RAJASTHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY
(2018-2022)

Compiler Design

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted To
Mr. Mukesh Choudhary
Dept. Of CSE

Submitted By
Nitish K. Mahto
18ERECS047
BATCH A2 (Vth sem)

SN	List of Experiments
1	Introduction: Objective, scope and outcome of the course.
2	To identify whether given string is keyword or not.
3	Count total no. of keywords in a file. [Taking file from user]
4	Count total no of operators in a file. [Taking file from user]
5	Count total occurrence of each character in a given file. [Taking file from user]
6	Write a C program to insert, delete and display the entries in Symbol Table.
7	Write a LEX program to identify following: 1. Valid mobile number 2. Valid url 3. Valid identifier 4. Valid date (dd/mm/yyyy) 5. Valid time (hh:mm:ss)
8	Write a lex program to count blank spaces,words,lines in a given file.
9	Write a lex program to count the no. of vowels and consonants in a C file.
10	Write a YACC program to recognize strings aaab,abbb using a^nb^n, where b>=0.
11	Write a YACC program to evaluate an arithmetic expression involving operators +,-,* and /.
12	Write a YACC program to check validity of a strings abcd,aabbcd using grammar a^nb^nc^md^m, where n , m>0
13	Write a C program to find first of any grammar.

INTRODUCTION

OBJECTIVE:

This laboratory course is intended to make the students experiment on the basic techniques of compiler construction and tools that can used to perform syntax-directed translation of a high-level programming language into an executable code. Students will design and implement language processors in C by using tools to automate parts of the implementation process. This will provide deeper insights into the more advanced semantics aspects of programming languages, code generation, machine independent optimizations, dynamic memory allocation, and object orientation.

OUTCOMES:

Upon the completion of Compiler Design practical course, the student will be able to:

1. Understand the working of lex and YACC compiler for debugging of programs.

2. Understand & define the role of lexical analyzer, use of regular expression & transition diagrams.

3. Understand and use Context free grammar, and parse tree construction.

4. Learn & use the new tools and technologies used for designing a compiler.

5. Develop program for solving parser problems.

6. Learn how to write programs that execute faster

EXPERIMENT 1

Objective : To identify whether given string is keyword or not.

Introduction : Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier

Code:

```
#include <stdio.h>
#include <string.h> // string file for strcmp function

char kwds[32][10] = {"auto", "double", "int", "struct", "break", "else", "long",      "switch", "case",
                     "enum", "register", "typedef", "char", "extern", "return", "union", "const",
                     "float", "short", "unsigned", "continue",        "for", "signed", "void",
                     "default", "goto", "sizeof", "volatile", "do", "if", "static", "while"};



int isKeyword(char word[10])
{
        for(int i = 0 ; i < 32; i++)
                if(!strcmp(word, kwds[i])) // return zero if equal
                        return 1;
        return 0;
}



int main()
{
        char string[10];
        printf("Enter the String : ");
        scanf("%s", string);

        if(isKeyword(string))
                printf("%s is a keyword\n", string);
        else
                printf("%s is not a keyword\n", string);
        return 0;
}
```

OUTPUT:

Case 1:

```
Enter the String : float
float is a keyword
```

Case 2:

```
Enter the String : while
while is a keyword
```

Case 3:

```
Enter the String : new
new is not a keyword
```

EXPERIMENT 2

Objective : *Count total no of keywords in file(user)*

Introduction :

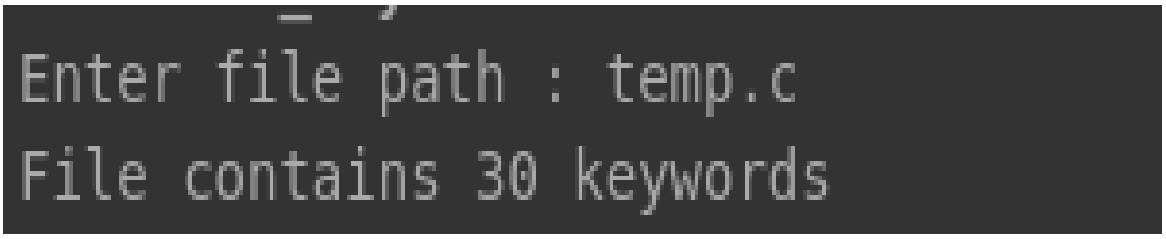
Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>
#include<string.h>
int main ()
{
        char file [50] = "access modifier in class.cpp";
        printf("Enter File name : ");
        gets(file);
        printf("\n");

