

Rekurzivni spust

* Rekurzivni spust je metod parsiranja koji uključuje korištenje rekurzivnih poziva procedura i radi odosgo naniže. Može se primjeniti samo na LL(1) gramatike.

* Uopšteno govoreći, za svaki neterminal gramatičke pišemo funkciju, koja korištenjem simbola iz skupova izbora pravila koja s lijeve strane imaju taj neterminal bira pravila koja će se koristiti, a za neterminele koji se nalaze s desne strane tih odgovarajućih pravila poziva njihove funkcije, a terminale preko leksičkog analizatora upoređuje s onim što je na redu u ulaznom toku.

Zadaci:

① Napisati rekurzivni spust za gramatiku:

$$S \rightarrow AB \mid rB$$

$$A \rightarrow pSg \mid \epsilon$$

$$B \rightarrow qt.$$

VJEŽBE 12
DO ZADATKA 2

Rešenje: Zad1.c

Za datu gramatiku smo dokazali da je LL(1).

Kod:

```
#include <stdio.h> // par direktiva za uključenje C biblioteka u
#include <stdlib.h> // kojima se nalaze funkcije kao što su getchar();
int nailazeci; // deklaracija cijelobrojne promjenjive nailazeci, koja uzima vrijednosti iz skupa izbora za pravila sa istom lijevom stranom;
```

```
void greska()
{
    fprintf(stderr, "Greska!\n");
    system("pause");
    exit(1);
}
```

```
void uporedi (int xi)
{
    if (xi == nailazeci) nailazeci = getchar();
    else greska();
}
```

na funkciju getchar(), jer sve leksičke klase u stvari

// obrada greške;
exit(1) - obično označava neuspješno izvršenje programa-
prekid;
exit(0) - prekid uspješno
izvršenog programa; //

// upoređuje svoj
cijelobrojni argument
sa trenutnom
vrijednošću promjenjive
nailazeci i poziva
lexički analizator
(koji se ovde svodi

predstavljaju samo po jedan karakter. //

void s(), a(), b(); // deklaracija funkcija za neterminalne.
Kako u C-u funkcija mora biti makar deklarisana ako se poziva, a ove funkcije se uzajamno pozivaju, stavljemo deklaracije tih funkcija prije definicije prve od njih.

```
void s()
{
    if (nailazeci == 'r')
    {
        uporedi ('r');
        b();
    }
    else
    {
        a();
        b();
    }
}
```

```
void a()
{
    if (nailazeci == 'p')
    {
        uporedi ('p');
        s();
        uporedi ('g');
    }
    else; // obrada A → ε;
}
```

```
void b()
{
    if (nailazeci == 'g')
    {
        uporedi ('g');
        a();
    }
}
```

```
else greska(); // nailazeći mora imati vrijednost 'g' ili  
} se radi o grešci; //
```

```
void analiza()  
{ nailazeći = getchar();  
    s();  
    if(nailazeći != '\n')  
        greska();  
}
```

} // uzima jedan karakter sa ulaza i poziva funkciju koja odgovara startnom simbolu S, a nakon izlaska iz te funkcije funkcija analiza provjerava da li je došlo do kraja unosa - ovdje kraj unosa označavamo sa Enter, pa se tu trenutna vrijednost promjenjive nailazeći uporeduje sa "\n".

```
main()  
{ printf("Unesite nisku! \n");  
    analiza();  
    printf("Niska je sintaksno ispravna! \n");  
}
```

} Funkcija main() komunicira sa korisnikom, poziva funkciju analiza() i obavještava korisnika o rezultatu.

