PRACTICAL FILE

BE (CSE) 6th Semester

COMPILER DESIGN (CS654)

March 2021 – May 2021

Submitted By

Jatin Ghai
(UE183042)

Submitted To

Dr. Akashdeep
Assistant Professor, Computer Science and Engineering

Computer Science and Engineering
University Institute of Engineering and Technology
Panjab University, Chandigarh – 160014, INDIA
2021

Experiment No. 3

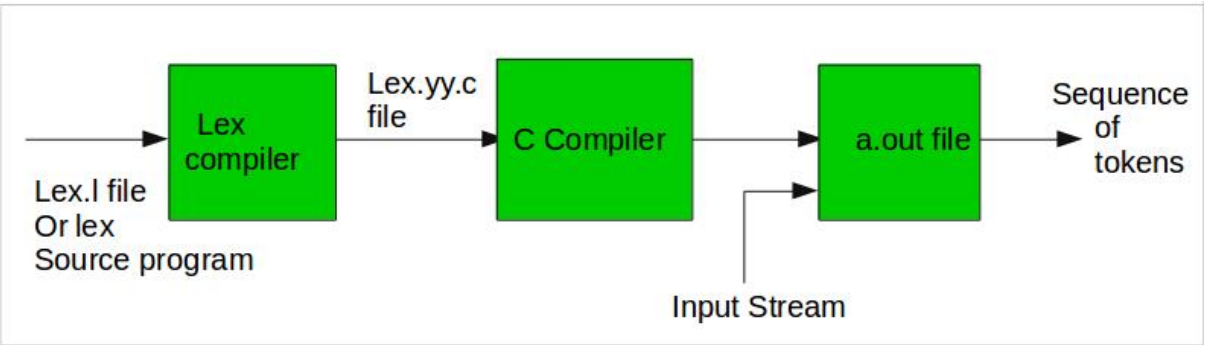
AIM :- To Design and implement following LEX codes

Flex (Fast Lexical Analyzer Generator) :
Flex is a computer program that generates lexical analyzers. Instead of writing a scanner from scratch, you only need to identify the regular expression for that particular token in a language.

An lex file (*.l) file is provided to the flex tool, which then generates a "lex.yy.c" file which is later compiled using GCC compiler. The compiler generated an out file called "a.out". This file is then executed to generate tokens from the input stream.

yylex() is automatically generated by the flex when it is provided with a .l file and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.

There is a version of Flex for windows called win-flex. I have used win-flex to implement the following programs.



1) Converting upper case letters to lowercase letters

SOURCE CODE:

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

```
%}
%%
([A-Z]) {fprintf(yyout,"%c",yytext[0]+32);}
([. ,\n]) { fprintf(yyout,"%c",yytext[0]);}
%%
int  yywrap(){}
int main(){

    extern FILE *yyin, *yyout;

    yyin = fopen("uppercase input file.txt", "r");
    yyout = fopen("uppercase ouput file.txt", "w");

    yylex();
    return 0;
}
```

Input :

```
This IS A TEXT IN UPper CASE MostLY.
CONVert IT alL to LOWERcase;
Some NUmbers 98JSSY*$B&J
```

Output :

```
this is a text in upper case mostly.
convert it all to lowercase;
some numbers 98jssy*$b&j
```

2) Count the number of words

SOURCE CODE :

```
%{
#include<stdio.h>
#include<string.h>
int count = 0;
%}
%%
([A-Za-z0-9])* {count ++;}
([;,.* ,\n]) { ; }
%%
int  yywrap(){}
int main(){

    extern FILE *yyin, *yyout;

    yyin = fopen("Input.txt", "r");
    yyout = fopen("Output.txt", "w");

    yylex();
    printf("%d",count);
    fprintf(yyout,"%d",count);
    return 0;
}
```

Input :

```
This file is to count number of words;
Some numbers 98234;
Some alphanumeric text 2334hjk43
```

Output :

```
15
```

3) Count words greater than 10 and truncate the rest of the portion

SOURCE CODE :

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void sub(char *);
int count =0;
%}

%%
([A-Za-z0-9])* { sub(yytext); }
([. ,\n]) {fprintf(yyout,"%s",yytext);}
%%
int  yywrap(){ return 1;}
int main(){

    extern FILE *yyin, *yyout;

    yyin = fopen("Input.txt", "r");
    yyout = fopen("Output.txt", "w");

    yylex();
    fprintf(yyout,"\nCount : %d",count);
    return 0;
}

void sub(char * s){
    int len = strlen(s);
    if(len > 10 ){
        count++;
        len = 10;
    }
    for(int i = 0; i< len; i++)
        fprintf(yyout,"%c",s[i]);
}
```

Input :

```
Averylongtext anotherone 123alphalong
innextlinemainly ys
```

Output :

