EXPERIMENT NO:1 Design and Implement Lexical Analyzer Using C

```
#include <stdbool.h>
 #include <stdio.h>
 #include <string.h>
 #include <stdlib.h>
 #include<ctype.h>
 bool isDelimiter(char ch){
 if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == ',' || ch ==
';' || ch == '>' ||
 ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch ==
'{' || ch == '}')
 return (true);
 return (false);
 }
 bool isOperator(char ch){
 if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '>' || ch == '<' || ch ==
'=')
 return (true);
 return (false);
 }
 bool validIdentifier(char* str){
 if (isdigit(str[0]) || isDelimiter(str[0]) == true)
 return (false);
}
 return (true);
 bool isKeyword(char* str){
 if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") || !strcmp(str,
"do") ||
 !strcmp(str, "break") || !strcmp(str, "continue") || !strcmp(str, "int") ||
 !strcmp(str, "double")
 || !strcmp(str, "float")|| !strcmp(str, "return") || !strcmp(str, "char")
 || !strcmp(str, "case") || !strcmp(str, "char") || !strcmp(str, "sizeof") ||
 !strcmp(str, "long")
 || !strcmp(str, "short") || !strcmp(str, "typedef") || !strcmp(str, "switch") ||
 !strcmp(str, "unsigned")
 || !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") ||
 !strcmp(str, "goto"))
 return (true);
 return (false);
 }
 char* subString(char* str, int left, int right){
 int i;
 char* subStr = (char*)malloc(sizeof(char) * (right- left + 2));
```

```
for (i = left; i <= right; i++)
subStr[i- left] = str[i];
subStr[right- left + 1] = '\0';
return (subStr);
}
void parse(char* str){
int left = 0, right = 0;
int len = strlen(str);
while (right <= len && left <= right) {
if (isDelimiter(str[right]) == false)
right++;
if (isDelimiter(str[right]) == true && left == right) {
if (isOperator(str[right]) == true)
printf("'%c' IS AN OPERATOR\n", str[right]);
right++;
left = right;
} else if (isDelimiter(str[right]) == true && left != right
|| (right == len && left != right)) {
char* subStr = subString(str, left, right- 1);
if (isKeyword(subStr) == true)
printf("'%s' IS A KEYWORD\n", subStr);
else if (validIdentifier(subStr) == true
&& isDelimiter(str[right- 1]) == false)
printf("'%s' IS A VALID IDENTIFIER\n", subStr);
else if (validIdentifier(subStr) == false
&& isDelimiter(str[right- 1]) == false)
printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);
left = right;
}
}
return;
}
int main()
{
char str[100] = "int sum = number1 + number2; ";
printf("\n%s\n\n",str);
parse(str);
return (0);
}
```

EXPERIMENT NO:2 Implement Lexical Analyzer Using LEX

```
%{
 int COMMENT=0;
 %}
 identifier [a-zA-Z][a-zA-Z0-9]*
 %%
 #.* {printf("\n%s is a preprocessor directive",yytext);}
 7
int |
 float |
 char |
 double |
 while |
 for |
 struct |
 typedef |
 do |
 if |
 break |
 continue |
 void |
 switch |
 return |
 else |
 goto {printf("\n\t%s is a keyword",yytext);}
 "/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
 {identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
 \{ {if(!COMMENT)printf("\n BLOCK BEGINS");}
 \} {if(!COMMENT)printf("BLOCK ENDS ");}
 {identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
 \".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
 [0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
 \)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
 \( ECHO;
 = {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
 \<= |
 \>= |
 \< |
 == |
 \> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
 %%
 int main(int argc, char **argv)
 {
```

```
FILE *file;
file=fopen("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

EXPERIMENT NO:3 LEX Program to DIsplay Number of Lines and Words

```
%{
 %}
 %%
 int chars=0,words=0,lines=0;
 [^\t\n ]+ {words++;
 chars=chars+yyleng;}
 [ ]* {chars++;}
 [\n] {lines++;
 chars++;}
 %%
 int main(){
 yyin = fopen("input.txt","r");
 yylex();
 fclose(yyin);
 printf("\nwords = %d",words);
 printf("\ncharacters = %d",chars);
 printf("\nlines = %d\n",lines);
 return 0;
 }
```

EXPERIMENT NO:4 LEX Program to Convert the Substring abc to ABC

```
%{
 %}
 %%
 [a-zA-Z]* {
 for(int i=0;i<yyleng-2; i++){
 if(yytext[i] == ’a’ && yytext[i+1] == ’b’ && yytext[i+2] == ’c’){
 yytext[i] = ’A’;
 yytext[i+1] = ’B’;
 yytext[i+2] = ’C’;
 }
 }
 printf("%s\n",yytext);
 }
 %%
 int main(){
 yylex();
 return 0;
 }
```

EXPERIMENT NO:5 LEX Program to find the Number of Vowels and Consonant