* Sada sa standardnog ulaza unosimo niske koje pripadaju jeziku:

kao npr. rg jer $S \Rightarrow rB \Rightarrow rgA \Rightarrow rg$

rgprgg jer $S \Rightarrow rB \Rightarrow rgA \Rightarrow rgPSg \Rightarrow rgprBg \Rightarrow rgprgAg \Rightarrow rgprgg$

i niske koje ne pripadaju jeziku kao npr. p, gg itd □

② Za datu gramatiku napisati rekursivni spust:

$\langle Izraz \rangle ::= \langle Sabirak \rangle \langle NastavakI \rangle ;$

$\langle NastavakI \rangle ::= "+" \langle Sabirak \rangle \langle NastavakI \rangle$
| "-" $\langle Sabirak \rangle \langle NastavakI \rangle$
| ε;

$\langle Sabirak \rangle ::= \langle Faktor \rangle \langle NastavakS \rangle ;$

$\langle NastavakS \rangle ::= "*" \langle Faktor \rangle \langle NastavakS \rangle$
| "/" $\langle Faktor \rangle \langle NastavakS \rangle$
| ε;

$\langle \text{faktor} \rangle ::= "+" \langle \text{faktor} \rangle$
 | " $-$ " $\langle \text{faktor} \rangle$
 | $\langle \text{prost faktor} \rangle;$

$\langle \text{prost faktor} \rangle ::= \text{NCEO} \mid \text{IDENT} \mid "(" \langle \text{izraz} \rangle ")";$

Rešenje: zad2.c

Ova gramatika opisuje jezik aritmetičkih izraza, koji kao operacije uključuju sabiranje, oduzimanje, množenje, dijeljenje, kao i unarni plus i unarni minus, a kao argumente cijele brojeve (NCEO) i identifikatore (koji se sastoje iz slova i cifara, a počinju slovom - IDENT).

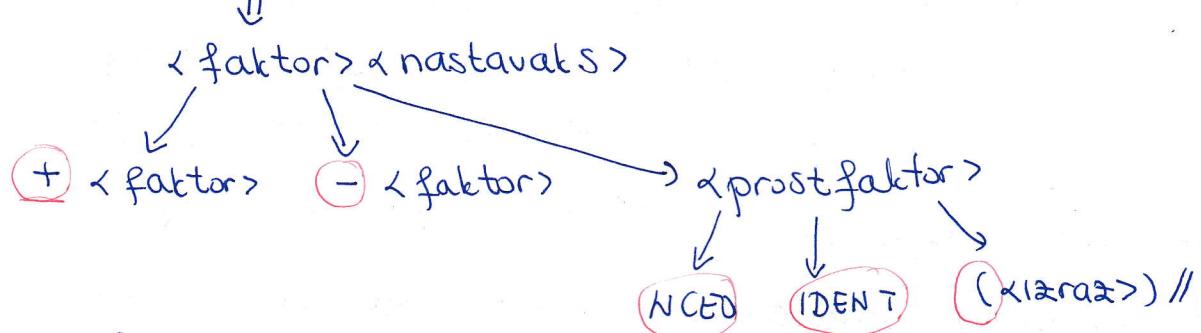
Proujerimo da li je data gramatika LL(1), računanjem skupova izbora.

$T = \{ +, -, *, /, \text{NCEO}, \text{IDENT}, (,) \}$ // skup terminala

$N = \{ \langle \text{izraz} \rangle, \langle \text{sabirak} \rangle, \langle \text{nastavakI} \rangle, \langle \text{faktor} \rangle, \langle \text{nastavakS} \rangle, \langle \text{prost faktor} \rangle \}$

$\text{pri} \langle \text{izraz} \rangle = \{ +, -, \text{NCEO}, \text{IDENT}, (\}$

// jer $\langle \text{izraz} \rangle ::= \langle \text{sabirak} \rangle \langle \text{nastavakI} \rangle$



$\text{pri} \langle \text{sabirak} \rangle = \{ +, -, \text{NCEO}, \text{IDENT}, (\}$

$\text{pri} \langle \text{nastavakI} \rangle = \{ +, -, \epsilon \}$

$\text{pri} \langle \text{faktor} \rangle = \{ +, -, \text{NCEO}, \text{IDENT}, (\}$

$\text{pri} \langle \text{nastavakS} \rangle = \{ *, /, \epsilon \}$

$\text{pri} \langle \text{prost faktor} \rangle = \{ \text{NCEO}, \text{IDENT}, (\}$

