

## Zadatak 2:

Napisati sintaksnii analizator koji prepoznauje i računa aritmetičke izraze.

### Resenje:

\* Opis za FLEX:

%option noyywrap

cijeli [0-9]+

%%

```
{cijeli} {yyval = atoi(yytext); return BROJ;}
```

```
[ It] ;
```

```
. return yytext EOF;
```

VJEŽBE 15

Kao što smo već vidjeli promjenljiva yytext čuva poslednju prepoznatu nisku - u ovom slučaju prepoznatu nisku koja predstavlja cijeli broj a C-ovska funkcija atoi (int) pretvara nisku u cijeli broj. Bjeline ignorisemo, a tačka predstavlja default - u slučaju da se prepozna bilo šta što nije ni cifra ni bjelina, leksički analizator će to (prvi element niske yytext) vratiti parseru.

\* Generišemo leksički analizator, tako što fleksujemo:

flex zad2.l

\* Opis za YACC:

%token BROJ

%token left '+' '-'

%token left '\*' '/'

%token left UMINUS

%%

S: izraz

```
{ printf ("%d\n", $1); }
```

izraz: izraz '+' izraz

```
{ $$ = $1 + $3; }
```

| izraz '-' izraz

```
{ $$ = $1 - $3; }
```

```

| izraz '*' izraz
{ $$ = $1 * $3; }

| izraz '/' izraz
{ if ($3 == 0) yyerror ("Djeljenje sa nulom");
  else $$ = $1 / $3; }

| '-' izraz %prec UMINUS
{ $$ = - $2; }

| '(' izraz ')'
{ $$ = $2; }

| BROJ;

```

```

%{
#include "putanja\Lex.yg.c"
yyerror (char *s)
{ printf ("%s\n", s);
}

main()
{ return yyparse ();
}

```

Ovdje vidimo ilustraciju onoga o što smo razmatrali u prethodnom tekstu - s jedne strane pitanje prioriteta i asocijativnosti, a s druge pitanja akcija.

\* Zatim, "jakujemo" opis za YACC, kako bismo realizovali parser:

bison zad2.y

Primjetite da nema izvještaja o broju konfliktata u gramatici. To je posljedica toga što smo ih eliminirali eksplicitnim unosnjem prioriteta i asocijativnosti.

Npr. za ulaz  $5 * 56 - 2$  vraca 278



### ✓ Zadatak 3:

Prestpostavimo da se kompleksnost aritmetičkih izraza definise formulom:

$$A + 2B + 3C$$

gdje je A broj binarnih operatora sabiranja i oduzimanja, B broj unarnih operatora plus i minus, a C broj operatora množenja i dijeljenja u izrazu.

Napisati sintakson analizator koji računa kompleksnost aritmetičkih izraza.

### Rešenje:

U odnosu na prethodni primjer, razlike su samo u akcijama, koje ovde ne računaju vrijednost izraza, nego njegovu kompleksnost, prema gore navedenom pravilu.

Npr. Za izraz  $-25 + 3 * 8 - 4$

$$A = 2 \quad B = 1 \quad C = 1$$

$$A + 2B + 3C = 2 + 2 + 3 = 7$$

\* U opisu za FLEX su uključeni i identifikatori, tako da smo definisali leksičku klasu i za njih. U nju smo ubacili i brojeve, jer se ne traži računanje njihove vrijednosti, već samo obavještenje da su prepoznati u ulaznom toku. Tako za identifikatore i brojeve imamo jednu leksičku klasu.

```
q0 option noyywrap
```

```
prom [a-zA-Z_]+
```

```
bjelina [ \t\n]
```

```
bjeline {bjelina}+
```

```
q0 q0
```

```
{bjeline} ;
```

```
{prom} {return PROM;}
```

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

Sačuvamo kao: zad3.l, zatim "fleksujemo" sa:  
flex zad3.l

Opis za YACC:

%token PROM

%left '+' '-'

%left '\*' '/'

%%

S : Izraz

{ printf (" %d \n", \$1); } ;

Izraz : Izraz '+' Izraz

{ \$\$ = \$1 + \$3 + 1; }

| Izraz '-' Izraz

{ \$\$ = \$1 + \$3 + 1; }

| Izraz '\*' Izraz

{ \$\$ = \$1 + \$3 + 3; }

| Izraz '/' Izraz

{ \$\$ = \$1 + \$3 + 3; }

| '-' Izraz

{ \$\$ = \$2 + 2; }

| '+' Izraz

{ \$\$ = \$2 + 2; }

| '(' Izraz ')'

