System Programming Practical File

Name – SONU

Examination Roll No. – 20009570049

Roll No. – 2002058

Semester – Vth

Year – 2022-23

1. Write a Lex program to count the number of lines and characters in the input file.

Source Code: -

```
%{
#include<stdio.h>
int sc=0,wc=0,lc=0,cc=0;
%}

%%
[\n] { lc++; cc+=yyleng;}
[  \t] { sc++; cc+=yyleng;}
[^\t\n ]+ { wc++;  cc+=yyleng;}
%%

int main(int argc ,char* argv[ ])
{
        printf("Enter the input:\n");
        yylex();
        printf("The number of lines=%d\n",lc);
        printf("The number of spaces=%d\n",sc);
        printf("The number of words=%d\n",wc);
        printf("The number of characters are=%d\n",cc);
}

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

Output: -

```
Enter the input:
Abhineet is good poet
The number of lines=1
The number of spaces=3
The number of words=4
The number of characters are=22
```

2. Write a Lex program that implements the Caesar cipher: it replaces every letter with the one three letters after in

alphabetical order, wrapping around at Z. e.g. a is replaced by d, b by e, and so on z by c.

Source Code: -

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

%%
[a-z] {
            char ch = yytext[0];
            ch += 3;

            if(ch > 'z')
            ch -= ('z'+1 - 'a');

      printf("%c",ch);
            yytext[0] = ch;

    }

[A-Z] {
            char ch = yytext[0];
            ch += 3;

            if(ch > 'Z')
            ch -= ('Z'+1 - 'A');

      printf("%c",ch);
            yytext[0] = ch;

    }
%%

int main(){
        yyin = fopen("input.txt","r");
        yylex();
        return 0;
}

yywrap(){}
yyerror(){}
```

Output: -

```
Abhi or Kavi
Dekl ru Ndyl
^C
```

3. Write a Lex program that finds the longest word (defined as a contiguous string of upper- and lower-case letters) in the input.

Source Code: -

```
%{

  //to find longest string and its length

   #include<stdio.h>

   #include<string.h>

   int longest = 0;

   char longestString[30];
%}
%%
[a-zA-Z]+ {

if(yyleng>longest){

   longest = yyleng;

   strcpy(longestString,yytext);
}
}


%%



int main(void){

   yylex();

   printf("The longest string is %s \n", longestString);

   printf("Length of a longest string is %d \n",longest);
}
```

Output: -

```
Abhineet is happy with this output

The longest string is Abhineet
Length of a longest string is 8
```

4. Write a Lex program that distinguishes keywords, integers, floats, identifiers, operators, and comments in any simple programming language.

Source Code: -

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

%}

%%
(void|int|float|double|string|return|bool|char|for|if|do|while|exit("[0-
9]+")"|case|break|continue|switch|enum|struct|"size of")  { printf("\n%s -> ",yytext);
                                                          printf("%s\n","Keyword");}

[0-9]+ {printf("\n%s -> ",yytext);
      printf("%s\n","Integer");}
[0-9]*(.)[0-9]+('f'|'d'){1} {printf("\n%s -> ",yytext);
            printf("%s\n","Float");}
[_a-zA-Z]+[_a-zA-Z0-9]* { printf("\n%s -> ",yytext);
                    printf("%s\n","Identifier");}


[+*/&<>|()-=] { printf("\n%s -> ",yytext);
          printf("%s\n","Operator");}

"//".* {printf("\n%s -> ",yytext);
          printf("%s\n","Single Line Comment");}

"/*".*"*/" {printf("\n%s -> ",yytext);
          printf("%s\n","Multi Line Comment");}

\t ;
\n ;
" ";

. {     printf("\n%s -> ",yytext);
       printf("%s\n","Special Character");}
%%

int main(){
          yylex();
          return 0;
}
```

```
yywrap(){}
yyerror(){}
```

Output: -

```
int a=4;

int -> Keyword

  -> Special Character

a -> Identifier

= -> Operator

4 -> Integer

; -> Operator
```

5. Write a Lex program to count the number of identifiers in a C file.

Source Code: -

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

%%
"int" |
"float" |
"char" |
"double" |
"bool" {
          char ch;
          ch = input();

          while(1){
                if(ch == ','){
                      num++;
                }

                if(ch == ";"){
                      num++;
                      break;
                }
```

```
                    if(ch == '\n')
                    break;

                    ch = input();
            }
}

.|'\n' ;
%%

int main(int argc,char *argv[]){

            if(argc!=2){
                    printf("\n\tYou didn't specify file in arguement\n");

                    exit(0);
   }

            else{
     yyin = fopen(argv[1],"r");

     if(yyin){
       yylex();
       printf("\nNo. of Identifiers = %d\n",num);
     }

     else{
        printf("\n\tError while opening file\n");
     }
            }

            return 0;

}

yywrap(){}

yyerror(){}
```