$\text{pri} (x) = \{ xc \} \quad \text{za } xc \in T;$

• sled($\langle \text{izraz} \rangle$) = {+, -}

// marker kraja + jer $\langle \text{izraz} \rangle$ je startni simbol;

) zbrog $\langle \text{prostfaktor} \rangle \rightarrow "(\langle \text{izraz} \rangle)"$; //

• sled($\langle \text{sabirak} \rangle$) = {+, -} \cup sled($\langle \text{nastavak} \rangle$) \cup sled($\langle \text{izraz} \rangle$)
) sled($\langle \text{izraz} \rangle$) = {+, -, +, -}

// +, - zbrog $\langle \text{izraz} \rangle ::= \langle \text{sabirak} \rangle \langle \text{nastavak} \rangle$ pa prvi($\langle \text{nastavak} \rangle$) \subseteq sled($\langle \text{sabirak} \rangle$); sled($\langle \text{izraz} \rangle$ jer $\langle \text{nastavak} \rangle \rightarrow \epsilon$ pa onda ostaje $\langle \text{izraz} \rangle ::= \langle \text{sabirak} \rangle$;

• sled($\langle \text{nastavak} \rangle$) = {+, -}

// jer $\langle \text{izraz} \rangle ::= \langle \text{sabirak} \rangle \langle \text{nastavak} \rangle$ pa je sled($\langle \text{izraz} \rangle$) \subseteq sled($\langle \text{nastavak} \rangle$) //

• sled($\langle \text{faktor} \rangle$) = {*}, / \cup sled($\langle \text{sabirak} \rangle$) = {*}, /, +, -, +, -}

// *, / prvi($\langle \text{nastavak} \rangle$) \subseteq sled($\langle \text{faktor} \rangle$);

$\langle \text{nastavak} \rangle \rightarrow \epsilon \Rightarrow \langle \text{sabirak} \rangle ::= \langle \text{faktor} \rangle \Rightarrow$ sled($\langle \text{sabirak} \rangle$) \subseteq

\subseteq sled($\langle \text{faktor} \rangle$) //

• sled($\langle \text{nastavak} \rangle$) = sled($\langle \text{sabirak} \rangle$) = {+, -, +, -}

// zbrog $\langle \text{sabirak} \rangle ::= \langle \text{faktor} \rangle \langle \text{nastavak} \rangle$ //

• sled($\langle \text{prostfaktor} \rangle$) = sled($\langle \text{faktor} \rangle$) = {*}, /, +, -, +, -

// zbrog $\langle \text{faktor} \rangle ::= \langle \text{prostfaktor} \rangle$

izbor ($\langle \text{izraz} \rangle ::= \langle \text{sabirak} \rangle \langle \text{nastavak} \rangle$) = prvi($\langle \text{sabirak} \rangle$) =
 = {+, -, NCE0, IDENT, ()}

izbor ($\langle \text{nastavak} \rangle ::= "+" \langle \text{sabirak} \rangle \langle \text{nastavak} \rangle$) = {+}

izbor ($\langle \text{nastavak} \rangle ::= "-" \langle \text{sabirak} \rangle \langle \text{nastavak} \rangle$) = {-}

izbor ($\langle \text{nastavak} \rangle ::= \epsilon$) = sled($\langle \text{nastavak} \rangle$) = {+, -}

$\text{Izbor } (\langle \text{faktor} \rangle \langle \text{nastavak} \rangle) ::= \text{prvi } (\langle \text{faktor} \rangle) =$
 $= \{ +, -, \text{NCEO}, \text{IDENT} \}$

$\text{Izbor } (\langle \text{nastavak} \rangle) ::= "*" \langle \text{faktor} \rangle \langle \text{nastavak} \rangle = \{ * \}$

$\text{Izbor } (\langle \text{nastavak} \rangle) ::= "/" \langle \text{faktor} \rangle \langle \text{nastavak} \rangle = \{ / \}$

$\text{Izbor } (\langle \text{nastavak} \rangle) ::= \varepsilon) = \text{sled } (\langle \text{nastavak} \rangle = \{ +, -, +, \})$