{ \$\$ = \$2; }

| PROM

{ \$\$ = 0; }

%%

#include " putanja\lex.yg.c"

yyerror (char \*s)

{ printf ("%s\n", s); }

```
main()
{
    return yyparse();
}
```

## Zadatak 4:

Napisati YACC specifikaciju za kalkulator koji ima 26 registara označenih slovima od a do z, koji prihvata aritmetičke izraze koje čine operatori +, -, \*, /, % (dijeljenje po modulu), & (bitsko i), | (bitsko ili) i operator dodjela. Ako je izraz na najvišem nivou dodjela, vrijednost se ne štampa, inače se štampa ako cijeli broj počinje sa 0 smatra se oktalnim, inače decimalnim. (dekadnim)

### Rješenje:

Opis za FLEX:

```
%option noyywrap
slovo [a-zA-Z]
cifra [0-9]           bijelina [\t]+
%/%
{ slovo } { yyval=yytext[0]-'a';
             return SLOVO; }

{ cifra } { yyval=yytext[0]-'0';
             return CIFRA; }

{ bijelina }
{ return (yytext[0]); }
```

Ovo fleksujemo sa: flex zad4.l

Opis za YACC:

```
%{
#include <ctype.h>
int registri[26];
int baza;
%}
%start List
%token CIFRA SLOVO
```

% left '1'  
% left '2'  
% left '+' '-'  
% left '\*' '/'  
% left UMINUS

% %

list :

| <sup>2</sup> <sub>1</sub> list iskaz 'ln'  
| list / error / 'Xn'  
| ~~if yerror;~~ }  
| <sub>2</sub>

iskaz : izraz

{ printf ("%od\n", \$1); }  
| SLOVO '=' izraz  
{ registri [\$1] = \$3; }

izraz : '(' izraz ')'

{ \$\$ = \$2; }

| izraz '+' izraz

{ \$\$ = \$1 + \$3; }

| izraz '-' izraz

{ \$\$ = \$1 - \$3; }

| izraz '\*' izraz

{ \$\$ = \$1 \* \$3; }

| izraz '/' izraz

{ \$\$ = \$1 / \$3; }

| izraz '%' izraz

{ \$\$ = \$1 % \$3; }

| Izraz ' & ' Izraz  
{ \$\$ = \$1 & \$3; }

| Izraz ' | ' Izraz  
{ \$\$ = \$3 | \$1; }

| ' - ' Izraz % prec UMINUS  
{ \$\$ = -\$2; }

| SLOVO

{ \$\$ = registri[\$1]; }

| broj;

broj : CIFRA

{ \$\$ = \$1; baza = (\$1 == 0)? 8 : 10; }

| broj CIFRA

{ \$\$ = baza\*\$1 + \$2; };

% %

#include "putanja\Lex.yy.c"

yyerror (char \*s)

{ fprintf(stderr, "%s\n", s); }

main()

{ return yyparse(); }

}

Sačuvamo kao zad4.y, pa "jakujemo":

bison zad4.y

\* Za promjenu, možemo da pogledamo i parser koji uključuje direktno napisan leksički analizator - dakle, funkcija `yylex()` upisana je direktno, bez koristenja FLEX-a.

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
int registri [26];  
int baza;  
%}  
  
% start List  
% token CIFRA SLOVO  
% Left '1'  
% Left 'l'  
% Left '+ -'  
% Left '* /'  
% left UMINUS  
  