```
%{
 %}
 int vowels=0,cons=0;
 %%
 [aeiouAEIOU] { vowels++;}
 [a-zA-Z] { cons++; }
 %%
 int main(){
 yylex();
 printf("\nno of vowels = %d",vowels);
 printf("\nno of consonants = %d\n",cons);
 return 0;
 }
```

EXPERIMENT NO:6 Generate YACC Specification to Recognize a Valid Arithmetic Expression

```
%{
 %}
 #include<stdio.h>
 #include<stdlib.h>
 %token NUMBER ID NL
 %left '+' '-'
 %left '*' '/'
 %left '(' ')'
 %%
 valid : e NL{printf("\n valid expression!\n");}
 e : e '+' e
 | e '-' e
 | e '*' e
 | e '/' e
 | '('e')'
 | NUMBER
 | ID ;
 %%
 int main()
 {
 printf("\n Enter an expression: ");
 yyparse();
 }
 int yyerror(char *s)
 {
 printf("\n invalid expression\n");
 exit(1);
 }
 %{
 %}
 LEX Code
 #include "y.tab.h"
 int yylval;
%%
 [0-9]+ {yylval=atoi(yytext);
 return NUMBER;}
 [a-zA-Z]+ {return ID;}
 [\t]+;
 \n {return NL;}
 . {return yytext[0];}
 %%
 int yywrap()
```

```
{
return 1;
}
```

EXPERIMENT NO:7 Generate YACC Specification to Recognize a Valid Identifier

YACC Code

```
%{
%}
#include<stdio.h>
#include<stdlib.h>
%token DIGIT LETTER NL UND
%%
valid : id NL {printf("\n valid identifier!\n");}
id : LETTER alphanum;
alphanum : LETTER alphanum
| DIGIT alphanum
| UND alphanum
| LETTER
| DIGIT
| UND ;
%%
int main()
{
printf("\n Enter an identifier: ");
yyparse();
}
int yyerror()
{
printf("\n invalid identifier\n");
exit(1);
}
%{
%}
%%
```

LEX Code

```
#include "y.tab.h"
[0-9] {return DIGIT;}
[a-zA-Z] {return LETTER;}
[\n] {return NL;}
[_] {return UND;}
. {return yytext[0];}
%%
int yywrap()
{
return 1;
}
```

EXPERIMENT NO:8 Implementation of Calculator Using LEX and YACC

YACC Code

```
%{
%}
#include<stdio.h>
int flag=0;
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
valid : e{printf("\n Result=%d\n",);}
e : e '+' e {=1+3;}
| e '-' e {=1−3;}
| e '*' e {=1∗3;}
| e '/' e {=1/3;}
| e '%' e {=1|'('e')'=2;}
| NUMBER {=
```

LEX Code

```
%{
%}
%%
#include<stdio.h>
#include "y.tab.h"
int yylval;
[0-9]+ {yylval=atoi(yytext);
return NUMBER;}
[\t];
\n {return 0;}
. {return yytext[0];}
%%
int yywrap()
{
return 1;
}
```

EXPERIMENT NO:9 Program to Find ε-Closure of All States of NFA

```
#include<stdio.h>
 #include<string.h>
 char result[20][20], copy[3], states[20][20];
 void add_state(char a[3], int i) {
 strcpy(result[i], a);
 }
 void display(int n) {
 int k = 0;
 printf("Epsilon closure of %s = { ", copy);
 while (k < n) {
 printf(" %s", result[k]);
 k++;
 }
 printf(" }\n");
 }
 int main() {
 FILE * INPUT;
 INPUT = fopen("input.dat", "r");
 char state[3];
 int end, i = 0, n, k = 0;
 char state1[3], input[3], state2[3];
 printf("Enter the no of states: ");
 scanf("%d", & n);
 printf("Enter the states :");
 for (k = 0; k < 3; k++) {
 scanf("%s", states[k]);
 }
 for (k = 0; k < n; k++) {
 i = 0;
 strcpy(state, states[k]);
 strcpy(copy, state);
 add_state(state, i++);
 while (1) {
 end = fscanf(INPUT, "%s%s%s", state1, input, state2);
if (end == EOF) {
 break;
 }
 if (strcmp(state, state1) == 0) {
 if (strcmp(input, "e") == 0) {
 add_state(state2, i++);
 strcpy(state, state2);
 }
```

```
}
}
display(i);
rewind(INPUT);
}
return 0;
}
```

EXPERIMENT NO:10 Program to Find First and Follow of Any Grammar

```
#include<stdio.h>
 #include<math.h>
 #include<string.h>
 #include<ctype.h>
 #include<stdlib.h>
 int n,m=0,p,i=0,j=0;
 char a[10][10],f[10];
 void follow(char c);
 void first(char c);
 int main(){
 int i,z;
 char c,ch;
 //clrscr();
 printf("Enter the no of prooductions:\n");
 scanf("%d",&n);
 printf("Enter the productions:\n");
 for(i=0;i<n;i++)
 scanf("%s%c",a[i],&ch);
 do{
 m=0;
 printf("Enter the elemets whose fisrt & follow is to be found:");
 scanf("%c",&c);
 first(c);
 printf("First(%c)={",c);
 for(i=0;i<m;i++)
 printf("%c",f[i]);
 printf("}\n");
strcpy(f," ");
 m=0;
 follow(c);
 printf("Follow(%c)={",c);
 for(i=0;i<m;i++)
 printf("%c",f[i]);
 printf("}\n");
 printf("Continue(0/1)?");
 scanf("%d%c",&z,&ch);
 }while(z==1);
 return(0);
 }
 void first(char c)
 {
```

