρήσουμε στο επόμενο στάδιο τον Λεκτικό Αναλυτή.

- Ο **Λεκτικός Αναλυτής** πρόκειται για το πρώτο στάδιο ενός μεταγλωττιστή όπου με τη βοήθεια των regex **αναγνωρίζει χαρακτήρες ή και λέξεις**. Σκοπός του είναι αφού αναγνωρίσει τις λέξεις να τις ονομάσει και να τις στείλει στο επόμενο στάδιο, τον Συντακτικό Αναλυτή

- Ένα καλό εργαλείο που μπορούμε να χρησιμοποιήσουμε για την ανάπτυξη ενός ΛΑ είναι το **FLEX**. Το FLEX πρόκειται για έναν **wrapper** της γλώσσας C όπου μας βοηθάει στην **ανάπτυξη ενός ΛΑ** πάνω στη **γλώσσα C**

- Ένα πρόγραμμα FLEX αποτελείται από 3 μέλη: Τις **ρυθμίσεις**, τους **κανόνες** και τον **κώδικα σε γλώσσα C του χρήστη**.

    - Οι **ρυθμίσεις** ξεκινάνε με το σύμβολο `%` και πρόκειται για τις ρυθμίσεις που θέλουμε να γίνει η Λεκτική Ανάλυση. Στην περίπτωση μας χρησιμοποιούμε τις ρυθμίσεις : '`%option noyywrap`' οπού ενημερώνει τον FLEX να μην κληθεί η συνάρτηση `yywrap` η οποία χρησιμοποιείται για τη ΛΑ πολλαπλών αρχείων αλλά εμείς έχουμε μόνο ένα και `%x error` για να μπορέσουμε να ελέγξουμε τις περιπτώσεις που βρεθεί λάθος.
    - Οι **ρυθμίσεις** μπορούν επίσης να περιέχουν και κώδικα C για τα header files ή και για αρχικοποίηση μεταβλητών. Ο κώδικας C συμπεριλαμβάνετε ανάμεσα στους χαρακτήρες:
      `%{  #include <stdio.h> %}`
      Στις ρυθμίσεις μπορούν επίσης να αρχικοποιηθούν τα regex μας ως μεταβλητές

    - Το κομμάτι των **κανόνων** αποτελείται από τα regex και την ενέργεια που πρόκειται να γίνει αφού αναγνωριστεί το συγκεκριμένο regex που αναγνωρίσθηκε. Οι κανόνες βρίσκονται ανάμεσα στα σύμβολα `%% rules %%` πχ:
      ```
      %%
      {NUMBER} { printf("found a NUMBER"); return NUMBER; }
      ([A-Z]|[a-z])+
      %%
      ```
      Μπορούμε να χρησιμοποιήσουμε τη μεταβλητή που φτιάξαμε στο κομμάτι των ρυθμίσεων ή και το ίδιο το regex.

    - Τέλος, ο **κώδικας σε γλώσσα C του χρήστη** είναι προαιρετικός μιας και το πρόγραμμα μιας μπορεί να τρέξει με ή χωρίς συντακτικό αναλυτή. Εάν δεν υπάρχει ΣΑ τότε θα χρειαστεί να χρησιμοποιήσουμε τη συνάρτηση `yylex();` για τη Λεκτική Ανάλυση μίας γραμμής.

Συντακτική Ανάλυση

Ο Λεκτικός αναλυτής συνήθως πάει πακέτο με έναν Συντακτικό αναλυτή με σκοπό την πλήρη αναγνώριση μιας έκφρασης.

- Είπαμε προηγουμένως πως η δουλειά του Λεκτικού αναλυτή είναι αναγνωρίζει μια συμβολοσειρά με βάση τα regex και να εκτελεί κάποιες λειτουργίες για τη συγκεκριμένη συμβολοσειρά.
  Αν υπάρχει ΣΑ τότε ο ΛΑ θα πρέπει για το συγκεκριμένη συμβολοσειρά που αναγνωρίστηκε να επιστρέφει ένα **token.**
  Η δουλειά του ΣΑ είναι να αναγνωρίζει εκφράσεις οι οποίες αποτελούνται από μοτίβα από **tokens** όπου έχουμε ορίσει εμείς και να εκτελεί συγκεκριμένες λειτουργίες που ορίσαμε για κάθε έκφραση.