% %
```

List:  
| List iskaz '\n'  
| List error '\n'  
| yyerror; }

iskaz: izraz  
{ printf (" očod\n", \$1); }  
| SLOVO != izraz  
{ registri [\$1] = \$3; };

izraz: '(' izraz ')'  
{ \$\$ = \$2; }  
| izraz '+' izraz  
{ \$\$ = \$1 + \$3; }  
| izraz '-' izraz  
{ \$\$ = \$1 - \$3; }

| razraz '\*' razraz  
{ \$\$ = \$1 \* \$3; }

| razraz '/' razraz  
{ \$\$ = \$1 / \$3; }

| razraz '%' razraz  
{ \$\$ = \$1 % \$3; }

| razraz '£' razraz  
{ \$\$ = \$1 £ \$3; }

| razraz '!' razraz  
{ \$\$ = \$5 | \$1; }

| '-' razraz      % prec UMINUS  
{ \$\$ = - \$2; }

| SLOVO

{ \$\$ = registri [ \$1 ]; }

| broj;

broj : CIFRA

{ \$\$ = \$1 ;    baza = (\$1 == 0)? 8 : 10; }

| broj CIFRA

{ \$\$ = baza \* \$1 + \$2 ; };

% %

yylex()

{ int c ;  
while ((c = getchar()) != ' ' )  
{ /\* preskoci blankove \*/ }

if (islower(c))

{ yyval = c - 'a' ; }

} return (SLOVO);

// rutina za leksičku analizu  
vraca slovo ako prepozna  
malo slovo //

```

if (isdigit (c))
    // vraca CIFRA ako prepozna
    // cifru i
    yylval = c - '0';
    return (CIFRA);
}

return (c);           // sve druge karaktere vraca odmah //
}

yyerror (char *s)
{
    fprintf (stderr, "%s\n", s);
}

main ()
{
    return yyparse ();
}

```

□

### \* Prevodenje u obrnuta poljsku notaciju

Obrnuta poljska notacija je način zapisivanja aritmetičkih izraza pri kojem se binarni operatori ispisuju posfiksno (umjesto infiksno, kao što je uobičajno). Dakle, umjesto  $a+b$  pišemo  $ab+$ , dakle, prvo argumente a zatim oznaku operadora. Prednost obrnute poljske notacije je u tome što takav zapis eliminira potrebu za zagradama i eksplisitnim zadavanjem prioriteta, npr.

$a-b-a$  prevodi se u  $ab-a-$

$a-(b-a)$  prevodi se u  $aba--$

- Izračunavanje izraza zapisanih obrnutom poljskom notacijom je najlakše objasniti pomoću steka: dok očitavamo promjenjive složimo ih na stek, a kada očitamo operator primjenimo ga na dve prethodne promjenjive koje su na vrhu steka i rezultat opet upišemo na vrh steka. Na primjer:

Izraz  $3 - 7 - 5$ , čija je vrijednost  $-9$ , prevodi se u  $3 \# - 5$  i izračunava se na sledeći način:

	$\#$	$5$		
3	3	-4	-4	-9

S druge strane, izraz  $3 - (7 - 5)$ , čija je vrijednost  $1$ , prevodi se u izraz  $3 \# 5 --$ , koji se izračunava na sledeći način:

	$\#$	$5$		
3	3	3	3	1

Dakle, zadatak je sledeći:

### Zadatak 5:

(a) Napisati opis za YACC koji prevodi aritmetičke izraze u obrnutu poljsku notaciju. Za početak operatori su  $+$  i  $-$ , a operandi  $a$  i  $b$ .

**Rešenje:** prf.y.y

Kako se sve lekseme svode na samo po jedan karakter, leksički analizator se svodi na getchar() funkciju koja uzima jedan po jedan karakter s ulaza:

\*/\}

#include <stdio.h>

\*/\}

\*/\%

Izraz: sabirale nastavak;

nastavak: '+' sabirak { printf ("+"); } nastavak  
| '-' sabirak { printf ("-"); } nastavak  
| ;

sabirak : 'a' { printf ("a"); }  
| 'b' { printf ("b"); }

\*/\%

```

yylex()
{
    getchar();
}

yyerror (char *s)
{
    printf ("%qos", s);
}

main()
{
    printf ("\n Unesite aritmetički izraz sa
            operandima a i b i operatorima + i - \n");
    return yyparse();
    printf ("\n");
}

```

(b) Uz dodate operatore \* i /.

Rešenje: prfa.y

```
% {
#include <stdio.h>
%
```

```
%%
```

Izraz : sabirak nastavak;

Nastavak: '+' sabirak { printf ("+"); } nastavak  
 | '-' sabirak { printf ("-"); } nastavak  
 | ;

sabirak: faktor nastavak;

Nastavak: '\*' faktor { printf ("\*"); } nastavak  
 | '/' faktor { printf ("/"); } nastavak  
 | ;