Output: -

```
No. of Identifiers = 27
```

6. Write a Lex program to count the number of words, characters, blank spaces and lines in a C file.

Source Code: -

```
%{
        #include<stdio.h>
        #include<strings.h>
        #include<stdlib.h>
  int yyflex();
        int lines=0,words=0,Characters=0,blank=0;
%}

%%
[^ \t\n]+ {words++;
      Characters+=yyleng;}
[\n] {lines++;}

" " blank++;
\t  blank+=5;
%%

int main(){
        yyin = fopen("input.txt","r");

        yylex();


  printf("\n\t=============== This file contains ==============\n");
  printf("\tTotal %d Lines\n",lines);
  printf("\tTotal %d words\n",words);
  printf("\tTotal %d Characters \n",Characters);
  printf("\tTotal %d Blanks \n",blank);



return 0;
}

 yywrap(){}

yyerror(){
printf("\nError\n");
exit(0);
}
```

Output: -

```
=============== This file contains ==============
Total 0 Lines
Total 0 words
Total 0 Characters
Total 0 Blanks
```

7. Write a Lex specification program that generates a C program which takes a string "abcd" and prints the following output.

abcd

abc

ab

a

Source Code: -

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


%}

%%
[a-zA-Z]*  {
            for(int i=yyleng-1;i>=0;i--){
                printf("\t\t");
        for(int j=0;j<=i;j++)
        printf("%c\t",yytext[j]);

        printf("\n");
            }

    }

%%

int main(){
        yylex();

        return 0;
}

int yywrap(){
  return 0;
}

int yyerror(){
        printf("\n******* ERROR *******\n");
        exit(1);
}
```

Output: -

```
abcd
        a       b       c       d
        a       b       c
        a       b
        a
```

8. A program in Lex to recognize a valid arithmetic expression.

Source Code: -

```
/* Lex program to recognize valid arithmetic expression
                and identify the identifiers and operators */
%{
#include <stdio.h>
#include <string.h>
        int operators_count = 0, operands_count = 0, valid = 1, top = -1, l = 0, j = 0;
        char operands[10][10], operators[10][10], stack[100];
%}
%%
"(" {
        top++;
        stack[top] = '(';
}
"{" {
        top++;
        stack[top] = '{';
}
"[" {
        top++;
        stack[top] = '[';
}
")" {
        if (stack[top] != '(') {
                valid = 0;
        }
        else if(operands_count>0 && (operands_count-operators_count)!=1){
                valid=0;
        }
        else{
                top--;
                operands_count=1;
                operators_count=0;
        }
}
"}" {
        if (stack[top] != '{') {
```

```
                valid = 0;
        }
        else if(operands_count>0 && (operands_count-operators_count)!=1){
                valid=0;
        }
        else{
                top--;
                operands_count=1;
                operators_count=0;
        }
}
"]" {
        if (stack[top] != '[') {
                valid = 0;
        }
        else if(operands_count>0 && (operands_count-operators_count)!=1){
                valid=0;
        }
        else{
                top--;
                operands_count=1;
                operators_count=0;
        }

}
"+"|"-"|"*"|"/" {
        operators_count++;
        strcpy(operators[l], yytext);
        l++;
}
[0-9]+|[a-zA-Z][a-zA-Z0-9_]* {
        operands_count++;
        strcpy(operands[j], yytext);
        j++;
}
%%


int yywrap()
{
        return 1;
}
int main()
{
        int k;
        printf("Enter the arithmetic expression: ");
        yylex();

        if (valid == 1 && top == -1) {
                printf("\nValid Expression\n");
        }
        else
```

```
                printf("\nInvalid Expression\n");

        return 0;
}
```

Output: -

```
Enter the arithmetic expression: a-b*c


Valid Expression
```

9. Write a YACC program to find the validity of a given expression (for operators + - * and /)

Source Code (Lex): -

```
%{
/* 4a.l Yacc Program to check the validity of an arithmetic Expression that uses operators +, -, *, /
*/
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)?                { return NUM;}
[a-zA-Z_][_a-zA-Z0-9]*   { return ID; }
[\t]                             ;
\n                               return 0;
.                                return yytext[0];
%%
yywrap()
{}
```

Source Code (Yacc): -

```
%{
/* 4a.y Yacc Program to check the validity of an arithmetic Expression that uses operators +, -, *, /
*/
#include<stdio.h>
#include<stdlib.h>
%}
%token NUM  ID
%left '+' '-'
%left '*' '/'
%%
```

