Week 4 Assignment

1. Implementation of lexical analyzer using LEX for recognizing the following tokens:

• A minimum of 10 keywords of your choice

• Identifiers with the regular expression : letter(letter | digit)*

• Integers with the regular expression: digit+

• Relational operators: <, >, <=, >=, ==, !=

• Ignores everything between multi line comments (/* …. */)

• Storing identifiers in symbol table

• Using files for input and output.

Program:

```
%{
   #include<stdio.h>
   #include<string.h>
   int un_used=0;
   int check_index;
   int symbol_table_size=30;
   struct SymbolTable{
      char lexeme[20];
      char token_type[30];
   };
   struct SymbolTable Table[30];
   int symbol_tabe_ptr=0;
   void insert_into_symbol_table(char *lexeme,char *type){
      if(symbol_tabe_ptr<symbol_table_size){
         strcpy(Table[symbol_tabe_ptr].lexeme,lexeme);
         strcpy(Table[symbol_tabe_ptr++].token_type,type);
      }
   }
   int check(char* lexeme){
```

```
        for(int i=0;i<symbol_table_size;i++){
            if(strcmp(Table[i].lexeme,lexeme)==0){
                return i;
            }
        }
        return -1;
    }
%}
%%
"#include "\s*\<[a-z|\.]*\> {
    fprintf(yyout,"%s : is a Pre-processor directive\n",yytext);
}
[ \n\t\s]+ {
    un_used++;
}
\/\/(.*) {
    un_used++;
}
\/\*(.|\n|\t)*\*\/ {
    un_used++;
}
"int"|"main"|"void"|"return"|"if"|"else"|"for"|"else if"|"break"|"while"|"char" {
    fprintf(yyout,"%s : is an Keyword\n",yytext);
    check_index=check(yytext);
    if(check_index==-1){
        insert_into_symbol_table(yytext,"Keyword");
    }
}
[a-z][a-zA-Z0-9]* {
    fprintf(yyout,"%s : is a Identifier\n",yytext);
```

```
    check_index=check(yytext);
    if(check_index==-1){
        insert_into_symbol_table(yytext,"Identifier");
    }
}
[0-9]+ {
    fprintf(yyout,"%s : is an Integer\n",yytext);
    check_index=check(yytext);
    if(check_index==-1){
        insert_into_symbol_table(yytext,"Integer");
    }
}
"<"|">"|"<="|"<="|"=="|"!=" {
    fprintf(yyout,"%s : is an Relational Operator\n",yytext);
    check_index=check(yytext);
    if(check_index==-1){
        insert_into_symbol_table(yytext,"Relational Operator");
    }
}
"("|")"|"{"|"}"|";"|"," {
    fprintf(yyout,"%s : is a Delimiter\n",yytext);
    check_index=check(yytext);
    if(check_index==-1){
        insert_into_symbol_table(yytext,"Delimiter");
    }
}
"="|"+="|"-="|"*="|"/=" {
    fprintf(yyout,"%s is an Assignment Operator\n",yytext);
    check_index=check(yytext);
    if(check_index==-1){
```

```
        insert_into_symbol_table(yytext,"Assignment Operator");
    }
}
"*"|"-"|"/"|"%"|"+" {
    fprintf(yyout,"%s is an Arithmetic Operator\n",yytext);
    check_index=check(yytext);
    if(check_index==-1){
        insert_into_symbol_table(yytext,"Arithhmetic Operator");
    }
}
.    {
    fprintf(yyout,"%s",yytext);
}
%%
int yywrap(){return 1;}
int main(){
    extern FILE *yyin,*yyout;
    char *input_file;
    char *output_file;
    yyin=fopen("input.c","r");
    yyout=fopen("output.txt","w");
    if (!yyin||!yyout){
        printf("Can not open files.\n");
        return 1;
    }
    yylex();
    printf("Contents of symbol table\n");
    for(int i=0;i<symbol_tabe_ptr;i++){
        printf("%s : %s\n",Table[i].lexeme,Table[i].token_type);
    }
```

```
    return 0;
}
```

Input file:

Week4 > C input.c > main()

```c
#include <iostream>
#include <string.h>
// Hello world welocme to compiler desing
int main()
{
    int a = 3;
    int b = 4;
    int c = a * b;
    return 1;
    /* This is a multiline
    comment */
}
```

Output file (.txt):

Week4 > output.txt

```
#include <iostream> : is a Pre-processor directive
#include <string.h> : is a Pre-processor directive
int : is an Keyword
main : is an Keyword
( : is a Delimiter
) : is a Delimiter
{ : is a Delimiter
int : is an Keyword
a : is a Identifier
= is an Arithmetic Operator
3 : is an Integer
; : is a Delimiter
int : is an Keyword
b : is a Identifier
= is an Arithmetic Operator
4 : is an Integer
; : is a Delimiter
int : is an Keyword
c : is a Identifier
= is an Arithmetic Operator
a : is a Identifier
* is an Arithmetic Operator
b : is a Identifier
; : is a Delimiter
return : is an Keyword
1 : is an Integer
; : is a Delimiter
} : is a Delimiter
```

2. Write a C Program to Scan and Count the number of characters, words, and lines in a file.

Program:

```
#include <stdio.h>
int main()
{
   char filename[100];
   printf("Enter the name of the file: ");
   scanf("%s", filename);
   FILE *file = fopen(filename, "r");
   if (file == NULL)
   {
      printf("Unable to open the file. Exiting...\n");
      return 1;
   }
   int no_of_chars = 0;
   int no_of_words = 0;
   int no_of_lines = 0;
   int in_word = 0;
   char ch;
   while ((ch = fgetc(file)) != EOF)
   {
      no_of_chars++;
      if (ch != ' ' && ch != '\t' && ch != '\n' && ch != '\r' && ch != '\f' && ch != '\v')
      {
         in_word = 1;
      }
      if ((ch == ' ' || ch == '\t' || ch == '\n' || ch == '\r' || ch == '\f' || ch == '\v') &&
         in_word)
      {
```

```
            no_of_words++;
            in_word = 0;
        }
        if (ch == '\n' || ch == '\0')
        {
            no_of_lines++;
        }
    }
    fclose(file);
    printf("Number of characters: %d\n", no_of_chars);
    printf("Number of words: %d\n", no_of_words);
    printf("Number of lines: %d\n", no_of_lines);
    return 0;
}
```

Input:

```
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type
specimen book. It has survived not only five centuries, but also the leap
into electronic typesetting, remaining essentially unchanged.
It was popularised in the 1960s with the release of Letraset sheets containing
Lorem Ipsum passages, and more recently with desktop publishing software like
Aldus PageMaker including versions of Lorem Ipsum.
```

Ouptut:

```
PS D:\SRM AP\SEM 5\Compiler design Lab\Week4> .\count_chars_words_lines
Enter the name of the file: input2.txt
input2.txtNumber of characters: 581
Number of words: 90
Number of lines: 7
PS D:\SRM AP\SEM 5\Compiler design Lab\Week4>
```