```

faktor: 'a' { printf ("a"); }
| 'b' { printf ("b"); }
| '(' izraz ')';
%
% yylex()
{
    getchar();
}
yyerror (char *s)
{
    printf ("%os", s);
}
main()
{
    printf ("Unesite aritmetički izraz sa operandima
            a i b , operatorima +,-,*,/ . Zagrade su
            dozvoljene.");
    return yyparse();
    printf ("\n");
}

```

(c) Treća verzija kao argumente prihvata cijele brojeve i identifikatore.

**Rešenje:** prob.y

Za ovu verziju nam je potreban leksički analizator, koji se dobija pomoću FLEX-a.

Opis za FLEX:

qq option noyywrap

slovo [A-zA-Z]

cifra [0-9]

bjel [\t\n]

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

```
%eo {cifra}+
%%
{ident} {return IDENT;}
{nco} {return NCEO;}
{bjel}
.
{ return yytext[0];}
```

Opis za YACC:

```
%{
#include <stdio.h>
%}
%token IDENT NCEO
%%
izraz : sabirak nastavak;
nastavak : '+' sabirak {printf ("+"); } nastavak
          | '-' sabirak {printf ("-"); } nastavak
          | ;
sabirak : faktor nastavak;
nastavak : '*' faktor {printf ("*"); } nastavak
          | '/' faktor {printf ("/"); } nastavak
          | ;
faktor : NCEO {printf ("~%s~", yytext); }
        | IDENT {printf ("~%s~", yytext); }
        | '(' izraz ')';
%%
#include "putanja\lex.yy.c"
yyerror (char *s)
{ printf ("%s", s);
}
```

```

main()
{ printf ("Unesite aritmetički izraz");
    return yyparse();
    printf ("\n");
}

```

□

### Zadatak 6:

Napisati opis za yacc koji prevodi rimske brojeve u arapske.

#### Rešenje:

Leksički analizator se svodi na funkciju getch(), koja uzima po jedan karakter iz ulaznog toka.

%{

#include <stdio.h>

%}

% %

s : t r g P

{ \$\$ = 1000 \* \$1 + 100 \* \$2 + 10 \* \$3 + \$4;  
 printf ("Broj je %d!\n", \$\$); }

p: 'I' { \$\$ = 1; }

'I'I'I' { \$\$ = 2; }

'I'I'I'I'I' { \$\$ = 3; }

'I'I'I'VI' { \$\$ = 4; }

'I'VI' { \$\$ = 5; }

'IV'I'I' { \$\$ = 6; }

'IV'I'I'I'I' { \$\$ = 7; }

'IV'I'I'I'I'I'I' { \$\$ = 8; }

'I'I'X' { \$\$ = 9; }

		$\{ \text{ $$} = 0 \} ;$
g:	'x'	$\{ \text{ $$} = 1; \}$
	'x' 'x'	$\{ \text{ $$} = 2; \}$
	'x' 'x' 'x'	$\{ \text{ $$} = 3; \}$
	'x' 'x' 'L'	$\{ \text{ $$} = 4; \}$
	'L'	$\{ \text{ $$} = 5; \}$
	'L' 'x'	$\{ \text{ $$} = 6; \}$
	'L' 'x' 'x'	$\{ \text{ $$} = 7; \}$
	'L' 'x' 'x' 'x'	$\{ \text{ $$} = 8; \}$
	'x' 'C'	$\{ \text{ $$} = 9; \}$
		$\{ \text{ $$} = 0; \};$
r:	'C'	$\{ \text{ $$} = 1; \}$
	'C' 'C'	$\{ \text{ $$} = 2; \}$
	'C' 'C' 'C'	$\{ \text{ $$} = 3; \}$
	'C' 'D'	$\{ \text{ $$} = 4; \}$
	'D'	$\{ \text{ $$} = 5; \}$
	'D' 'C'	$\{ \text{ $$} = 6; \}$
	'D' 'C' 'C'	$\{ \text{ $$} = 7; \}$
	'D' 'C' 'C' 'C'	$\{ \text{ $$} = 8; \}$
	'C' 'M'	$\{ \text{ $$} = 9; \}$
		$\{ \text{ $$} = 0; \};$

```
t: 'M' { $$ = 1; }
| 'M' 'M' { $$ = 2; }
| 'M' 'M' 'M' { $$ = 3; }
|
{ $$ = 0; }
```

%

yylex ()

```
{ getchar();
```

```
}
```

yyerror (char \*s)

```
{ printf ("%os\n", s);
```

```
}
```

main()

```
{ return yyparse();}
```

```
}
```

□