Datoteka:      Lista\_zapisa EOF  
Lsta\_zapisa:      Zapis | Lista\_zapisa Zapis  
Zapis:         Ime Prezime Godina NL | NL | /\*empty\*/  
Ime:           IDENTIFIKATOR  
Prezime:       IDENTIFIKATOR  
Godina:        BROJ  
Tokeni su: EOF, NL-nova linija, IDENTIFIKATOR, BROJ. Podrazumijeva se da se tokeni dobiju funkcijom yylex() i da je leksem zapisan u globalnoj varijabli yytext.  
  
Napišite  parser pomoću programa YACC, kojim se ucitava iz datoteke  u  niz data[]:   
  
struct  \_data  
{

char ime[50];

char prezime[50];

int godina

} data[1000];  
  
U yacc-u treba koristiti globalnu varijablu  
  
int indeks=0;  
  
kojom se prati indeks do kojeg je popunjen niz;  
  
  
Alternativno upis možete vršite u i listu  
  
typedef struct  \_dataElem  
{

char ime[50];

char prezime[50];

int godina;

struct \_dataElem \* next;

} dataElem;  
  
koju u yacc-u deklariramo kao globalnu varijablu:  
  
dataElem \* List = NULL;

**VEZANA LISTA**

**LEX:**  
  
%{  
#include <stdlib.h>  
#include "strukt.h"  
#include "y.tab.h"  
char \*strdup(char \*s);  
%}  
  
              
CHAR [A-Za-z]              
STRING {CHAR}+  
INT    [1-9][0-9]\*  
  
              
%%  
  
\n        return NL;  
^{STRING}      { yylval.string = strdup(yytext);       return IDENTIFIKATOR;}  
{STRING}      { yylval.string = strdup(yytext);       return IDENTIFIKATOR;}  
{INT}        { yylval.integer = atoi(yytext);        return BROJ; }      
<<EOF>>    return EOF;  
[ \t]+         ;  
      
%%          
int yywrap(void)  
{

return 1;

}  
              
char \*strdup(char \*s)  
{              
     char \*d = (char \*)malloc(strlen(s)+1);  
      strcpy(d,s);  
       return d;              
}   
  
  
**strukt.h**  
  
typedef struct  \_dataElem  
{

char ime[50];

char prezime[50];

int godina;

struct \_dataElem \* next;

} \*dataElem;  
  
**YACC:**  
          
                      
%{  
#include <stdio.h>  
#include <stdio.h>  
#include “strukt.h”      
      
dataElem list = NULL;  
print\_userinfo\_list();          
%}  
  
  
%union {

int integer;  
char\* string;  
dataElem user;

}   
                                      
%token EOF NL                  
%token <string> IDENTIFIKATOR  
%token <integer> BROJ  
%type <user> Datoteka Lista\_Zapisa Zapis  
                      
%%  
  
Datoteka       :               Lista\_Zapisa EOF             {list = $$;}  
            ;  
Lista\_Zapisa  :        Zapis                    {$$ = $1;}

| Lista\_Zapisa Zapis             {$$ = $1; $$->next=$2;}

;

Zapis            :        IDENTIFIKATOR IDENTIFIKATOR BROJ NL

{ $$ = (dataElem)malloc(sizeof(struct \_dataElem));

strcpy($$->ime,$1);

strcpy($$->prezime, $2);

$$->godina=$3;

$$->next = NULL;}

| NL

;                 

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

yyparse();

       print\_userinfo\_list();  
 }  
                  
print\_userinfo\_list()  
{      
 while (list != NULL)  
  {                  
    printf("%s",list->ime); /\* samo ispisujemo ime \*/  
    list = list->next;  
   }  
}   
               

**NIZ**

              
**LEX:**  
  
%{  
#include <stdlib.h>  
#include "strukt.h"  
#include "y.tab.h"  
%}  
      
CHAR [A-Za-z]              
STRING {CHAR}+  
INT    [1-9][0-9]\*  
          
%%  
  
\n        return NL;  
^{STRING}      return IDENTIFIKATOR;  
{STRING}      return IDENTIFIKATOR;  
{INT}        return BROJ;     
<<EOF>>    return EOF;  
[ \t]+         ;  
      
