**Sveučilište u Zagrebu**

**Fakultet elektrotehnike i računarstva**

**Seminarski rad iz predmeta**

**„Prevođenje programskih jezika“**

Zadatak broj: 3007

**Zagreb, siječanj 2011.**

**Seminarski rad iz predmeta „Prevođenje programskih jezika“**

**Student:**

**Matični broj:**

**Tekst zadatka 3007:** Izgraditi program simulator leksičkog analizatora zasnovan na tablici prijelaza DKA prema algoritmu iz udžbenika (“Prevođenje programskih jezika”, str. 58). Ulaz za program simulator je skup proizvoljno zadanih regularnih izraza kojima se opisuju klase dozvoljenih leksičkih jedinki. Radi jednostavnosti, može se pretpostaviti da u jeziku zadanom regularnim izrazima nema nejednoznačnosti. Program simulator treba omogućiti izvođenje simulacije po koracima, te praćenje stanja kazaljki početak, završetak, posljednji i izraz.

1. Uvod
   1. Jezični procesor

Zadaća svakog jezičnog procesora (JP) je prevođenje programa. Program se prevodi iz jezika kojim je pisan, izvornog jezika, u jezik koji prihvaća i razumije računalo na kojem se konkretni jezični procesor izvodi. Taj se jezik, prihvatljiv računalu, naziva ciljnim jezikom. Prilikom prevođenja jezika jezični procesor prolazi dvije faze: analiza izvornog programa i sinteza ciljnog programa. Pod analizom programa podrazumijeva se leksicka, sintaksna i semantička analiza, dok se pod sintezu ubraja generiranje međukoda, strojno nezavisno optimiranje, generiranje strojnog programa, strojno zavisno optimiranje i priprema programa za izvođenje.

* 1. Leksička analiza

Leksička analiza je početni korak u radu jezičnog procesora. Leksički analizator čita tekst izvornog programa znak po znak, grupira znakove u leksičke jedinke, određuje im klasu te ih zapisuje u tablicu znakova. Ujedno leksički analizator iz izvornog teksta izbacuje sve nepotrebne znakove poput komentara, praznina, znakova tabulatora i sličnih. Klase leksičkih jedinki određene su pravilima koje najčešće zadajemo u obliku regularnih izraza. Svaka leksička mora zadovoljiti pravila klase prije no što joj se ista odredi.

1. Ostvarenje
   1. Definiranje problema

Na oko zadak je napraviti simulator leksičkog analizatora prema navedenom algoritmu iz udžbenika, no sljedeća rečenica „Ulaz u program je skup proizvljno zadanih regularnih izraza.“ nam znatno komplicira stvari. Ako nam ulaz u program definira način rada simulatora, proizvoljni ulaz znači da moramo napraviti generator koji će prilagoditi pravila na ulazu, regularne izraze, u pravila koja će koristiti sam simulator. Prilagođeni oblik pravila je skup stanja automata. Simulator je zasnovan na tablici prijelaza DKA. Inicijalno za potrebe leksičke analize generira se epsilon-NKA. Pojedini DKA označava jedan i samo jedan regularni izraz. Iz dodanog početnog stanja prelazi se u pojedne DKA. Naš zadatak je iz tog epsilon-NKA automata izgraditi DKA tablicu prijelaza. Prvo ćemo epsilon-NKA pretvoriti u običan NKA, a zatim NKA u DKA. Pretvaranje simulatora zasnovanog na tablici prijelaza epsilon-NKA u simulator zasnovan na tablici prijelaza DKA u konačnici daje brži leksički analizator.

* 1. Programsko ostvarenje

Programsko ostvarenje napisano je u programskom jeziku C#. Ulazne datoteke u sam program su:

* regularniIzrazi.txt – Sadrže proizvoljno zadane regularne izraze u obliku „nazivIzraza definicijaIzraza“.
* ulaz.txt – Sadrži tekst izvornog programa koji se treba analizirati.

Između naziva i definicije izraza mora biti točno jedan razmak. Podržani su izrazi kao sve kombinacije tri tipa zagrada: () obična zagrada, ($) zagrada sa epsilonom i ()\* zagrada sa kleenovim operatorom. Znak „$“ uznačava epsion. Rezervirani znakovi su (,),$ i |. Svi ostali znakovi mogu se nalaziti unutar zagrada gdje je znak | operator izbora ili. Datoteka ulaz.txt sadrži znakove za grupiranje u jednom redu. Razmaci su dopušteni.