$\text{Izbor } (\langle \text{faktor} \rangle) ::= "+" \langle \text{faktor} \rangle = \{ + \}$

$\text{Izbor } (\langle \text{faktor} \rangle) ::= "-" \langle \text{faktor} \rangle = \{ - \}$

$\text{Izbor } (\langle \text{faktor} \rangle) ::= \langle \text{prost faktor} \rangle = \text{prvi } (\langle \text{prost faktor} \rangle) =$
 $= \{ \text{NCEO}, \text{IDENT}, (\}$

$\text{Izbor } (\langle \text{prost faktor} \rangle) ::= \text{NCEO} = \{ \text{NCEO} \}$

$\text{Izbor } (\langle \text{prost faktor} \rangle) ::= \text{IDENT} = \{ \text{IDENT} \}$

$\text{Izbor } (\langle \text{prost faktor} \rangle) ::= "(" \langle \text{izraz} \rangle ")" = \{ (\}$

Kako su skupovi izbora za pravila sa istom lijevom stranom disjunktni data gramatika je LL(1).

* Za dulje leksičke klase, cijele brojeve i identifikatore, trebaju nam leksički analizator. Njega ćemo napraviti pomoći FLEX-a.

U zagлавju ćemo učaćuti tab. h koji izgleda ovako:

```
#define NCEO 25f  
#define IDENT 258
```

Definisane su dulje leksičke klase, NCEO i IDENT i pripojeni su im brojevi 25f i 258. To je zbog toga što leksički analizator mora parseru vratiti cjelobrojan rezultat - po defaultu, kad prepozna karaktere iz ASCII skupa, leksički analizator vrati brojeve koji im odgovaraju u ASCII skupu, tj. brojeve od 1 do 256, pa ćemo novim leksičkim klasama dodjeljivati brojeve od 25f, i kad leksički analizator vrati taj broj, parser će dobiti informaciju da je prepoznata ta leksička klasa.

* Specifikacija za FLEX pomoću koje se dobija leksički analizator koji prepoznaje identifikatore i cijele brojeve:

```
%option nowrap
```

```
%{
```

```
#include "tab.h" (putanja)
```

```
%}
```

```
slово [A-Za-z]
```

```
cifra [0-9]
```

```
bjel [\t\n]
```

```
ident {slovo}({slovo}|{cifra})*
```

```
ncoo {cifra}+
```

```
praz {bjel}+
```

```
%%
```

```
{ident} {return IDENT;}
```

```
{ncoo} {return NCOO;}
```

```
{praz} ;
```

```
. {return yytext[0];}
```

* Obratite pažnju da je treća sekcija programa, u ovom opisu prazna (tj. nema je!), za razliku od opisa za FLEX koji smo ranije radili. To je zato što će funkcije main() i yylex(), koje smo šablonski uključivali u tu sekciju, biti pozvane iz parsera, tako da nema potrebe da ih tu navodimo.

* Parser će očekivati da se leksički analizator nađe u datoteci Lex.yylex.c pa zbog toga moramo prvo da fleksujemo gore navedeni opis za FLEX. Odnosno, dav izvršimo naredbu:

```
flex pr2.L
```

* Tako smo dobili leksički analizator. Naravno, on sada ne može da se koristi sam sebe (jer nema main()), već treba napraviti i parser koji će ga pozivati.