%%          
int yywrap(void) {return 1; }  
      
  
  
  
  
  
          
**YACC:**  
                  
                      
%{  
     #include <stdio.h>  
    struct  \_data  
     {              
          char ime[50];  
           char prezime[50];  
           int godina;          
    } data[1000];  
                      
    int count = 0;  
    void printData();  
    void yyerror(char \*);  
                  
 %}  
  
 %token IDENTIFIKATOR BROJ NL EOF    //nemamo union, stog, pa ne triba tip stavit  
  
%%  
  
                      
Datoteka: Lista\_zapisa EOF ;  
Lista\_zapisa: Zapis  
                   | Lista\_zapisa Zapis

       ;

Zapis: Ime Prezime Godina NL { count++; }

| NL             {count++;}

| /\*empty\*/

;

Ime: IDENTIFIKATOR         { strcpy(data[count].ime, yytext); }  
      ;  
Prezime: IDENTIFIKATOR     { strcpy(data[count].prezime, yytext); }   
             ;  
Godina: BROJ                { data[count].godina = atoi(yytext); }  
           ;  
                      
%%                      
void yyerror(char \*s)  
{  
    fprintf(stderr, "Error: %s\n", s);  
}  
  
void printData()  
{  
    for(int i = 0; i < count; i++)  
    printf("%s %s - %d\n", data[i].ime, data[i].prezime,  
    data[i].godina);  
}  
  
int main()  
{          
          yyparse();  
           printData();  
}   
          
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Zadatak 3.**  Koristite sljedeću (nepotpunu) specifikacije gramatike i semantičkih akcija za proračun izraza pomoću YACC programa  
    %union {  
        double broj;

}

%%

Naredba : Izraz END\_TOKEN   {print(“Rezultat =%f”, $1);}

       ;

Izraz       : Izraz ‘+‘ Clan    {$$ = ????} // $$=$1+$3;

       | Izraz ‘-‘ Clan    {$$ = ????} // $$=$1-$3;

       | Clan              {$$ = ????} // $$=$1

;

Clan    : Clan ‘\*‘ Faktor   {$$ = ????} // $$=$1\*$3;

       | Clan ‘/‘ Faktor   {$$ = ????} // $$=$1/$3;

       | Faktor            {$$ = ????} // $$=$1;

       ;

Faktor  : NUM\_TOKEN         {$$ = ????}     // $$=$1->broj; ili $$=$1;

       | '(' Izraz ')'     {$$ = ????}         // $$=$2;

       ;

%%

void main() {yyparse();}

a)  Napišite specifikaciju leksičkog analizatora za program LEX koja vraća potrebne tokene: ‘+‘,‘-‘, ‘\*‘,‘/‘ , NUM\_TOKEN (realni broj) i END\_TOKEN (znak nove linije).

b)      Dopunite YACC specifikaciju - napišite semantičke akcije pomoću $n-atributa koji se definiraju  u %union sekciji.  Atribut NUM\_TOKEN je realni broj. U YACC specifikaciji se može dobiti pozivom funkcije atof(yytext).

**LEX:**  
  
%{  
#include <stdlib.h>  
#include "y.tab.h"  
%}  
          
BROJ [0-9]+(\.[0-9]+)?  
  
%%  
  
BROJ         {yylval.broj = atof(yytext);  return NUM\_TOKEN;}  
\n        return END\_TOKEN;  
[-+\*/]        return yytext;  
  
[ \t]+         ;  
      
%%          
int yywrap(void) {return 1; }  
  
  
  
  
  
  
  
**Zadatak :** U datoteci su zapisani podaci prema pravilu:  
  
Datoteka: Zapis\* EOF  
Zapis: Ime Prezime Godina NL  
Ime: IDENTIFIKATOR  
Prezime: IDENTIFIKATOR  
Godina: BROJ  
  
Tokeni se dobiju funkcijom  int yylex():  
EOF – kraj datoteke  
NL - nova linija,  
IDENTIFIKATOR – niz slova,  
BROJ – niz brojeva  
ERR – svi ostali unosi su greška  
  
String tokena je u globalnoj varijabli char \*yytext.  
  