Program se sastoji od sljedećih klasa:

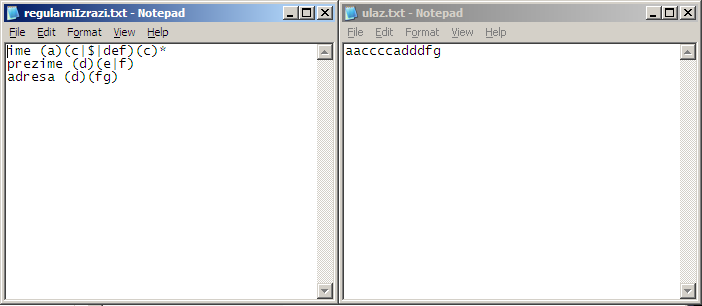
* MainClass – Ulazna točka u program. Učitava regularne izraze i poziva sve ostale klase.
* Parsiranje- Klasa koja obrađuje učitane regularne izraze i generira DKA automate za pojedine regularne izraze.
* Simulator – Epsilon-NKA pretvara u DKA generirajući DKA prijelaze i izvršava samu simulaciju.

Postupak generiranja DKA za pojedini regularni izraz je sljedeći. Prvo odredimo broj i tip zagrada u izrazu. Nakon toga odredimo koja zagrada nam daje prihvatljivo stanje. Primjer: za regularni izraz (a|b)(c|$)(d) imali bi tri zagrade, a treća je ta koja definira prihvatljivo stanje. Kada smo to odredili krećemo u generiranje prijelaza. Svaka zagrada u izrazu obrađuje se posebno sa za to napravljenom funkcijom. Prilikom generiranja prijelaza u posebnu varijablu dodaju se generirana prihvatljiva stanja koja su nam potrebna za samu simulaciju.

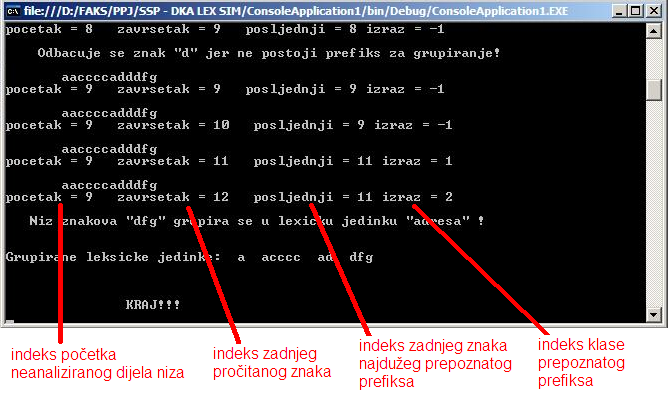
Preduvjet samoj simulaciji je da prethodni epsilon-NKA pretvorimo u DKA. Za pretvorbu odredimo ulaznu abecedu te stvorimo novu listu prijelaza koju će simulator koristiti. Smulator čita znakove iz ulazne datoteke, iz liste prijelza odabire odgovarajući prijelaz. Ukoliko prijelaz ne postoji niz znakova se grupira u leksičku jedniku ili se odbacuje krajnji lijevi znak ne analiziranog dijela niza ako nije pronađen prefiks za gupiranje.

* 1. Primjer izvođenja

U ulazne datoteke se upisuju željeni podatci:



Rezultat izvođenja programa:



1. Zaključak

Izgradnja simulatora leksičkog analizatora je u cijelosti bila zanimljiv studentski projekt. Izgradnja simulatora zasnovanog na tablici prijelaza DKA isprva se činila kao nepotrebno kompliciranje kad već postoji simulator zasnovan za tablici prijelaza epsilon-NKA. No prilikom ostvarenja pretvorba epsilon-NKA u DKA upće se nije pokazala teškom te je riješenje možda zgodnije za uporabu. Također takav simulator bi radio brže no onaj sa epsilon-NKA tablicom. Glavni problem u procesu ostvarenja bio je odrediti kako iz regularne definicije izgraditi DKA automat. Taj proces nije jednoznačno određen, stoga je ovo i najzanimljiviji dio projekta.