4.4.2001.

Predmet: Automati, formalni jezici i jezični procesori II
Školska godina: 2000/2001.

ZEMRIS

Dinko Korunić
0036355514

Laboratorijske vježbe

Dokumentacija za izvedbu lexera za ANSI-C jezik

Opis radnog zadatka:

Pomoću programskog paketa Lex izvesti leksičku obradu programskog jezika C.

Implementacija:

Odlučio sam se za ANSI-C specifikaciju jezika C radi inicijalne jednostavnosti i striktnosti. Osim toga, za prevođenje *Lex* specifikacija u izlaznu datoteku jezika C koristio sam *Flex*, kao nešto napredniji alat od originalnog *Lex*-a.

U samoj izradi *Lex* specifikacija koristio sam se originalnim ANSI-C specifikacijama gramatike i leksičkih jedinki. Osim toga, odlučio sam dodatno implementirati prepoznavanje preprocesorskih direktiva koje standardno obrađuje preprocesor (u mojem slučaju *Cpp*) prije samog compilera.

U razvoju sam se koristio tipičnim Unixoidnim editorom Vim zbog same prirode platforme na kojoj sam izvorni kod i razvijao - Linux i Solaris operativni sustavi.

```
/*
  Dinko Korunic,
  Fri Mar 30 18:54:40 CEST 2001

  Mostly according to literature and ANSI-C grammar
*/

%option case-sensitive
```

Ova opcija se odnosi na Flex - specificira se da regexp izrazi koje će lexer prepoznavati moraju biti "case sensitive", odnosno da se izričito velika i mala slova moraju razlikovati.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define lexd(t) { fprintf(stderr, t); }
```

Makro lexd() mi ovdje služi da na "standard error" izlaz ispisujem leksičke jedinke koje lexer prepoznaje i koje bi inače vratio nadležnom procesu, recimo *Yacc* generiranom parseru.

```
void count(void);
void comment(void);
void include(void);
```

Ovo su prototipovi funkcija koje kasnije koristim za obradu nekih prepoznatih izraza.

```
int column=0, newrow=1, row=0;
```

Globalne varijable i brojači: column je broj retka (prepoznate leksičke jedinke u tom redu), newrow je oznaka da li je započeo novi redak ili ne, a row je broj linije koju lexer upravo obrađuje.

```
%}
```

Unaprijed definirane klase regularnih izraza.

```
cpp      #.*
```

Preprocesorska direktiva: počinje sa znakom # i odnosi se do kraja reda

```
digit    [0-9]
```

Znamenka. Može biti bilo koja od 0 do 9, oba uključno

```
letter   [a-zA-Z_]
```

Po ANSI-C standardu dozvoljeno slovo (naprimjer za identifikator, string, itd..) može biti od a do z, od A do Z i povlaka _.

```
hex      [a-fA-F0-9]
```

Dozvoljeni znakovi u heksadecimalnom broju.

```
exponent [Ee][+-]?{digit}+
```

Dozvoljeni znakovi u eksponentu.

```
FS       (f|F|l|L)
```

Znakovi u oznaci tipa (dužine) brojevne konstante

```
IS       (u|U|l|L)*
```

Znakovi u oznaci tipa i predznaka brojevne konstante. Primijetite da se u kasnijim specifikacijama C jezika smiju ponavljati samo do tri puta {IS}

```
include  #include[ \t]*(\<.*\>|\".*\")
```

Nešto složeniji regularni izraz za preprocesorsku direktivu za #include. Kao što znamo, #include može biti oblika: #include "datoteka" ili #include <datoteka>

```
%%

{include}               { count(); include(); }
```

Povećaj broj stupca i obradi klasu include.

```
{cpp}                   { count(); lexd("CPP_DIRECTIVE "); }
```

Povećaj broj stupca i ispiši na ekran leksičku jedinku.

```
"/*"                    { count(); lexd("COMMENT ");
comment(); }
```

U slučaju C komentara imamo poseban tretman. Naime, oni se mogu protezati kroz par redaka, te zato imamo posebnu funkciju koja će "pojesti" takve znakove i vratiti samo jednu leksičku jedinku. Preciznije, neće vratiti jer ih ignoriramo, ali će u ovom slučaju ispisati na ekran.