Parser napisan metodom rekurzivnog spusta:

```
#include "Lex.yg.c" (putanja) + preimenovati
#include <stdio.h>
#include <stdlib.h>

void greska()
{ fprintf(stderr, "Greska!\n");
  exit(1);
}

int nailazeci;

void uporedi (int xc)
{ if (xc == nailazeci) nailazeci = yylex();
  else greska();
}

void Izraz(), Sabirak(), Nastavak(), Faktor(), NastavakS(),
prostf();

void Izraz()
{
  Sabirak();
  Nastavak();
}

void Nastavak()
{
  if (nailazeci == '+')
  {
    Uporedi('+');
    Sabirak();
    Nastavak();
  }
  else if (nailazeci == '-')
  {
    Uporedi('-');
    Sabirak();
    Nastavak();
  }
}
```

```
else ;
```

```
}
```

```
void sabirak()
```

```
{ faktor();
```

```
nastavaks();
```

```
}
```

```
void nastavaks()
```

```
{ if (nailazeci == '*')
```

```
{ uporedi('*');
```

```
faktor();
```

```
nastavaks();
```

```
}
```

```
else if (nailazeci == '/')
```

```
{ uporedi('/');
```

```
faktor();
```

```
nastavaks();
```

```
}
```

```
else ;
```

```
}
```

```
void faktor()
```

```
{ if (nailazeci == '+')
```

```
{ uporedi('+');
```

```
faktor();
```

```
}
```

```
else if (nailazeci == '-')
```

```
{ uporedi('-');
```

```
faktor();
```

```
}
```

```
else prostf();
```

```
}
```

```
void prostf()
```

```
{ if (nailazeci == NCEO) uporedi (NCEO);  
    else if (nailazeci == IDENT) uporedi (IDENT);  
    else if (nailazeci == '(')  
    { uporedi ('(');  
        Izraz();  
        uporedi (')');  
    }  
    else greska();  
}
```

```
void analiza()
```

```
{ nailazeci = yylex();  
    Izraz();  
    if (nailazeci != 0) greska();  
}
```

```
main()
```

```
{ printf ("Unesite izraz \n");  
    analiza();  
    printf ("Izraz je sintaksno ispravan!\n");  
}
```

* Primjetimo da su razlike u odnosu na prethodni primjer minimalne - moramo uključiti datoteku u kojoj se nalazi leksički analizator, a u funkcijama uporedi i analiza umjesto funkcije getchar koristi se funkcija yylex().

$$\text{Prvihata: } 23 + 5x^2 \cdot 5 \\ x \dots \quad (x+5) \cdot 2 \quad ; \quad 23 \cdot x^5 - 11$$

17 je kraj

③ Napisati rekursivni spust za gramatiku:

$\langle \text{naredba} \rangle ::= \langle \text{while-naredba} \rangle$
 | $\langle \text{begin-end-naredba} \rangle$
 | $\langle \text{naredba-dodele} \rangle$

$\langle \text{while-naredba} \rangle ::= \text{while } \langle \text{iskaz} \rangle \text{ do } \langle \text{naredba} \rangle ;$

$\langle \text{begin-end-naredba} \rangle ::= \text{begin } \langle \text{naredba} \rangle ; [\langle \text{naredba} \rangle ;]^* \text{ end}$

$\langle \text{naredba-dodele} \rangle ::= \langle \text{promjenjiva} \rangle := \langle \text{izraz} \rangle$

$\langle \text{iskaz} \rangle ::= \langle \text{izraz} \rangle = \langle \text{izraz} \rangle$

$\langle \text{promjenjiva} \rangle ::= \langle \text{slovo} \rangle [\langle \text{slovo} \rangle | \langle \text{cifra} \rangle]^*$

$\langle \text{izraz} \rangle ::= \langle \text{pozicifra} \rangle [\text{cifra}]^*$

$\langle \text{pozicifra} \rangle ::= 1|2|3|4|5|6|7|8|9$

$\langle \text{cifra} \rangle ::= \langle \text{pozifra} \rangle | 0$

$\langle \text{slovo} \rangle ::= a|...|z|A|...|Z$

Rešenje:

Proujerimo da li je data gramatika LL(1).