- Το εργαλείο που συνεργάζεται με το **FLEX** για τη δημιουργία **Συντακτικού Αναλυτή** πάλι με τη χρήση της γλώσσας C είναι το **BISON.** Το **BISON** πρόκειται πάλι για έναν **wrapper** της γλώσσας C όπου μας βοηθάει στην **ανάπτυξη ενός Συντακτικού Αναλυτή** πάνω στη **γλώσσα C**

- Η σύνταξη του BISON είναι παρόμοια με αυτή του FLEX και αποτελείται από 3 μέρη:
  Τις **ρυθμίσεις**, τους **κανόνες** και τον **κώδικα σε γλώσσα C του χρήστη**.

  - Οι **ρυθμίσεις** ξεκινάνε με το σύμβολο `%` και πρόκειται για τα header file σε κώδικα C, και τις μεταβλητές που θέλουμε.
    Υστέρα, θα πρέπει να αρχικοποιήσουμε όλα τα **token** που πρόκειται να στείλει το FLEX πχ: `%token NUMBER CHAR`.
    Τέλος, θα πρέπει να δώσουμε την αρχή της ΣΑ: '`%start program`' όπου 'program' μπορεί να είναι όποιο όνομα θέλουμε για την έκφραση μας.

  - Στο κομμάτι των **κανόνων** ορίζουμε τα μοτίβα των εκφράσεων και τις λειτουργίες που πρόκειται να εκτελεστούν αν αναγνωριστεί η συγκεκριμένη έκφραση. Οι εκφράσεις ξεκινάνε πάντα με τα όνομα που δώσαμε στο `%start` σε αυτήν την περίπτωση είναι '`program`' το οποίο είναι το όνομα της τελικής έκφρασης.
    Η κάθε έκφραση μπορεί να αποτελείτε από tokens από τον FLEX ή από άλλες 'μικρότερες' εκφράσεις. Για να καλύψουμε όσες περισσότερες καταστάσεις / μοτίβα μπορούμε μια καλή πρακτική είναι να σπάμε τις εκφράσεις σε μικρότερα κομμάτια εκφράσεων.
    Πχ: η αρχική έκφραση `addWord` μπορεί να αποτελείτε από άλλες δυο μικρότερες εκφράσεις τη '`add`' και τη '`word`' όπου '`add`' είναι της μορφής `[ number + number ]` (`number` είναι token από τον FLEX σε αυτή την περίπτωση) και '`word`'
    είναι `[ CHAR CHAR CHAR CHAR ]` (το ίδιο ισχύει και για το `CHAR`)
    Οπότε το BISON θα αναγνωρίσει το μοτίβο **`program addWord`** αν διαβάσει πχ:
    `1+2abcd`

    Σημείωση: Δε θα μιλήσουμε για τη στοίβα σε αυτό το κομμάτι.

- Τέλος, ο **κώδικας σε γλώσσα C του χρήστη** πρόκειται για το πεδίο που θα ελεγχτεί αν η Συντακτική ανάλυση ήταν επιτυχής μέσω της ειδικής συνάρτησης `yyparse();` όπου ξεκινάει τη Συντακτική ανάλυση στο `program` και επιστρέφει την τιμή 0 για επιτυχία και 1 για σφάλματα. Μπορούμε επίσης σε αυτό το κομμάτι να ορίσουμε να δέχεται ως είσοδο ένα αρχείο.

Task Μαθήματος

Να αναπτυχθεί κώδικας :

- FLEX για λεκτική ανάλυση που θα αναγνωρίζει τις συμβολοσειρές:
  - 'VasilisKoumanis12' & 'SuperMario641996' ως `USERNAME`
  - '?!pass1957632?%' & 'passwordsimple64' ως `PASSWORD`
  - 'example@string.com' ως `EMAIL`

- Συνδυάστε τον κώδικα FLEX που αναπτύξατε για το παραπάνω ερώτημα με το BISON για τη Συντακτική Ανάλυση που όταν θα αναγνωρίζει τα μοτίβα:
  - '`program USERNAME PASSWORD`' θα εμφανίζει (**`LOGIN`**)
  - '`program EMAIL USERNAME PASSWORD`' θα εμφανίζει (**`REGISTER`**)
  - '`program USERNAME`' θα εμφανίζει (**`MISSING PASSWORD`**)
  - '`program USERNAME EMAIL PASSWORD`' θα εμφανίζει (**`WRONG FORMAT`**)

- Ως bonus βαθμό να καλέσετε τη συνάρτηση `yyparse();` να δέχεται είσοδο ένα αρχείο από το τερματικό που να περιέχει τις παραπάνω συμβολοσειρές αντί να τις πληκτρολογήσετε εσείς, να μετράει και να εμφανίζει τις σωστές εκφράσεις που βρέθηκαν.