```
"auto"                  { count(); lexd("AUTO "); }
"break"                 { count(); lexd("BREAK "); }
"case"                  { count(); lexd("CASE "); }
"char"                  { count(); lexd("CHAR "); }
"const"                 { count(); lexd("CONST "); }
"continue"              { count(); lexd("CONTINUE "); }
"default"               { count(); lexd("DEFAULT "); }
"do"                    { count(); lexd("DO "); }
"double"                { count(); lexd("DOUBLE "); }
"else"                  { count(); lexd("ELSE "); }
"enum"                  { count(); lexd("ENUM "); }
"extern"                { count(); lexd("EXTERN "); }
"float"                 { count(); lexd("FLOAT "); }
"for"                   { count(); lexd("FOR "); }
"goto"                  { count(); lexd("GOTO "); }
"if"                    { count(); lexd("IF "); }
"int"                   { count(); lexd("INT "); }
"long"                  { count(); lexd("LONG "); }
"register"              { count(); lexd("REGISTER "); }
"return"                { count(); lexd("RETURN "); }
"short"                 { count(); lexd("SHORT "); }
"signed"                { count(); lexd("SIGNED "); }
"sizeof"                { count(); lexd("SIZEOF "); }
"static"                { count(); lexd("STATIC "); }
"struct"                { count(); lexd("STRUCT "); }
"switch"                { count(); lexd("SWITCH "); }
"typedef"               { count(); lexd("TYPEDEF "); }
"union"                 { count(); lexd("UNION "); }
```

```
"unsigned"              { count(); lexd("UNSIGNED "); }
"void"                  { count(); lexd("VOID "); }
"volatile"              { count(); lexd("VOLATILE "); }
"while"                 { count(); lexd("WHILE "); }
```

Niz rezerviranih riječi.

```
{letter}({letter}|{digit})*  { count(); lexd("IDENTIFIER
"); }
```

Identifikator. Mora započeti sa slovom, te može nastaviti sa proizvoljno puno slova ili brojeva.

```
0[xX]{hex}+{IS}?           { count(); lexd("CONSTANT "); }
0{digit}+{IS}?             { count(); lexd("CONSTANT "); }
{digit}+{IS}?              { count(); lexd("CONSTANT "); }
{letter}?'(\\.|[^\\'])+'   { count(); lexd("CONSTANT "); }
```

Cjelobrojne konstante: heksadecimalna, oktalna, decimalna. Slovna konstanta.

```
{digit}+{exponent}{FS}?               { count();
lexd("CONSTANT "); }
```

Brojevna konstanta sa eksponentom.

```
{digit}*"."{digit}+({exponent})?{FS}? { count();
lexd("CONSTANT "); }
{digit}+"."{digit}*({exponent})?{FS}? { count();
lexd("CONSTANT "); }
```

Konstante sa pomičnim zarezom.

```
{letter}?\"(\\.|[^\\"])*\"      { count();
lexd("STRING_LITERAL "); }
```

String konstanta: niz slova omeđen dvostrukim navodnicima.

```
"..."                   { count(); lexd("ELLIPSIS "); }

">>="                   { count(); lexd("RIGHT_ASSIGN "); }
"<<="                   { count(); lexd("LEFT_ASSIGN "); }
"+="                    { count(); lexd("ADD_ASSIGN "); }
"-="                    { count(); lexd("SUB_ASSIGN "); }
"*="                    { count(); lexd("MUL_ASSIGN "); }
"/="                    { count(); lexd("DIV_ASSIGN "); }
```

```
"%="                    { count(); lexd("MOD_ASSIGN "); }
"&="                    { count(); lexd("AND_ASSIGN "); }
"^="                    { count(); lexd("XOR_ASSIGN "); }
"|="                    { count(); lexd("OR_ASSIGN "); }

">>"                    { count(); lexd("RIGHT_OP "); }
"<<"                    { count(); lexd("LEFT_OP "); }
"++"                    { count(); lexd("INC_OP "); }
"--"                    { count(); lexd("DEC_OP "); }
"->"                    { count(); lexd("PTR_OP "); }
"&&"                    { count(); lexd("AND_OP "); }
"||"                    { count(); lexd("OR_OP "); }
"<="                    { count(); lexd("LE_OP "); }
">="                    { count(); lexd("GE_OP "); }
"=="                    { count(); lexd("EQ_OP "); }
"!="                    { count(); lexd("NE_OP "); }

";"                     { count(); lexd("; "); }
("{"|"<% ")             { count(); lexd("{ "); }
("}"|"%> ")             { count(); lexd("} "); }
","                     { count(); lexd(", "); }
":"                     { count(); lexd(": "); }
"="                     { count(); lexd("= "); }
"("                     { count(); lexd("( "); }
")"                     { count(); lexd(") "); }
("["|"<: ")             { count(); lexd("[ "); }
("]"|":> ")             { count(); lexd("] "); }
"."                     { count(); lexd(". "); }
"&"                     { count(); lexd("& "); }
"!"                     { count(); lexd("! "); }
"~"                     { count(); lexd("~ "); }
"-"                     { count(); lexd("- "); }
"+"                     { count(); lexd("+ "); }
"*"                     { count(); lexd("* "); }
"/"                     { count(); lexd("/ "); }
"%"                     { count(); lexd("%% "); }
"<"                     { count(); lexd("< "); }
">"                     { count(); lexd("> "); }
"^"                     { count(); lexd("^ "); }
"|"                     { count(); lexd("| "); }
"?"                     { count(); lexd("? "); }
```

Operatori, zagrade, elipsa, itd..