Napišite rekurzivni parser kojim se učitavaju podaci iz datoteke u  vezanu listu definiranu strukturom:   
  
typedef struct  \_data  
{  
 char ime[50]; char prezime[50]; int godina; struct  \_data \*next  
} dataNode;  
  
Početno je definirana prazna lista:  
  
dataNode \*List = NULL;  
  
  
**LEX:**  
  
%{  
#include <stdlib.h>  
%}  
      
CHAR [A-Za-z]              
STRING {CHAR}+  
INT    [1-9][0-9]\*  
          
%%  
  
\n        return NL;  
^{STRING}      return IDENTIFIKATOR;  
{STRING}      return IDENTIFIKATOR;  
{INT}        return BROJ;     
<<EOF>>    return EOF;  
[ \t]+         ;  
.        return ERR;  
  
%%          
int yywrap(void) {return 1; }  
  
  
  
**Parser**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include "lex.yy.c"

#define NL            0

#define IDENTIFIKATOR     1

#define BROJ         2

#define EOF2         3

#define ERR         4

struct \_data;

typedef struct  \_data

{

   char ime[50];

   char prezime[50];

   int godina;

   struct \_data \*next;

} dataNode;

dataNode \*List = NULL;

int getToken() {

    return yylex();

}

void parseZapis();

void parseIme();

void parsePrezime();

void parseGodina();

int lookaheadToken;

void reportError(char \*str)

{

    printf("%s\n",str);

    exit(1);

}

void match(int s)

{

    if (lookaheadToken == s)

       lookaheadToken = getToken();

    else

       reportError("syntax error");

}

void parseDatoteka() {

    while(lookaheadToken != EOF2 && lookaheadToken != ERR) {

        parseZapis();

       match(NL);

   }

    if (lookaheadToken == ERR) {

        reportError("greska");

   }

   if (lookaheadToken == EOF2) {

        reportError("super");

   }

}

void parseZapis() {

   dataNode \*temp =(dataNode\*)malloc(sizeof(struct \_data));

   List->next = temp;

   temp->next = NULL;

   if (lookaheadToken==IDENTIFIKATOR) {

       strcpy (temp->ime, yytext);

       match(IDENTIFIKATOR);

   }

   if (lookaheadToken==IDENTIFIKATOR) {

       strcpy (temp->prezime, yytext);

       match(IDENTIFIKATOR);

   }

   if (lookaheadToken==BROJ) {

       temp->godina=atof(yytext);

       match(BROJ);

   }

}

int main() {

   List = (dataNode\*)malloc(sizeof(struct \_data));

   List->next =NULL;

   lookaheadToken = getToken();

   parseDatoteka();

   return 0;

}

**Zadatak 3.**  Napišite leksički analizator i rekurzivni parser za gramatiku  matematičkih izraza u kojima se se mogu pojavljivati  realni brojevi, operatori (+,-, \*, /) i separatori (zagrade, prazna mjesta i tabulatori);

Izraz   : Clan  (PLUS | MINUS) Izraz)\* NL

 ;

Clan    : Faktor ((PUTA | KROZ) Clan)\*

       ;

Faktor  : BROJ

       | LIJEVA\_ZAGRADA Izraz DESNA\_ZAGRADA

       ;

Leksički analizator treba biti specificiran za program Lex, koji generira kod lesičkog analizatora unutar funkcije int yylex(). Funkcija vraća vrijednost sljedećeg tokena, a leksem tokena se bilježi u globalnom stringu  char yytext[].  
  
Tokeni (i leksemi) su NL (‘\n’), BROJ (realni broj), PLUS ('+'), MINUS ('-)'. PUTA ('\*'), KROZ('/') , LIJEVA\_ZAGRADA ('(')  i DESNA\_ZAGRADA ')'.  
  
Nakon prepoznavanja izraza, koji je zaključen znakom nove linije parser treba dati numeričku vrijednost izraza , odnosno glavna funkcija parsera je oblika:  
  
      double Izraz();   
  
  
**LEX:**  
  
%{  
#include <stdlib.h>  
%}  
  
INT    0([1-9][0-9]\*)](\.[0-9]+)?  
              