```
Averylongt anotherone 123alphalo
innextline ys
Count : 3
```

4) Add line number of lines to the program

SOURCE CODE :

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int line_no = 0;
%}

%%
([\n]) { line_no++; fprintf(yyout,"%s%d ",yytext,line_no);}
. {fprintf(yyout,"%s",yytext);}
%%
int  yywrap(){ return 1;}
int main(){

    extern FILE *yyin, *yyout;

    yyin = fopen("Input.txt", "r");
    yyout = fopen("Output.txt", "w");

    line_no++; fprintf(yyout,"%d ",line_no);

    yylex();
    return 0;
}
```

Input :

```
#include<iostream.h>
using namespace std;
int main(){
    int a, b;
    a = 5; b = 6;
    cout << a+b << endl;
    cout << "Printing something" << endl;
```

```
}
```

Output :

```
#include<iostream.h>
using namespace std;
int main(){
    int a, b;
    a = 5; b = 6;
    cout << a+b << endl;
    cout << "Printing something" << endl;
}
```

5) Identify all numbers from the program

SOURCE CODE :

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

%%
(['\n' , ';']) {}
([A-Za-z][0-9a-zA-z]*) { }

([0-9]*['.'][0-9]+) { fprintf(yyout,"%s\n",yytext); }

[-]?([0-9])+ { fprintf(yyout,"%s\n",yytext); }

. {}
%%
int  yywrap(){ return 1;}

int main(){

    extern FILE *yyin, *yyout;

    yyin = fopen("Input.txt", "r");
    yyout = fopen("Output.txt", "w");

    yylex();
    return 0;
}
```

Input :

```
#include<iostream.h>
using namespace std;
int main(){
    int a, b;
```

```
    a = -5; b =6;
    double a7 = 5.7898;
    double b67g78 =8.;
    if(a == 9) cout << "67"<<endl;
    cout << a+b << endl;
    cout << 9*7<<endl;
    cout << "Printing something" << endl;
}
```

Output :

```
-5
6
5.7898
8
9
67
9
7
```

6) To generate all the tokens

SOURCE CODE :

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

%%
include|break|continue|using|namespace|main|std|cout|cin|endl {
printf("Keywords %s\n",yytext); }

int|double|string|float|short|long { printf("Datatype
%s\n",yytext); }

[<].*[>] { printf("Libraries %s\n",yytext); }

"=="|"++"|"+="|"-="|"--"|"*="|"**"|"<<"|">>"|"%="|"="|"-"|"+"|"%"|"
*" { printf("Operators %s\n",yytext); }

([A-Za-z][0-9a-zA-z]*) { printf("Identifiers %s\n",yytext); }

([0-9]*['.'][0-9]+) { printf("Literal %s\n",yytext); }

([0-9]+) { printf("Literal %s\n",yytext); }

["].*["] { printf("Literal %s\n",yytext); }

","|"("|")"|"{"|"}"|"#"|";" { printf("punctuator %s\n",yytext); }

. { ; }
%%

int  yywrap(){ return 1;}

int main(){

    extern FILE *yyin, *yyout;
```

```
    yyin = fopen("Input.txt", "r");
    yyout = fopen("Output.txt", "w");

    yylex();
    return 0;
}
```

Input :

```
#include<iostream.h>
using namespace std;
int main(){
    int a, b;
    a = -5; b =6;
    double a7 = 5.7898;
    double b67g78 =8.;
    if(a == 9) cout << "67"<<endl;
    cout << a+b << endl;
    cout << 9*7<<endl;
    cout << 8-1 <<endl;
    cout << "Printing something" << endl;
}
```

Output :

```
punctuator #
Keywords include
Libraries <iostream.h>
Keywords using
Keywords namespace
Keywords std
punctuator ;
Datatype int
Keywords main
punctuator (
punctuator )
punctuator {
```

```
Datatype int
Identifiers a
punctuator ,
Identifiers b
punctuator ;
Identifiers a
Operators =
Operators -
Literal 5
punctuator ;
Identifiers b
Operators =
Literal 6
punctuator ;
Datatype double
Identifiers a7
Operators =
Literal 5.7898
punctuator ;
Datatype double
Identifiers b67g78
Operators =
Literal 8
punctuator ;
Identifiers if
punctuator (
Identifiers a
Operators ==
Literal 9
punctuator )
Keywords cout
Operators <<
Literal "67"
Operators <<
Keywords endl
punctuator ;
Keywords cout
Operators <<
Identifiers a
Operators +
```

```
Identifiers b
Operators <<
Keywords endl
punctuator ;
Keywords cout
Operators <<
Literal 9
Operators *
Literal 7
Operators <<
Keywords endl
punctuator ;
Keywords cout
Operators <<
Literal 8
Operators -
Literal 1
Operators <<
Keywords endl
punctuator ;
Keywords cout
Operators <<
Literal "Printing something"
Operators <<
Keywords endl
punctuator ;
punctuator }
```

Learning from the experiment:

1. I learned about the Lex tool and its application in token detection.
2. It is easy to build lexical analyser using lex tool