```
[ \t\v\f]             { count(); }
```

Ignoriraj tabulatore, oznake za vertikalni ili horizontalni pomak, itd.

```
"\r\n"              { count(); ++newrow; fprintf(stderr,
"\n"); }
"\n"                { count(); ++newrow; fprintf(stderr,
"\n"); }
```

U slučaju prelaska u slijedeći red, "pojedi ga" i povećaj brojač redaka.

```
.                   { fprintf(stderr, "bad: %c\n",
yytext[0]); }
```

Znak koji nije "upao" u niti jednu klasu. Izvadi ga iz toka i ispiši.

%%

```
int yywrap(void)
{
 return 1;
}
```

Lex wrapper. Znači da nema više datoteka za prepoznavanje.

```
void comment(void)
{
  int c;

  while (1)
  {
    while ((c=input())!='*' && c!=EOF)  /* eat up text of
comment */
      if (c=='\n') ++row;
    if (c=='*')
    {
      while ((c=input())=='*');
      if (c=='/')
        break;    /* found the end */
    }

    if (c==EOF)
    {
      /* error("EOF in comment"); */
      break;
    }
  }
}
```

Funkcija za vađenje komentara iz toka. Počinje sa /* i završava sa EOF ili */. U slučaju EOF (kraj datoteke) prijavljuje grešku. Koristi input() makro samog Flexa za vađenje iz internog buffera i streama.

```
void count(void)
{
 int i;

 for (i=0; yytext[i]!='\0'; i++)
   if (yytext[i]=='\n')
     column=0;
   else
     if (yytext[i]=='\t')
       column+=8-(column%8);
     else
       column++;

  if (newrow)
  {
    fprintf(stderr, "%d: ", row+=newrow);
    newrow=0;
  }
}
```

Brojač stupca. Tabulator tretiramo kao 8 znakova.

```
void include(void)
{
  char *c;
  if ((c=index(yytext, '<'))==NULL)
    *index(c=index(yytext, '"')+1, '"')=0;
  else
    *index(++c, '>')=0;
  fprintf(stderr, "CPP_INCLUDE %s", c);
}
```

Jednostavni kod za ispis CPP include datoteka. U stvarnom slučaju naš bi lexer morao imati "stacking" include datoteka radi rekurzivnog pozivanja i prepoznavanja, i pomoću yy_switch_buffer() i slično.

```
int main(int argc, char **argv)
{
  if (argc>1)
    yyin=fopen(argv[1], "r");
  else
    yyin=stdin;
  yylex();
```

```
}
```

Trivijalni glavni program.

Primjer izvršavanja ovog lexera bio bi na jednoj tipičnoj ulaznoj datoteci:

```
/* This is comment */
/**************  Too *************/

#include <stdio.h>
#include <stdlib.h>
#include "nema.h"
#define NOTHING NULL
int main(void)
{
  long myhex=0xdeadbeefL, burek;
  srandom(time(NULL)+myhex%getpid());
  burek=1+(int)(0666*(float)random()/(RAND_MAX+1.0));
  printf("Hello %s #%d.\n", "world", burek);
}
```

Ovakav:

```
1: COMMENT
2: COMMENT
3:
4: CPP_INCLUDE stdio.h
5: CPP_INCLUDE stdlib.h
6: CPP_INCLUDE nema.h
7: CPP_DIRECTIVE
8: INT IDENTIFIER ( VOID )
9: {
10: LONG IDENTIFIER = CONSTANT , IDENTIFIER ;
11: IDENTIFIER ( IDENTIFIER ( IDENTIFIER ) + IDENTIFIER %
IDENTIFIER ( ) ) ;
12: IDENTIFIER = CONSTANT + ( INT ) ( CONSTANT * ( FLOAT )
IDENTIFIER ( ) / ( IDENTIFIER + CONSTANT ) ) ;
13: IDENTIFIER ( STRING_LITERAL , STRING_LITERAL ,
IDENTIFIER ) ;
14: }
```