        FILE * fp = fopen(file, "r");
        FILE * kout = fopen("keyword.txt", "w");
        FILE * iout = fopen("identifier.txt", "w");

        if (fp == NULL)
        {
                printf("\n'%s' file not found...\n", file);
                getch();
                return 1;
        }

        int k, result, count, kcount;
        char c, str[10];
        kcount = count = 0;

        char keywords[][10] = {"auto", "break", "case", "char", "const", "continue",
        "default","do", "double", "else", "enum", "extern", "float", "for", "goto", "if",
        "int","long", "register", "return", "short", "signed", "sizeof", "static",
        "struct","switch", "typedef", "union", "unsigned", "void", "volatile", "while" };

        while((c = fgetc(fp)) != EOF)
                { if(c == ' ' || c == '\n')
                        { ++count;
```

```
            printf("%d). %s\n", count, str);
        for(k=0; k<32; k++)
        {
        result = strcmp(keywords[k], str);
        if (result == 0)
        {
        ++kcount;
        printf(str);
        printf("\n");
        break;
        }
        }

        if (result == 0)
        fprintf(kout, "%s\n", str);
        else
        fprintf(iout, "%s\n", str);

        strcpy(str,"");
        }
        else
        {
        if (isalpha(c) && c!=' ')
        strncat(str, &c, 1);
        }
            }

            fclose(fp);
            fclose(kout);
            fclose(iout);

            printf("\n\t...'%s' file has %d words in it.\n", "keyword.txt", kcount);
        getch();
        return 0;
}
```

Output



Experiment 3

Objective : Count total no of operators in a file(User).

Code:

1. Total Operators.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include<math.h>


int isOp(char c)
{
	FILE *opFile;
	char file[100],  temp;
	// Operator file
	opFile = fopen("operators.txt", "r");

	if (opFile == NULL)
	{
		printf("file not found\n");
		exit(0);
	}

	// Read contents from file
	temp = fgetc(opFile);
	while (temp != EOF)
	{
		if(c == temp)
			return 1;

		temp = fgetc(opFile);
	}
	fclose(opFile);
	return 0;
}


int main()
{
	FILE *fptr;
	int count = 0;
	char file[1000], c;

	printf("Enter the file to open \n");
	scanf("%s", file);
```

```
	// Open User file

	fptr = fopen(file, "r");
	if (fptr == NULL)
	{
		printf("file  Not found\n");
		exit(0);
	}

	// Read contents from file
	c = fgetc(fptr);
	while (c != EOF)
	{
		if(isOp(c))
		{
			count++;
		}
		c = fgetc(fptr);
	}
	printf("The file has total %d operators\n", count);

	return 0;
}
```

2. Operators.txt：

“-+*/=|!%&?><”

3. add.c：

```
int main(int argc, char const *argv[])
{
	int a, b;
	a = 5;
	b = 3;
	printf("%d\n", a+b);
	return 0;
}
```

Output :

```
Enter the file to open
add.c
The file has total 5 operators
```

EXPERIMENT 4

Objective : *Count total occurences of each character.*

Code:

1. Total Occurrences.c

```
#include <stdio.h>
#define MAX 100


int main()
{
  FILE *fp;
  int count = 0;

  char filename[FILE_NAME_SIZE];

  char c;

  printf("Enter the file name\n");
  scanf("%s", filename);

  fp = fopen(filename, "r+");

  if(fp==NULL)
    {
      printf("File not found\n");
      return 0;
    }

  for(c = getc(fp); c != EOF; c = getc(fp))
  {
    count++;
  }

  fclose(fp);

  printf("The file %s has %d characters\n", filename, count);


  return 0;
```

```
}
```