$T = \{ \text{while}, \text{do}, ;, \text{begin}, \text{end}, a, \dots, z, A, \dots, Z, 0, \dots, 9 \}$

$N = \{ \langle \text{naredba} \rangle, \langle \text{while-naredba} \rangle, \langle \text{begin-end-naredba} \rangle, \langle \text{iskaz} \rangle,$
 $\langle \text{naredba-dodele} \rangle, \langle \text{promjenjiva} \rangle, \langle \text{izraz} \rangle, \langle \text{slovo} \rangle, \langle \text{pozicifra} \rangle,$
 $\langle \text{cifra} \rangle \}$

$\text{prvi}(\alpha) = \{ \alpha \} \quad \forall \alpha \in T$

$\text{prvi}(\langle \text{naredba} \rangle) = \{ \text{while}, \text{begin}, a, \dots, z, A, \dots, Z \}$

$\text{prvi}(\langle \text{while-naredba} \rangle) = \{ \text{while} \}$

$\text{prvi}(\langle \text{begin-end-naredba} \rangle) = \{ \text{begin} \}$

$\text{prvi}(\langle \text{naredba-dodele} \rangle) = \{ a, \dots, z, A, \dots, Z \}$

$\text{prvi}(\langle \text{iskaz} \rangle) = \{ 1, 2, \dots, 9 \}$

$\text{prvi}(\langle \text{promjenjiva} \rangle) = \{ a, \dots, z, A, \dots, Z \}$

$\text{prvi}(\langle \text{izraz} \rangle) = \{ 1, 2, \dots, 9 \}$

$\text{prvi}(\langle \text{slovo} \rangle) = \{ a, \dots, z, A, \dots, Z \}$

$\text{prvi}(\langle \text{pozicifra} \rangle) = \{ 1, \dots, 9 \}$

$\text{prvi}(\langle \text{cifra} \rangle) = \{ 0, 1, 2, \dots, 9 \}$

$\text{sled}(\langle \text{naredba} \rangle) = \{ +, ; \}$

$\text{sled}(\langle \text{while_naredba} \rangle) = \{ +, ; \}$

$\text{sled}(\langle \text{begin_end_naredba} \rangle) = \{ +, ; \}$

$\text{sled}(\langle \text{naredba_dodele} \rangle) = \{ +, ; \}$

$\text{sled}(\langle \text{iskaz} \rangle) = \{ \text{do} \}$

$\text{sled}(\langle \text{promjenjiva} \rangle) = \{ := \}$

$\text{sled}(\langle \text{izraz} \rangle) = \{ =, +, ; \text{do} \}$

// +, ; zbog $\langle \text{naredba_dodele} \rangle :: = \langle \text{promjenjiva} \rangle := \langle \text{izraz} \rangle$

pa $\text{sled}(\langle \text{nareda_dodele} \rangle) \subseteq \text{sled}(\langle \text{izraz} \rangle)$;

do zbog $\langle \text{iskaz} \rangle :: = \langle \text{izraz} \rangle = \langle \text{izraz} \rangle ; \text{do}$ //

$\text{sled}(\langle \text{slово} \rangle) = \{ a, \dots, z, A, \dots, Z, 0, 1, 2, \dots, 9, := \}$

// := zbog $\langle \text{promjenjiva} \rangle :: = \langle \text{slovo} \rangle \underbrace{[\langle \text{slovo} \rangle | \langle \text{cifra} \rangle]}_e^*$

pa $\text{sled}(\langle \text{promjenjiva} \rangle) \subseteq \text{sled}(\langle \text{slovo} \rangle) //$

$\text{sled}(\langle \text{cifra} \rangle) = \{ a, \dots, z, A, \dots, Z, 0, 1, 2, \dots, 9, :=, =, +, ;, \text{do} \}$

// $\text{sled}(\langle \text{izraz} \rangle) \subseteq \text{sled}(\langle \text{cifra} \rangle)$ //

$\text{sled}(\langle \text{pozicifra} \rangle) = \{ 0, 1, 2, \dots, 9, =, +, ;, \text{do} \}$

// $\text{sled}(\langle \text{izraz} \rangle) \subseteq \text{sled}(\langle \text{pozicifra} \rangle)$ //

=====

$\text{izbor}(\langle \text{naredba} \rangle :: = \langle \text{while_naredba} \rangle) = \text{prvi}(\langle \text{while_naredba} \rangle) = \{ \text{while} \}$

$\text{izbor}(\langle \text{naredba} \rangle :: = \langle \text{begin_end_naredba} \rangle) = \{ \text{begin} \}$