```
e : e '+' e
|   e '-' e
|   e '*' e
|   e '/' e
|   '('e')'
| NUM
| ID        ;
%%
main()
{
printf(" Type the Expression & Press Enter key\n");
yyparse();
printf(" Valid Expression \n");
}
yyerror()
{
printf(" Invalid Expresion!!!!\n"); exit(0);
}
```

Output: -

```
a+b
```

10. A Program in YACC which recognizes a valid variable which starts with letter followed by a digit. The letter should be in lowercase only.

Source Code (Lex): -

```
%{
#include"y.tab.h"
%}

%%
[a-zA-Z] {return LETTER;}
[0-9] {return DIGIT;}
[_] {return UND;}
[\n] {return NL;}
. {return yytext[0];}
%%
```

Source Code (Yacc): -

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

%token DIGIT LETTER UND NL

%%
stmt: variable NL {printf("valid identifiers\n"); exit(0);}
;
variable: LETTER alphanumeric
;
alphanumeric: LETTER alphanumeric | DIGIT alphanumeric | UND alphanumeric | LETTER | DIGIT | UND
;

%%

int yyerror(char *msg)
{
 printf("Invalid variable\n");
 exit(0);
}

main()
{
 printf("enter the variable: \n");
 yyparse();
}
```

Output: -

```
Var1 = 1;
```

11. A Program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division).

Source Code (Lex): -

```
%{

        #include<stdio.h>
        #include<stdlib.h>
        #include "y.tab.h"

        int yylval;
```

```
%}

%%

[0-9]+ {yylval = atoi(yytext);
       return NUM;}

[\t]+   ;

\n  {return 0;}

.   {return yytext[0];}

%%
```

Source Code (Yacc): -

```
%{

            #include<stdio.h>
            #include<stdlib.h>
            #include "y.tab.h"

%}

%token NUM
%left '+' '-'
%left '/' '*'
%left '(' ')'

%%
expr:e{printf("Result is :: %d\n",$$);
      return 0;}

e:e '+' e{$$ = $1+$3;}
 |e '-' e{$$ = $1-$3;}
 |e '*' e{$$ = $1*$3;}
 |e '/' e{

  if($3==0){
  printf("\nDivision By Zero\n");
  printf("Result is :: Undefined");
  return 0;

  }
  else
  $$ = $1/$3;}

 |'(' e ')'{$$ = $2;}
```

```
 |NUM {$$ = $1;}

%%

int main(){
   printf("\nEnter the arithmetic expression ::");

            yyparse();
            printf("\nValid Expression\n");

            return 0;

}

int yywrap(){
            return 0;
}

int yyerror(){
            printf("\nInvalid Expression\n");
            exit(1);
}
```

Output: -

```
46+35
```

12. Program in YACC to recognize the strings “ab”, “aabb”, “aaabbb”, … of the language ($a^n b^n$, n>=1).

Source Code (Lex): -

```
%{
            #include<stdio.h>
            #include<stdlib.h>
            #include "y.tab.h"


%}

%%

[a] {return A;}
[b] {return B;}
\n {return NL;}
```

```
. {return yytext[0];}


%%
```

Source Code (Yacc): -

```
%{

        #include<stdio.h>
        #include<stdlib.h>
        #include "y.tab.h"
%}

%token A B NL

%%
expr : S NL{printf("\nValid String\n");
        return 0;}
S : A S B
  |;

%%

int main(){
        printf("\nEnter the string :: ");
        yyparse();
        return 0;
}

yywrap(){}
yyerror(){
        printf("\nInvalid String");
}
```

Output: -

```
aaaaaaaabbbbbb
```

13. Program in YACC to recognize the language ($a^n b$, n>=10). (Output to say input is valid or not)

Source Code (Lex): -

```
%{
        #include<stdio.h>
        #include<stdlib.h>
        #include "y.tab.h"
%}

%%
[a] {return A;}
[b] {return B;}
\n  {return NL;}
. {return yytext[0];}
%%
```

Source Code (Yacc): -

```
%{
        #include<stdio.h>
        #include<stdlib.h>
        #include "y.tab.h"

%}

%token A B NL

%%
S : A A A A A A A A A A S1 B NL
   { printf("\nValid String \n");
     return 0;}

S1 : A S1
   |;

%%

main(){

        printf("\nEnter a String :: ");
        yyparse();

}

yywrap(){}

yyerror(){
        printf("\nInvalid String\n");
        return 0;
}
```

Output: -

```
aaaaaaaaaaab
```