2. add.c

```
int main(int argc, char const *argv[])
{
	int a, b;
	a = 5;
	b = 3;
	printf("%d\n", a+b);
	return 0;
}
```

OUTPUT:

```
Enter the file name
add.c
The file add.c has 102 characters
```

EXPERIMENT 5

Objective : To write a program for implementing symbol table in c

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <math.h>


int main()
{
	// code here
	int i=0, j=0, x=0, n;
	void *add[5], *p;
	char ch, srch, b[15], d[15], c;
	printf("Expression terminated by $:\n");
	while((c!=getchar())!='$')
	{
		b[i] = c;
		// printf("%c\n", c);
		i++;
	}

	n = i-1;

	printf("Given Expression : \n");
	i = 0;
	while(i<=n)
		{
			printf("%c\n", b[i]);
			i++;
		}
	printf("\nSymbol Table\n");
	printf("Symbol\taddr\ttype\n");

	while(j<=n)
	{
		c = b[j];
		if(isalpha(toascii(c)))
		{
```

```
                p=malloc(c);
                add[x]=p;
                d[x]=c;
                printf("\n%c\t%d\tidentifier\n", c,p);
                x++;
                j++;
            }

            else{
                ch = c;
                if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
                {
                    p = malloc(ch);
                    add[x]=p;
                    d[x]=ch;
                    printf("\n%c\t%d\tOperator\n", ch, p);
                    x++;
                    j++;
                }
            }
        }
        return 0;
}
```

EXPERIMENT 6

Objective : **Write a lex prgram for**

1. Valid mobile number
2. Valid URL
3. Valid identifier
4. Valid date (dd/mm/yyyy)
5. Valid time (hh: mm: ss)

Code :-

I) Valid Mobile Number

```
%%
[6-9][0-9]{9} {printf("\nMobile Number Valid\n");}

.+ {printf("\nMobile Number Invalid\n");}

%%

// driver code
int main()
{
    printf("\nEnter Mobile Number : ");
    yylex();
    printf("\n");
    return 0;
}
```

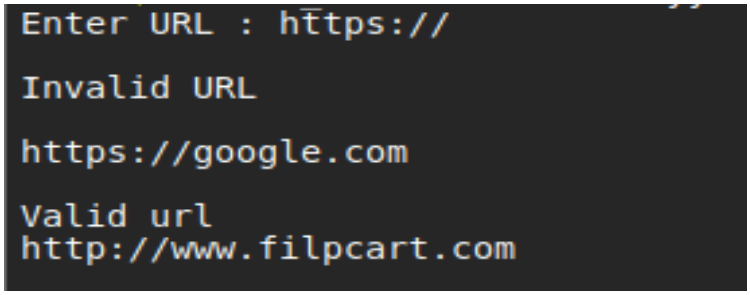
Output

```
Enter Mobile Number : 34343434

Mobile Number Invalid

9954545454

Mobile Number Valid
```

II) Valid URL

```
%%
https?:\/\/(www)?\.?[a-z]*\.[0-1a-zA-Z]* {printf("\nValid url ");}
.+ {printf("\nInvalid URL \n");}


%%


int main(){
	printf("Enter URL : ");
	yylex();
	printf("\n");
	return 0;
}
```

OUTPUT



III) Valid Identifier

```
%%
[_a-zA-Z][0-9a-zA-Z_]* {printf("\nValid Identifier ");}
.+ {printf("\nInvalid Identifier \n");}
```

