Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Mirko Jambrošić

Seminarski rad iz predmeta
Prevođenje programskih jezika

Zadatak broj: 2080

Zagreb, prosinac 2009.

Seminarski rad iz predmeta Prevođenje programskih jezika

Student: Mirko Jambrošić

Matični broj: 0036428589

Zadatak broj 2080 : Pomoću programa LEX ostvariti leksičku analizu programskog koda za jezik Haskell. Potrebno je podržati osnovne konstrukte jezika poput ulaznih i izlaznih naredbi, blokove, naredbe grananja, programske petlje te riješiti nejednoznačnosti koje se pojavljuju u leks. analizi. Izlaz iz programa su četiri tablice

Uvod

Leksička analiza je početni korak u radu jezičnoga procesora. Leksički analizator čita tekst izvornoga programa znak po znak, grupira znakove u leksičke jedinke i odeđuje im klasu u koje pripadaju. Ako program podržava komentare, također izbacuje komentare, nepotrebne simbole kao što su nepotrebne praznine, prazni redovi, tabulatori. Leksički analizator provjerava da li leksička jedinka zadovoljava pravila jezika.

LEX – Lexical Analyzer Generator je jedan od zastupljenijih programa koji se koriste kod izgradnje leksičkih analizatora. LEX učitava datoteku sa definicijama leksičkih pravila (*lex.l*), obrađuje ih i stvara datoteku sa kôdom u programskom jeziku *c* (*lex.yy.c*). Dotična datoteka se kopira i prevodi u jednom od C prevodioca, te je dobivena izvršna datoteka spremna za korištenje.

Izgled datoteke, koju učitava LEX je dana u nastavku.

```
%{
    Deklaracije
}%
    Regularne definicije
%%
    Pravila prevođenja
%%
    Pomoćne funkcije
```

Od gore navedenih pravila, neke su opcionalne, dok su neke neophodne. Najkraći mogući ispravni zadani izraz je: %%

Definicije su oblika: *ime definicija*
gdje definicija je u obliku regularnih izraza tipa: (primjer) *[a-zA-Z][a-zA-Z0-9]**

Pravila prevođenja su oblika: *uzorak akcija*
gdje uzorak opisuje koji niz ulaznih znakova treba odgovarati regularnom izrazu. Akcija se pise u istome redu.

Kod rješavanja nejjednoznačnosti koristimo se pretraživanjem lijevog konteksta. Način na koji se to rješava, te pravila pisanja pretraživanja lijevog konteksta nalaze se u nastavku.

```
%s MyState
%%
prvi {BEGIN MyState;}
...
<MyState>drugi {BEGIN 0;}
```

U gornjem primjeru programski dio *drugi* se izvodi tek ako se prije toga izveo programski odsjećak *prvi*.

Ostvarenje

Rješavanje zadatka svelo se na proučavanje oblika leksičke strukture programskog jezika Haskell. Haskell program je niz leksema odvojenih prazninama. Najduži niz znakova predstavlja leksemu. Komentari započinu sa dvije vodoravne crte -- te traju do kraja linije. Komentari također mogu započinjati vitičastom zagradom sa crticom {-, te trajati do prve zatvorene vitičaste zagrade sa crticom -}. Opetarori predstavljaju niz specijalnih simbola. Operatori koji počinju sa znakom dvotočke : su konstruktori, ostali su funkcije. Operatori se tretiraju kao identifikatori ako se stave u oble zagrade ( ). Identifikatori se tretiraju kao operatori ako se pišu unutar obrnutih navodnika ‘ ‘. U programskom jeziku Haskell identifikatori počinju velikim ili malim početnim znakom ili znakom donje crtice _ . Neimenovani identifikator predstavlja samo donja crtica _ nakon koje ne slijedi ništa

Izrazi, specijalni znakovi i operatori u programskom jeziku Haskell, tj KROS simboli su: , , , *let, in, if, then, else, case, of, do* \, ->,:: , :, [,], |, <-, >>, >>=, <<=, <<, List, *,/,+,-, =, ==.

