Laboratory-1

QUESTION-1

Build a lexical analyser that ignores white spaces & comments.

```
%{
#include <stdio.h>
#include <string.h>
%}
%%
\n?"/*"(.|\n)*"*/"\n ;
"//".*\n ;
[ ]+ {fprintf(yyout," ");}
[\t]+ {fprintf(yyout,"\t");}
[\n]+ {fprintf(yyout,"\n");}
. ECHO;
%%
int main()
{
    yyin = fopen("sample.c","r");
    yyout = fopen("cleaned.c","w");
    yylex();
    fclose(yyout);
    fclose(yyin);
    return 0;
}
int yywrap()
{
    return 1;
}
```

Input (sample.c):

```
#include<stdio.h>

int main()
{
    printf("Hello"); // first hello
    /*
        Let's try multi line comments
        now
        here
    */
    printf("Welcome to LEX tool !");
}
```

Output (cleaned.c):

```
#include<stdio.h>
int main()
{
 printf("Hello");
  printf("Welcome to LEX tool !");
}
```

Result: The white space and comment ignoring lex code has been implemented successfully.

QUESTION-2

Build a lexical analyser to analyse keywords and identifiers in C.

```
%{
    #include<stdio.h>
    #include<string.h>
%}
%%
if|for|while|else|do|int|"long
long"|short|float|double|include|main|return|printf|"<stdio.h>"
{printf("keyword %s \n",yytext);}
[_a-zA-Z$][_a-zA-Z0-9]* {
    if(yyleng <= 32){
        printf("identifier %s \n",yytext);
    }
    else{
        printf("Identifier size exceeded the max length\n");
    }
}
[0-9]+ {
    if(yyleng < 10 || (yyleng == 10 && yytext < "2147483648")){
        printf("Numeric value %s \n",yytext);
    }
    else{
        printf("Numeric value exceeded INT_MAX\n");
    }
}
. ;
%%
int yywrap(){
    return 1;
}
int main(){
    yyin = fopen("sample.c","r");
    yylex();
    fclose(yyin);
    return 0;
}
```

Input (sample.c):

```
#include <stdio.h>
int main()
{
    int a = 5;
    if(a > 0)
    {
        while(a--)
            printf("Good Day\n");
    }
    else
    {
        for(int i=0;i<a;i++)
            printf("May Day\n");
    }
    printf("Hello");
    return 0;
}
```

Output:

```
prajw@Prajwal_DELL MINGW64 ~/OneDrive/Desktop/NITT/Sem6/CSPC62 - Compilers/Lab-1/Q2
$ ./a.exe
#
include is a keyword.
<stdio.h> is a keyword.


int is a keyword.
main is a keyword.()
{

int is a keyword. a = 5;

if is a keyword.(a > 0)
    {

while is a keyword.(a--)

printf is a keyword.("
Good is an valid identifier.
Day is an valid identifier.\n");
    }

else is a keyword.
    {

for is a keyword.(
int is a keyword. i=0;i<a;i++)

printf is a keyword.("
May is an valid identifier.
Day is an valid identifier.\n");
    }
```

```
for is a keyword.(
int is a keyword. i=0;i<a;i++)

printf is a keyword.("
May is an valid identifier.
Day is an valid identifier.\n");
    }

printf is a keyword.("
Hello is an valid identifier.");

return is a keyword. 0;
}
prajw@Prajwal_DELL MINGW64 ~/OneDrive/Desktop/NITT/Sem6/CSPC62 - Compilers/Lab-1/Q2
$
```

Result:

The keywords and identifiers analysing lex code has been implemented successfully.

QUESTION-3

Build a lex program for identifying operators in C.

```
%{
    #include<stdio.h>
    #include<string.h>
%}
%%
">" {printf("Greater Than >\n");}
"<" {printf("Lesser Than <\n");}
"=" {printf("assigning =\n");}
"+" {printf("Addition +\n");}
"-" {printf("Subtraction -\n");}
"/" {printf("Division /\n");}
"*" {printf("Multiplication *\n");}
"%" {printf("Modulo %%\n");}
"!=" {printf("Not equal !=\n");}
"==" {printf("Comparision ==\n");}
">=" {printf("Greater Than equals to >=\n");}
"<=" {printf("Greater Than equals to <=\n");}
. ;
%%
int yywrap(){
    return 1;
}
int main(){
    yyin = fopen("sample.c","r");
    yylex();
    fclose(yyin);
    return 0;
}
```

Input (sample.c):

```
#include<stdio.h>

int main() {
    int a;
    printf("Hello world");
    int x = 5, y = 7;
}
```

Output:

```
prajw@Prajwal_DELL MINGW64 ~/OneDrive/Desktop/NITT/Sem6/CSPC62 - Compilers/Lab-1/Q3
$ ./a.exe
Lesser Than <
Greater Than >




assigning =
assigning =
```

Result:

Lex Code to analyse relational operators has been implemented successfully.