```
%%


int main(){
        printf("Enter Identifier : ");
        yylex();
        printf("\n");
        return 0;
}
```

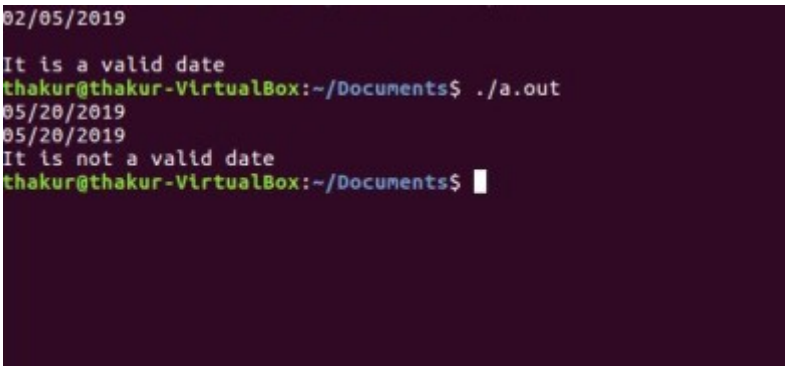
OUTPUT

```
Enter Identifier : _a

Valid Identifier
dkfjkld

Valid Identifier
-343-43-4

Invalid Identifier
```

IV) Valid Date

```
%{
   #include<stdio.h>
   int i=0, yr=0, valid=0;
%}

/* Rule Section */
%%
([0-2][0-9]|[3][0-1])\/((0(1|3|5|7|8))|(10|12)) \/([1-2][0-9][0-9][-0-9]) {valid=1;}

([0-2][0-9]|30)\/((0(4|6|9))|11) \/([1-2][0-9][0-9][0-9]) {valid=1;}

([0-1][0-9]|2[0-8])\/02 \/([1-2][0-9][0-9][0-9]) {valid=1;}

29\/02\/([1-2][0-9][0-9][0-9]) { while(yytext[i]!='/')i++; i++;
       while(yytext[i]!='/')i++;i++;
       while(i<yyleng)yr=(10*yr)+(yytext[i++]-'0');
       if(yr%4==0||(yr%100==0&&yr%400!=0))valid=1;}

%%

main()
{
  yyin=fopen("vpn.txt", "r");
  yylex();
  if(valid==1) printf("It is a valid date\n");
  else printf("It is not a valid date\n");
}
 int yywrap()
{
 return 1;
}
```

Output



V) Valid Time

```
%{
   #include<stdio.h>
   int i=0, yr=0, valid=0;
%}

/* Rule Section */
%%
([0-2][0-9]|[3][0-1])\/((0(1|3|5|7|8))|(10|12))
            \/([1-2][0-9][0-9][-0-9]) {valid=1;}

([0-2][0-9]|30)\/((0(4|6|9))|11)
         \/([1-2][0-9][0-9][0-9]) {valid=1;}

([0-1][0-9]|2[0-8])\/02
            \/([1-2][0-9][0-9][0-9]) {valid=1;}

29\/02\/([1-2][0-9][0-9][0-9])
     { while(yytext[i]!='/')i++; i++;
       while(yytext[i]!='/')i++;i++;
       while(i<yyleng)yr=(10*yr)+(yytext[i++]-'0');
       if(yr%4==0||(yr%100==0&&yr%400!=0))valid=1;}
%%

// driver code
main()
{
  yyin=fopen("vpn.txt", "r");
  yylex();
  if(valid==1) printf("It is a valid date\n");
  else printf("It is not a valid date\n");
}
 int yywrap()
{
 return 1;
}
```

OUTPUT

```
02/05/2019

It is a valid date
thakur@thakur-VirtualBox:~/Documents$ ./a.out
05/20/2019
05/20/2019
It is not a valid date
thakur@thakur-VirtualBox:~/Documents$
```

Experiment 7

Objective : Write a lex program to count total numbers of spaces, newlines, words.

Code :

```
// Header
%{
    #include<stdio.h>
    #include<string.h>
    int wrds = 0, spaces=0, newlines=0, chars=0;
    char word[30];
%}
```

// Rules

```
%%
([a-zA-Z0-9.]+) {wrds++;}
([ ]+) {++spaces;}
\n   {++newlines;}
([^a-zA-Z0-9\n. ]+) {printf("%s\n", yytext); }
%%

int main(int ac, char **av)
{
    FILE    *fd;

    if (ac == 2)
    {
        if (!(fd = fopen(av[1], "r")))
        {
            perror("Error: ");
            return (-1);
        }
        yyset_in(fd);
        yylex();
        fclose(fd);
    }
    else
        printf("Usage: a.out filename\n");

    printf("newlines : %d\n", newlines);
    printf("Words : %d\n", wrds);
    printf("spaces : %d\n", spaces);
    return (0);
}
```

File temp.c