```
int k;
if(!isupper(c))
f[m++]=c;
for(k=0;k<n;k++)
{
if(a[k][0]==c)
{
if(a[k][2]==''')follow(a[k][0]);elseif(islower(a[k][2]))f[m + +] =
a[k][2];elsefirst(a[k][2]);voidfollow(charc)if(
'
;
}
for(i=0;i<n;i++)
{
for(j=2;j<strlen(a[i]);j++)
{
if(a[i][j]==c)
{
if(a[i][j+1]!='\0')
first(a[i][j+1]);
if(a[i][j+1]=='\0' && c!=a[i][0])
follow(a[i][0]);
}
}
}
```

EXPERIMENT NO:11 Design and Implement Recursive Descent Parser

```
#include<stdio.h>
 #include<string.h>
 #include<ctype.h>
 char input[10];
 int i, error;
 void E();
 void T();
 void Eprime();
 void Tprime();
 void F();
 int main()
 {
 i = 0;
 error = 0;
 printf("Enter an arithmetic expression : "); // Eg: a+a*a
 scanf("%s",input);
 E();
 if (strlen(input) == i && error == 0)
 printf("\nAccepted..!!!\n");
 else printf("\nRejected..!!!\n");
 return 0;
 }
 void E()
 {
 T();
 Eprime();
 }
 void Eprime()
 {
 if (input[i] == '+')
 {
 i++;
 T();
 Eprime();
 }
 }
 void T()
 {
F();
 Tprime();
 }
 void Tprime()
```

```
{
if (input[i] == '*')
{
i++;
F();
Tprime();
}
}
void F()
{
if (isalnum(input[i]))i++;
else if (input[i] == '(')
{
i++;
E();
if (input[i] == ')')
i++;
else error = 1;
}
else error = 1;
}
```

EXPERIMENT NO:12 Construct Shift Reduce Parser

```
#include<stdio.h>
 #include<string.h>
 int k = 0, z = 0, i = 0, j = 0, c = 0;
 char a[16], ac[20], stk[15], act[10];
 void check();
 int main()
 {
 puts("GRAMMAR is E→E+E \n E→E*E \n E→(E) \n E→id");
 puts("enter input string ");
 scanf("%s", a);
 c = strlen(a);
 strcpy(act, "SHIFT→");
 puts("stack \t input \t action");
 for (k = 0, i = 0; j < c; k++, i++, j++)
 {
 if (a[j] == 'i' && a[j + 1] == 'd')
 {
 stk[i] = a[j];
 stk[i + 1] = a[j + 1];
 stk[i + 2] = '\0';
 a[j] = ' ';
 a[j + 1] = ' ';
 printf("\ncheck();elsestk[i] = a[j];stk[i + 1] =' ';a[j] =' ';printf("
 %s\t%sæ check();printf("");voidcheck()strcpy(ac,"REDUCETOE");for(z = 0;z < c;z + +)if(stk[z]
==' i'stk[z
 %s\t%sæ j+ +;for(z = 0;z < c;z ++)if(stk[z] ==' E'stk[z +1] ==' +'stk[z +2] ==' E')stk[z] =' E';stk[z
+1] =
 %s\t%sæ i= i −2;for(z = 0;z < c;z ++)if(stk[z] ==' E'stk[z +1] ==' ∗'stk[z +2] ==' E')stk[z] ='
E';stk[z +1] =
 %s\t%sæ i= i −2;for(z = 0;z < c;z ++)if(stk[z] ==' ('stk[z +1] ==' E'stk[z +2] ==')')stk[z] =' E';stk[z
+ 1] =
 %s\t%s
```

EXPERIMENT NO:13 Implementation of Back-end Compiler

```
#include<stdio.h>
 #include<string.h>
 void main() {
 char icode[10][30], str[20], opr[10];
 int i = 0;
 printf("\nEnter the set of intermediate code (terminated by exit):\n");
 do {
 scanf("%s", icode[i]);
 }
 while (strcmp(icode[i++], "exit") != 0);
 printf("\nTarget code generation");
 printf("\n********************");
 i = 0;
 do {
 strcpy(str, icode[i]);
 switch (str[3]) {
 case '+':
 strcpy(opr, "ADD");
 break;
 case '-':
 strcpy(opr, "SUB");
 break;
 case '*':
 strcpy(opr, "MUL");
 break;
 case '/':
 strcpy(opr, "DIV");
 break;
 }
 printf("\n\tMov %c,R%d", str[2], i);
 printf("\n\t%s %c,R%d", opr, str[4], i);
 printf("\n\tMov R%d,%c", i, str[0]);
} while (strcmp(icode[++i], "exit") != 0);
}
```