%%  
  
{INT}    return BROJ;  
\+    return PLUS;  
\-     return MINUS;  
\\*    return PUTA;  
\/    return KROZ;  
\(    return LIJEVA\_ZAGRADA;  
\)    return DESNA\_ZAGRADA;  
\n    return NL  
%%      
      
int yywrap(void) {return 1; }  
  
**PARSER:**  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include "lex.yy.c"  
  
#define PLUS                0   
#define MINUS                  1   
#define PUTA                 2  
#define KROZ                 3  
#define LIJEVA\_ZAGRADA         4  
#define DESNA\_ZAGRADA         5  
#define BROJ                 6  
#define NL                 7  
  
int lookaheadToken;  
int getToken() { return yylex(); }  
  
double parseIzraz();  
double parseClan();  
double Faktor();  
  
void reportError(char \*str)  
{printf("%s\n",str);  exit(1);  }  
  
void match(int s)  
{  
    if (lookaheadToken == s) {lookaheadToken = getToken();}  
    else {reportError("syntax error");}  
}   
  
  
double parseIzraz(){  
    double val = parseClan();  
    if (lookaheadToken == PLUS){  
        val += parseIzraz();  
        match(PLUS);

}

if (lookaheadToken == MINUS){

        val -= parseIzraz();  
        match(MINUS);

}

match(NL);

return val;

}  
  
  
double parseClan(){  
    double val = parseFaktor();  
      
    if (lookaheadToken == PUTA){  
        val \*= parseIzraz();  
        match(PUTA);

}

    if (lookaheadToken == KROZ){  
        val /= parseIzraz();  
        match(KROZ);

}

return val;

}  
  
double parseFaktor(){  
    double val;  
    if (lookaheadToken == BROJ){  
        val = atof(yytext);

match(BROJ);

}

    else if (lookaheadToken == LIJEVA\_ZAGRADA){  
        match(LIJEVA\_ZAGRADA);  
        val = parseIzraz();  
        match(DESNA\_ZAGRADA);

}

}  
  
int main() {  
  
    lookaheadToken = getToken();  
    printf(“Rezultat je: %lf”, parseIzraz());

return 0;

}

**Zadatak:** Napišite rekurzivni parser za koji se može dobaviti posebno zapisani realni broj. Dozvoljni su oblici : 1.78, 12, 12., 0.45, .45, i iza broja se smije zapisati sufiks znak (T- tera, M-mega, k-kilo, m- mili, u- mikro, n-nano)

Za izradu parsera koristite sljedeći EBNF zapis gramatike:

Broj : RealniBroj (sufix)?

RealniBroj : (digit)+ (dot (digit)+)?

                    |  dot (digit)+

               ;

sufix : 'T' | 'M' | 'k' | 'm'| 'u'| 'n';

digit :'0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' ;

dot : '.'

;

Parsiranje započnite funkcijom double ParseBroj(), koja treba vratiti vrijednost broja. Kao leksički analizator koristite funkciju int GetToken() koja u ovom jednostavnom slučaju vraća jedan od znakova određenih s pravilima gramatike (suffix, digit i dot) ili -1 za ostale znakove.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define BAD\_TOKEN           0

#define DOT\_TOKEN           1

#define NUM\_TOKEN           2

#define END\_TOKEN           3

#define SFX\_TOKEN           5

typedef int Token;

int lookaheadToken;

char lexeme[2]; /\* lexeme string \*/

void reportError(char \*str) /\* error handling \*/

{

    printf("%s\n",str);

    exit(1);

}

int GetToken(void)  /\* Scanner \*/

{

    int t;

    while(1)

    {

       t=getchar();

       if (t == ' '  || t=='\t')

           continue; /\* skip white spaces \*/

       if (t == '\n' || t==EOF) return END\_TOKEN;

       lexeme[0]=(char)t; lexeme[1]='\0';

       if (isdigit(t))

           return NUM\_TOKEN;

       else if (t == '.')

           return DOT\_TOKEN;

       else if (t == 'T' || t == 'M' || t == 'k' || t == 'm' || t == 'u' || t == 'n') {

           return SFX\_TOKEN;