```
#include <stdio.h>

int main(int argc, char const *argv[])
{
	printf("%d\n", argc);
	return 0;
}
```

Output :

```
┌─[nitish@Lenovo-G50]─[~/playground/git/labs/CD/lex]
└──╼ $./a.out temp.c
newlines : 9
Words : 17
spaces : 10
```

Experiment 8

Objective : Write a lex program to count the no. of vowels and consonants in a C file.

Code :

count_vowelConsonants.l

```
%option noyywrap

%{
   #include<stdio.h>
   #include<string.h>
   int vow=0, cons=0;
%}


%%
[aeiouAEIOU] {vow++;}
[b-dfghj-np-tv-z] {cons++;}
[.] {}
%%

int main(int ac, char **av)
{
   FILE    *fd;

   if (ac == 2)
   {
      if (!(fd = fopen(av[1], "r")))
      {
         perror("Error: ");
         return (-1);
      }
      yyset_in(fd);
      yylex();
      fclose(fd);
   }

   else
      printf("Usage: a.out filename\n");

   printf("Vowels : %d\n", vow);
   printf("Consonants : %d\n", cons);

   return (0);
```

```
}
```

temp.c

```
#include <stdio.h>

int main(){
	int a,b,c;
	a = 3;
	b = 5;
	c = a+b;
	return 0;
}
```

Output :

```
nitish@Lenovo-G50:~/playground/git/labs/CD/lex$ lex count_vowelConsonants.l && cc lex.yy.c   && ./a.out temp.c

Vowels : 14
Consonants : 23
```

Experiment 9

Objective : Write a YACC program to recognize strings aaab,abbb using a^nb^n, where n>=0.

Code :

1.**gram.y**

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

%token A B NL

%%
stmt: S NL {printf("valid string\n");
          exit(0);}
;
S: A S B |
;
%%

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

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

2. anbn.l

```
%{
/* Definition section */
#include "y.tab.h"
%}

%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%

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

Output :

```
┌─[nitish@Lenovo-G50]─[~/playground/git/labs]
└──╼ $yacc -d gram.y && lex anbn.l && cc y.tab.c lex.yy.c  && ./a.out
y.tab.c: In function ‘yyparse’:
y.tab.c:1115:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]
       yychar = yylex ();
                ^~~~~
y.tab.c:1251:7: warning: implicit declaration of function ‘yyerror’; did you mean ‘yyerrok’? [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~
       yyerrok
gram.y: At top level:
gram.y:23:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
enter the string
aabb
valid string
```

Experiment 10

Objective : Write a YACC program to evaluate an arithmetic expression involving operators +,-,*
and /.

Code :

1. gram.y

```
%{
    #include<stdio.h>
    int flag=0;

%}
%token NUMBER

%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{
         printf("\nResult=%d\n",$$);
         return 0;
        }
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E'*'E {$$=$1*$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}
 |'('E')' {$$=$2;}
 | NUMBER {$$=$1;}
;
%%

void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations
Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\
n");
   yyparse();
  if(flag==0)
   printf("\nEntered arithmetic expression is Valid\n\n");

}
void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
```

```
}
```

2. lex.l

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

%%
[0-9]+ {
        yylval=atoi(yytext);
        return NUMBER;
      }
[\t] ;
[\n] return 0;
. return yytext[0];

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

Output :

```
Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
3+3+34/343%3

Result=6

Entered arithmetic expression is Valid
```

Experiment 11

Objective :-

Write a YACC program to check validity of a strings abcd,aabbcd using grammar a^nb^nc^md^m, where n , m>0

Code:

1. grammar.y

```
%{
/* Definition section */
#include<stdio.h>
#include<stdlib.h>
%}

%token A B C D NL

/* Rule Section */
%%
stmt: S NL { printf("valid string\n");
                        exit(0); }
;
S: S1 S2 |
S1: A S1 B |
S2: C S2 D|
;
%%

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

//driver code
main()
{
printf("enter the string\n");
yyparse();
}
```

