UCS 1602 - Compiler Design

Exercise 2 - Implementation of Lexical Analyzer using LEX tool

Name:	Mahesh Bharadwaj K
Reg No:	185001089
Semester:	VI
Date:	February 9, 2021

Aim:

To create lexical analyser using LEX tool.

Program

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

pre_process ^#(.)*
line_comment    \/\/(.)*
multi_comment   \/\*(.|\n)*\*\/

keyword
↪  auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long

id          [a-zA-Z_]([a-zA-Z0-9_])*
function    {id}\((.)*\)

realConst (\+|\-)?[1-9][0-9]*\.[0-9]+
intConst (\+|\-)?[1-9][0-9]*
charConst   \'[a-zA-Z]\'
stringConst    \"[a-z A-Z]*\"

assignOp =
bitwiseOp "^"|"&"|"|"|"<<"|">>"
arithAssignOp "+="|"-="|"*="|"\="|"%="
relOp <|<=|>|>=|=|!=
arithOp "+"|"-"|"*"|"/"|"%"
logicOp &&|\|\||!
separators ";"|","|"."|"["|"]"|"("|")"|"{"|"}"|"["|"]"

/*printf("  | %25s | %-25s |\n", yytext, "Function call");*/
%%
{keyword}       {printf("  | %25s | %-25s |\n", yytext, "Keyword");}
{function}      {printf("  | %25s | %-25s |\n", yytext, "Function call");}
{id}            {printf("  | %25s | %-25s |\n", yytext, "Identifier");}
{realConst}     {printf("  | %25s | %-25s |\n", yytext, "Real const");}
{intConst}      {printf("  | %25s | %-25s |\n", yytext, "Integer Constant");}
{bitwiseOp}     {printf("  | %25s | %-25s |\n", yytext, "Bitwise Operator");}
{assignOp}      {printf("  | %25s | %-25s |\n", yytext, "Assignment Operator");}
{arithAssignOp} {printf("  | %25s | %-25s |\n", yytext, "Arith Assign Operator");}
```

```
{arithOp}       {printf("  | %25s | %-25s |\n", yytext, "Arithmetic Operator");}
{logicOp}       {printf("  | %25s | %-25s |\n", yytext, "Logical Operator");}
{relOp}         {printf("  | %25s | %-25s |\n", yytext, "Relational Operator");}
{charConst}     {printf("  | %25s | %-25s |\n", yytext, "Character Constant");}
{stringConst}   {printf("  | %25s | %-25s |\n", yytext, "String Constant");}
{separators}    {printf("  | %25s | %-25s |\n", yytext, "Seperators");}
{pre_process}   {printf("  | %25s | %-25s |\n", yytext, "Preprocessor Directive");}
{line_comment}  {printf("  | %25s | %-25s |\n", yytext, "Line comment");}
{multi_comment} {
    char *lines = strtok(yytext, "\n");
    while(lines){
        printf("  | %25s | ", lines);
        lines = strtok(NULL, "\n");
        printf("%-25s |\n",(lines!=NULL)?" ": "Multiline Comment");}
    }
.|\n {}
%%
int main(int argc, char **argv)
{
    if(argc != 2){
        fprintf(stderr,"Please Enter file as second argument!\n");
        return 1;
    }
    yyin = fopen(argv[1], "rt");
    if(yyin == NULL){
        fprintf(stderr,"File not found!\n");
        return 1;
    }
    printf("  +--------------------------+--------------------------+\n");
    yylex();
    printf("  +--------------------------+--------------------------+\n");

}
```

Output

Figure 1: **Sample Program file**

lexer.l sample.txt ×

Assignment-02 > sample.txt

```
#include <stdio.h>
//single line comment
main() {
    int a = 10,b = -20;
    float f = -10.23, g = 6.89;
    char c = 'a';
    char arr[] = "abcd";
/*
Multiline comment
in the source file
to test the code
*/
    if ( a > b )
        printf("a is greater");
    else
        printf("b is greater");
    a += 5;
    f >> 2;
}
```

Figure 2: **Lexical analysis output**

```
mahesh@mahesh-PC:~/Repositories/Compiler-Design/Assignment-02

❯
❯ ./lexer.out sample.txt
 +---------------------------+---------------------------+
 |        #include <stdio.h> | Preprocessor Directive    |
 |     //single line comment | Line comment              |
 |                     main() | Function call             |
 |                         { | Seperators                |
 |                       int | Keyword                   |
 |                         a | Identifier                |
 |                         = | Assignment Operator       |
 |                        10 | Integer Constant          |
 |                         , | Seperators                |
 |                         b | Identifier                |
 |                         = | Assignment Operator       |
 |                       -20 | Integer Constant          |
 |                         ; | Seperators                |
 |                     float | Keyword                   |
 |                         f | Identifier                |
 |                         = | Assignment Operator       |
 |                    -10.23 | Real const                |
 |                         , | Seperators                |
 |                         g | Identifier                |
 |                         = | Assignment Operator       |
 |                      6.89 | Real const                |
 |                         ; | Seperators                |
 |                      char | Keyword                   |
 |                         c | Identifier                |
 |                         = | Assignment Operator       |
 |                       'a' | Character Constant        |
 |                         ; | Seperators                |
 |                      char | Keyword                   |
 |                       arr | Identifier                |
 |                         [ | Seperators                |
 |                         ] | Seperators                |
 |                         = | Assignment Operator       |
 |                    "abcd" | String Constant           |
 |                         ; | Seperators                |
 |                        /* |                           |
 |         Multiline comment |                           |
 |        in the source file |                           |
 |          to test the code |                           |
 |                        */ | Multiline Comment         |
 |                        if | Keyword                   |
 |                         ( | Seperators                |
 |                         a | Identifier                |
 |                         > | Relational Operator       |
 |                         b | Identifier                |
 |                         ) | Seperators                |
 |    printf("a is greater") | Function call             |
 |                         ; | Seperators                |
 |                      else | Keyword                   |
 |    printf("b is greater") | Function call             |
 |                         ; | Seperators                |
 |                         a | Identifier                |
 |                        += | Arith Assign Operator     |
 |                         5 | Integer Constant          |
 |                         ; | Seperators                |
 |                         f | Identifier                |
 |                        >> | Bitwise Operator          |
 |                         2 | Integer Constant          |
 |                         ; | Seperators                |
 |                         } | Seperators                |
 +---------------------------+---------------------------+
```

Learning Outcomes

1. We learn to write regular expressions to identify tokens
2. We learn to combine regular expressions to identify complex tokens like functions
3. We learn to work with LEX tool to match regular expressions
4. We learn to design lexical analyser using LEX tool