       }

       else

           return BAD\_TOKEN;

    }

}

void match(Token s)

{

    if (lookaheadToken == s)

       lookaheadToken = GetToken();

    else

       reportError("syntax error");

}

double parseBroj();

double parseRealniBroj();

double parseSufix();

double parseDigit();

double parseDot();

double parseBroj(){

    double val;

    val = parseRealniBroj();

    double tmp = parseSufix();

    val \*= tmp;

    return val;

}

double parseRealniBroj(){

    double val = 0;

    val = parseDigit();

    val += parseDot();

    return val;

}

double parseDigit(){

    double val = 0;

    while (lookaheadToken == NUM\_TOKEN) {

       val += atof(lexeme);

       match(NUM\_TOKEN);

       if (lookaheadToken == NUM\_TOKEN) {

           val \*= 10;

       }

    }

    return val;

}

double parseDot(){

    double val, tmp;

    if (lookaheadToken == DOT\_TOKEN) {

       match(DOT\_TOKEN);

       int counter = 0;

       tmp = parseDigit();

       val = tmp;

               while (tmp > 1) {

           tmp /= 10;

           counter++;

       }

           tmp = 1;

       for (int i=0; i<counter; i++) {

           tmp \*=10;

       }

           return val/tmp;

    }

}

double parseSufix(){

    double val = 1;

    if (lookaheadToken == SFX\_TOKEN) {

       if (lexeme[0] == 'T') {

           val = 1000000000;

       }

       else if (lexeme[0] == 'M') {

           val = 1000000;

       }

       else if (lexeme[0] == 'k') {

           val = 1000;

       }

       else if (lexeme[0] == 'm') {

           val = 0.01;

       }

       else if (lexeme[0] == 'u') {

           val = 0.00001;

       }

       else if (lexeme[0] == 'n') {

           val = 0.00000001;

       }

       else {

           val = 1;

       }

    }

    return val;

}

int main()

{

    lookaheadToken=GetToken();

    printf("Broj je: %lf\n", parseBroj());

    return 0;

}

#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
  
#define BAD\_TOKEN         0  
#define DOT\_TOKEN         1  
#define NUM\_TOKEN         2  
#define END\_TOKEN         3   
#define SUF\_TOKEN         4  
  
typedef int Token;  
  
int lookaheadToken;  
char lexeme[2]; /\* lexeme string \*/  
  
  
void reportError(char \*str)    /\* error handling \*/  
{  
   printf("%s\n",str);  
   exit(1);      
}  
  
int GetToken(void)  /\* Scanner \*/  
{   
   int t;  
   while(1)  
   {   
       t=getchar();   
       if (t == ' '  || t=='\t')   
           continue; /\* skip white spaces \*/  
       if (t == '\n' || t==EOF)   
           return END\_TOKEN;  
       lexeme[0]=(char)t; lexeme[1]='\0';  
       if (isdigit(t))   
           return NUM\_TOKEN;  
       else if (t=='.')    
           return DOT\_TOKEN;  
       else if (t == 'T' || t == 'M' || t == 'k' || t == 'm' || t == 'u' || t == 'n')   
           return SUF\_TOKEN;   
       else  
           return BAD\_TOKEN;      
   }  
}  
  
void match(Token s)  
{  
   if (lookaheadToken == s)  
       lookaheadToken = GetToken();  
   else  
       reportError("syntax error");  
}   
  
double parsirajBroj();  
double parsirajDotBroj();  
double parsirajSufix();  
  
void parseNaredba()  
{        
   double val=0;  
    
   while(lookaheadToken==NUM\_TOKEN) {  
       val=10\*val+parsirajBroj();            
   }  
     
   val= val +parsirajDotBroj();      
   val= val \*parsirajSufix();   
     
   if (lookaheadToken==END\_TOKEN)  
       printf("dobro je: %lf", val);  
   else {  
       printf("lose je");  
   }  
}   
  
double parsirajBroj() {  
   double val=0;  
   if(lookaheadToken==NUM\_TOKEN) {   
       val=atof(lexeme);  
       match(NUM\_TOKEN);  
   }  
   return val;  
     