Lambda izrazi počinju sa znakom lijeve kose crte \ te prima argumente *a1, a2, a3*,.. a vraca vrijednost izraza *e* izgledaju ovako: \ a1 a2 a3 …. -> e

Prema gornjim pravilima pišemo regularne izraze za LEX.

```
nista [" "\t\n]+

realne [0-9]+"."[0-9]+

cjelobrojne [0-9]+

string ("\""[^\"\n]*"\"")

idn ("_"[a-zA-Z]*[0-9]*)|([a-zA-Z]+[0-9]*)

kom ("--"[a-zA-Z]*[0-9]*)|("{-"[[a-zA-Z]*[0-9]*\n\t]*"-}")
```

Kad program shvati da su komentari posrijedi, ne poduzimaju se nikakve akcije, već se sve samo lijepo ignorira. U slučaju identifikatora, konstante ili ključne riječi poduzimaju se odgovarajuće akcije.

Za KROS elemente je napravljen popis istih. Kod pisanja nekih specijalnih znakova kao sto su lijeva kosa crta i slično potrebno je pisati \znak. Npr da bi programski bilo dobro ostvareno, za prepoznavanje lambda izraza koji je pisan u gornje obliku, lambdu predstavljamo kao „\\“ (bez navodnika)

Leksički analizator stvara tablicu uniformnih znakova. Za ubacivanje uniformnih znakova u dotičnu tablicu napisana je jedna funkcija nazvana *ubaci* koja ne vraća nikakav podatak, dok prima dva argumenta- sam uniformni znak i kazaljku na taj znak.

```
void ubaci_uniformni(string znak, int
kazaljka){
FILE * pFile;
pFile = fopen ("uniformni.txt","a");
if (pFile!=NULL)
{
fprintf(pFile, "%s %d\n", &znak[0], kazaljka);
fclose (pFile);
}
}
```

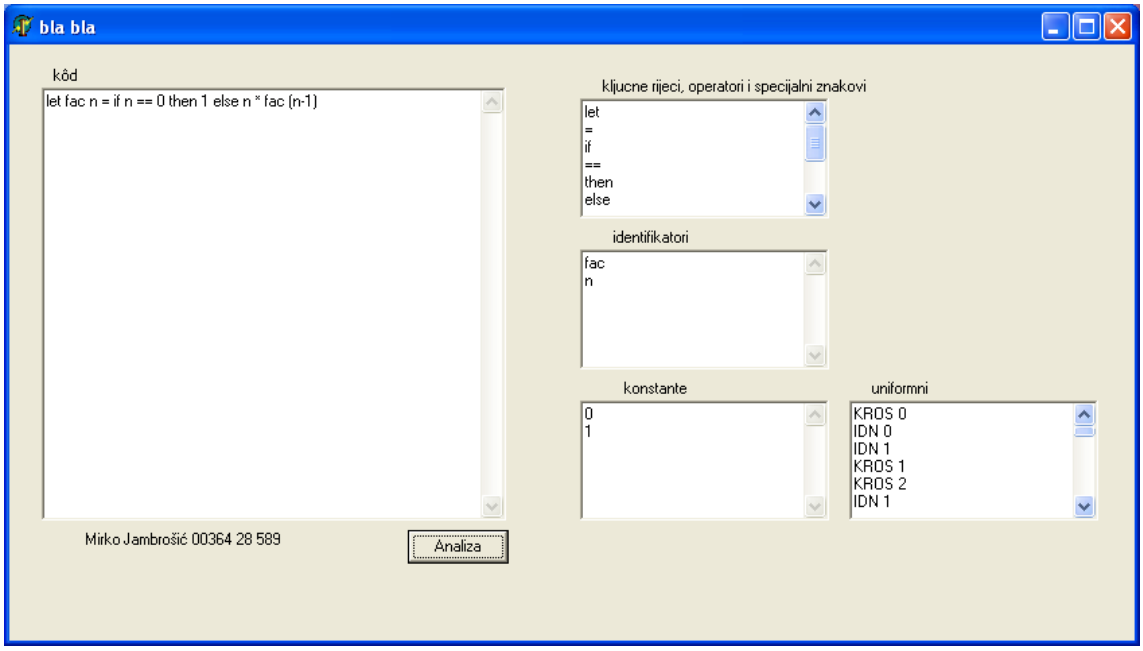
Koristimo još nekoliko funkcija za ubacivanje identifikatora i konstanti. Funkcije vraćaju kazaljku na mjesto gdje je spremljena dotična riječ. U sklopu te funkcije nalazi se „poziv još jednoj funkciji“ koja pretražuje datoteku identifikatra i konstanti, te ako se već dotična riječ nalazi u tim datotekama vraća samo kazaljku na njih i ne sprema ništa.

```
void ubaci_IDN(string znak){
  FILE * pFile;
  pFile = fopen("idn.txt","r+");
  int x=0;
  int y=-1;
  char z[1000];
  while (fscanf(pFile, "%s\n", &z)!=EOF)
  {
        if (strcmp(z, &znak[0])== 0)
        {
              y=x;
              break;
        }
        x++;
  }
  if (y==-1) //ubaci u tablicu taj znak
  {
        fprintf(pFile, "%s\n", &znak[0]);
  }
  fclose (pFile);
  ubaci_uniformni("IDN",x);
}
```