$\text{izbor}(\langle \text{naredba} \rangle :: = \langle \text{naredba_dodele} \rangle) = \text{prvi}(\langle \text{naredba_dodele} \rangle) = \{ a, \dots, z, A, \dots, Z \}$

=====

$\text{izbor}(\langle \text{cifra} \rangle :: = \langle \text{pozicifra} \rangle) = \{ 1, 2, \dots, 9 \}$

$\text{izbor}(\langle \text{cifra} \rangle :: = 0) = \{ 0 \}$

=====

Dodata gramatika jeste LL(1) jer su skupovi izbora za pravila sa istom lijevom stranom disjunktni.

Opis za FLEX koji daje odgovarajući leksički analizator:
(prije toga "primer1.h"):

```
#define WHILE 25f
#define DO 258
#define BEG 259
#define END 260
#define SEMI 261
#define PROM 262
#define BROJ 263
#define EQ 264
#define DODELA 265
=====)
%option noyywrap
int red = 1;
%{
#include "primer1.h" (putanja)
%}
ID [A-zA-zA-Z][A-Za-zA-Z0-9]*
NUM [1-9][0-9]*
% %
WHILE|while return WHILE;
DO|do return DO;
BEGIN|begin return BEG;
END|end return END;
";" return SEMI;
{ID} return PROM;
{NUM} return BROJ;
"=" return EQ;
":=" return DODELA;
\n printf ("Zavrsena analiza reda %d.\n", red++);
/* preskoči beLine */
. printf ("Nepoznat simbol u redu %d.\n", red);
% %
```

Program za rekurzivni spust:

```
#include <stdio.h>
#include <stdlib.h>
#include "primer1.h" } putanje
#include "Lex.yy.c"

void iskaz(void);
void izraz(void);

extern int red; // je leksičkog analizatora
extern char *yytext; // -1-
int ukupno=0; // brojimo greske
int nailazeći;

#define uporedi(x) (x==nailazeći) // vrati true ili false
#define citaj() nailazeći=yylex()
#define greska(x) { printf("Greska u redu %d: ", red),
                  printf(x),
                  printf(" umjesto lekseme %s\n", yytext);
                  ukupno++;
}

void naredba()
{
    if (uporedi(WHILE))
    {
        citaj();
        iskaz();
        if (!uporedi(DO))
            greska(" očekivano DO ");
        citaj();
        naredba();
        if (!uporedi(SEMI))
            greska(" očekivana ';' ");
        citaj();
    }
    return;
}
```

```
else if (uporedi (BEG))
{
    citaj();
    do
    {
        naredba();
        if (! uporedi (SEMI))
            greska ("Ocekivana ';'"));
        citaj();
    } while (! uporedi (END));
    citaj();
    return;
}

else if (uporedi (PROM))
{
    citaj();
    if (! uporedi (DODELA))
        greska ("Ocekivano ':=' ");
    citaj();
    izraz();
    return;
}

else
    greska ("Nepoznata naredba");

}

void izraz()
{
    if (! uporedi (BROJ))
        greska ("Ocekivan broj");
    citaj();
}
```

```

void iskaz()
{
    izraz();
    if (!uporedi (EQ))
        greska (" ocekivano '=' ");
    citaj();
    izraz();
}

```

DRS4 - Konstrukcija
kompilatora

Vježbe 12
21.05.2013

main (argc, argv)

može i
bez argv

```

// int argc;
// char **argv;
{
    ++argv, --argc;
    if (argc > 0)
        yyin = fopen (argv [0], "r");
    else
        yyin = stdin;
    utaj();
    naredba();
    if (ukupno)
        printf (" In U unetom fragmetu koda postoji
            %i gresaka! \n", ukupno);
    else
        printf (" In Nema sintaksnih gresaka u unetom fragmetu
            koda" );
    }

```

0 = FALSE
1, 2, ... = TRUE

Prihvata: while $23 = 23$ do $x := 14;$
begin $y := 14;$ $x := 15;$ end
 $x := 23;$

□