}  
double parsirajDotBroj() {  
   double val=0;  
   if(lookaheadToken==DOT\_TOKEN) {  
       match(DOT\_TOKEN);   
       double i=0.1;  
       while(lookaheadToken==NUM\_TOKEN) {  
           val=val+i\*parsirajBroj();      
           i=i/10;       
       }          
   }      
   return val;  
}  
  
  
double parsirajSufix() {  
   double val=1;  
   if (lookaheadToken == SUF\_TOKEN) {  
       if (lexeme[0]=='T')  
           val=1000000000;  
       if (lexeme[0]=='M')  
           val=1000000;  
       if (lexeme[0]=='k')  
           val=1000;  
       if (lexeme[0]=='m')  
           val=0.001;          
       if (lexeme[0]=='u')  
           val=0.000001;  
       if (lexeme[0]=='n')  
           val=0.000000001;  
         
       match(SUF\_TOKEN);          
   }      
    return val;      
}  
int main()   
{  
   lookaheadToken=GetToken();  
   parseNaredba();  
   printf("\n");  
   return 0;}  
  
**Zadatak 4:** U datoteci su zapisani podaci prema pravilu da je u svakoj liniji zapisano ime i prezime i opciono identifikacijsko broj. Taj broj može biti OIB ili JMBG. Ovo pravilo je određeno sljedećom gramatikom:  
  
  
Datoteka: Zapis\* EOF  
Zapis: Ime Prezime (Id)? NL  
Id: (OIB ':' BROJ) | (JMBG ':' BROJ)   
Ime: IDENTIFIKATOR  
Prezime: IDENTIFIKATOR  
  
  
Tokeni se dobiju funkcijom  int yylex(), koja vraća tokene:  
  
EOF – kraj datoteke  
NL - nova linija,  
IDENTIFIKATOR – niz slova (yytext sadrži string)  
BROJ – niz brojeva  (yytext sadrži string s nizom brojeva)  
OIB – oznaka identifikacijskog broja  (yytext sadrži string “OIB“)  
JMBG – oznaka starog matičnog broja (yytext sadrži string “JMBG“)  
  
  
String tokena je u globalnoj varijabli char \*yytext.  
  
  
Napišite rekurzivni parser kojim se učitavaju podaci iz datoteke u  vezanu listu definiranu strukturom:   
  
typedef struct  \_data  
{  
 char ime[50];  
 char prezime[50];  
 int Id; struct  \_data \*next  
} dataNode;

//Polje Id se formira prema pravilu:

//   pozitivni integer - znači da sadrži OIB)  
//   negativni integer - znači da sadrži JMBG)  
//   nula - znači da nije unesena vrijednost ni OIB-a ni JMBG)  
  
  
Početno je definirana prazna lista:  
  
dataNode \*List = NULL;  
  
  
  
  
LEX:  
  
%{  
#include <stdlib.h>  
%}  
      
CHAR [A-Za-z]              
STRING {CHAR}+  
          
%%  
[0-9]\*{11,13}       return BROJ;              
^{STRING}      return IDENTIFIKATOR;  
{STRING}      return IDENTIFIKATOR;  
"OIB"                return OIB;                 
"JMBG"             return JMBG;  
<<EOF>>    return EOF\_TOK;  
\n        return NL;  
.        return ERR;  
  
%%          
int yywrap(void) {return 1; }  
  
  
PARSER: VEZANA LISTA  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include "lex.yy.c"    
  
#define EOF            0  
#define NL            1  
#define IDENTIFIKATOR     2  
#define BROJ             3  
#define OIB            4  
#define JMBG            5  
  
struct \_data;  
  
typedef struct  \_data  
{  
 char ime[50];  
 char prezime[50];  
 int Id; struct  \_data \*next  
} dataNode;                             
  
dateNode \*List = NULL;  
dataNode \*tmp, \*zadnji;  
  
int getToken() {  
    return yylex();  
}  
  
void parseZapis();  
void parseId();  
void parseIme();  
void parsePrezime();  
  
int lookaheadToken;  
  
void reportError(char \*str)  
{  
    printf("%s\n",str);  
    exit(1);      
}  
  
void match(int s)  
{

if (lookaheadToken == s)

               lookaheadToken = getToken();  
    else  
               reportError("syntax error");  
}   
  
void parseDatoteka(){  
    while (lookaheadToken != EOF){

    parseZapis();