Gornja funkcija pretražuje datoteku sa identifikatorima te vraća mjesto na kojem se nalazi identifikator, ili -1 ako ne postoji u datoteci. Ako identifikator ne postoji u datoteci, ubacuje u datoteku, vraća redni broj na kojem se nalazi u datoteci te vraća u glavnu funkciju koja ubacije to u tablicu uniformnih znakova.

Iste funkcije su za ubacivanje konstanti te ključnih riječi, operatora i specijalnih simbola u KROS tablicu.

Napravljeno je grafičko sučelje u Delphiu programskom jeziku baziranom na Pascalu.



Programski je ostvareno na način da program pisan u Delphihu pokreće, pomoću Shell funkcija i procedura, izvršnu datoteku koju generira Microsoftov Visual C++, pričeka njezin završetak te prikazuje rezultate na formi.

```
Memo1.Lines.SaveToFile('ulaz.txt');
 ExecuteFile:='ssp_Beta2.exe';
  FillChar(SEInfo, SizeOf(SEInfo), 0) ;
  SEInfo.cbSize := SizeOf(TShellExecuteInfo) ;
  with SEInfo do begin
  fMask := SEE_MASK_NOCLOSEPROCESS;
  Wnd := Application.Handle;
  lpFile := PChar(ExecuteFile) ;
  nShow := SW_SHOWNORMAL;
  end;
  if ShellExecuteEx(@SEInfo) then begin
   repeat
    Application.ProcessMessages;
    GetExitCodeProcess(SEInfo.hProcess, ExitCode) ;
    until (ExitCode <> STILL_ACTIVE) or Application.Terminated;
    begin
     ShowMessage('leksicka analiza zavrsena') ;
     Memo2.Lines.LoadFromFile('kros.txt');
     Memo3.Lines.LoadFromFile('idn.txt');
     Memo4.Lines.LoadFromFile('kon.txt');
     Memo5.Lines.LoadFromFile('uniformni.txt');
     end;
     end
    else ShowMessage('greska kod pokretanja analizatora!') ;
```

Zaključak

Program LEX vrlo je jednostavno rješenje za izgradnju leksičkog analizatora. Najveći problem je pisanje regularnih izraza, te zbog neefikasnog sučelja dolazi do mnogo grešaka koje se teško ispravljaju. Prilikom rada sa LEXom, primjetio sam da radi vrlo dobro i pod Windows okruženjem, te nisam imao nikakvih većih problema sa istim.