}

match (EOF);

}  
  
void parseZapis(){  
  
    if (lista == NULL) {  
        lista = (\*dataNode)malloc(sizeof(dataNode));  
        lista->next = NULL;  
        tmp = lista;

} else {

        tmp = (\*dataNode)malloc(sizeof(dataNode));  
        zadnji->next = tmp;

}

    parseIme();  
    parsePrezime();  
    if (lookaheadToken == OIB || lookaheadToken == JMGB)  
        parseId();  
    match(NL);  
}  
  
void parseIme() {  
    strcpy(tmp->ime, yytext);  
    match(IDENTIFIKATOR);  
}  
  
void parsePrezime() {  
    strcpy(tmp->prezime, yytext);  
    match(IDENTIFIKATOR);  
}  
  
  
void parseId(){  
    if (lookaheadToken == OIB){

    match(OIB);

    match(“:”);

tmp->id = atoi(yytext);

    match(BROJ);

}

else if (lookaheadToken == JMBG){

    match(JMBG);

    match(“:”);

    tmp->id = -atoi(yytext);

    match(BROJ);

}

else

    tmp->id = 0;

}  
  
int main() {  
    lookaheadToken = getToken();  
    parseDatoteka();  
    tmp = lista;

while (tmp != NULL){

    prinf(“%s %s %d”, tmp->ime, tmp->prezime, tmp->id.);

    tmp = tmp->next;

}

    return 0;  
}  
  
/\* opcenito za liste  
dataNode \*tmp, \*zadnji;  
  
if (lista == NULL) {  
    lista = (\*dataNode)malloc(sizeof(dataNode));  
    lista->next = NULL;  
    tmp = lista;  
} else {  
    tmp = (\*dataNode)malloc(sizeof(dataNode));  
    zadnji->next = tmp;  
}  
zadnji = tmp;  
\*/  
  
**PRETHODNI PRIMJER PREKO NIZA**  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include "lex.yy.c"    
  
#define EOF            0  
#define NL            1  
#define IDENTIFIKATOR     2  
#define BROJ             3  
#define OIB            4  
#define JMBG            5  
  
struct  \_data  
{  
 char ime[50];  
 char prezime[50];  
 int Id;  
} data[1000];                             
  
int counter = 0;  
  
int getToken() {  
    return yylex();  
}  
  
void parseZapis();  
void parseId();  
void parseIme();  
void parsePrezime();  
  
int lookaheadToken;  
  
void reportError(char \*str)  
{  
    printf("%s\n",str);  
    exit(1);      
}  
  
void match(int s)  
{

if (lookaheadToken == s)

               lookaheadToken = getToken();  
    else  
               reportError("syntax error");  
}   
  
void parseDatoteka(){  
    while (lookaheadToken != EOF){

    parseZapis();

// counter ++;

}

match (EOF);

}  
  
void parseZapis(){  
  
    parseIme();  
    parsePrezime();  
    if (lookaheadToken == OIB || lookaheadToken == JMGB)  
        parseId();  
    match(NL);

counter++;

}  
  
void parseIme() {  
    strcpy(data[counter].ime, yytext);  
    match(IDENTIFIKATOR);  
}  
  
void parsePrezime() {  
    strcpy(data[counter].prezime, yytext);  
    match(IDENTIFIKATOR);  
}  
  
  
void parseId(){  
    if (lookaheadToken == OIB){

    match(OIB);

    match(“:”);

data[counter].id = atoi(yytext);

    match(BROJ);

}

else if (lookaheadToken == JMBG){

    match(JMBG);

    match(“:”);

    data[counter].id = -atoi(yytext);

    match(BROJ);

}

else

    tmp->id = 0;

}  
  
int main() {  
    int i;

lookaheadToken = getToken();

    parseDatoteka();

for(i = 0; i < counter; i++){

    prinf(“%s %s %d”, tmp[i]ime, tmp[i].prezime, tmp[i].id.);

}

    return 0;  
}