2. lex.l

```
%{
/* Definition section */
#include "y.tab.h"
```

```
%}

/* Rule Section */
%%
[aA] {return A;}
[bB] {return B;}
[cC] {return C;}
[dD] {return D;}

\n {return NL;}
. {return yytext[0];}
%%

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

Output :

```
┌─[tcj1@tcj]─[~/playground/git/labs/CD/lex]
└──╼ $lex anbncmdm.l && yacc -d grammar.y  && gcc lex.yy.c y.tab.c -w && ./
a.out
grammar.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]
grammar.y: note: rerun with option '-Wcounterexamples' to generate conflict
 counterexamples
enter the string
aabbccdd
valid string
┌─[tcj1@tcj]─[~/playground/git/labs/CD/lex]
└──╼ $lex anbncmdm.l && yacc -d grammar.y  && gcc lex.yy.c y.tab.c -w && ./
a.out
grammar.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]
grammar.y: note: rerun with option '-Wcounterexamples' to generate conflict
 counterexamples
enter the string
aabbccccddddd
valid string
┌─[tcj1@tcj]─[~/playground/git/labs/CD/lex]
└──╼ $lex anbncmdm.l && yacc -d grammar.y  && gcc lex.yy.c y.tab.c -w && ./
a.out
grammar.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]
grammar.y: note: rerun with option '-Wcounterexamples' to generate conflict
 counterexamples
enter the string
abccd
invalid string
```

Experiment 12

Objective : Write a C program to find first of any grammar.

Code :

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
main()
{
   int i;
   char choice;
   char c;
   char result[20];
   printf("How many number of productions ? :");
   scanf(" %d",&numOfProductions);
   for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T
   {
      printf("Enter productions Number %d : ",i+1);
      scanf(" %s",productionSet[i]);
   }
   do
   {
      printf("\n Find the FIRST of  :");
      scanf(" %c",&c);
      FIRST(result,c); //Compute FIRST; Get Answer in 'result' array
      printf("\n FIRST(%c)= { ",c);
      for(i=0;result[i]!='\0';i++)
      printf(" %c ",result[i]);       //Display result
      printf("}\n");
       printf("press 'y' to continue : ");
      scanf(" %c",&choice);
   }
   while(choice=='y'||choice =='Y');
}
/*
 *Function FIRST:
 *Compute the elements in FIRST(c) and write them
 *in Result Array.
 */



void FIRST(char* Result,char c)
{
```

```
    int i,j,k;
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
    Result[0]='\0';
    //If X is terminal, FIRST(X) = {X}.
    if(!(isupper(c)))
    {
        addToResultSet(Result,c);
              return ;
    }
    //If X is non terminal
    //Read each production
    for(i=0;i<numOfProductions;i++)
    {
//Find production with X as LHS
        if(productionSet[i][0]==c)
        {
//If X → ε is a production, then add ε to FIRST(X).
 if(productionSet[i][2]=='$') addToResultSet(Result,'$');
           //If X is a non-terminal, and X → Y1 Y2 … Yk
           //is a production, then add a to FIRST(X)
           //if for some i, a is in FIRST(Yi),
           //and ε is in all of FIRST(Y1), …, FIRST(Yi-1).
      else
           {
              j=2;
              while(productionSet[i][j]!='\0')
              {
              foundEpsilon=0;
              FIRST(subResult,productionSet[i][j]);
              for(k=0;subResult[k]!='\0';k++)
                 addToResultSet(Result,subResult[k]);
               for(k=0;subResult[k]!='\0';k++)
                   if(subResult[k]=='$')
                   {
                      foundEpsilon=1;
                      break;
                   }
               //No ε found, no need to check next element
               if(!foundEpsilon)
                   break;
               j++;
              }
           }
    }
}
    return ;
}
/* addToResultSet adds the computed
 *element to result set.
 *This code avoids multiple inclusion of elements
```

```
 */

void addToResultSet(char Result[],char val)
{
   int k;
   for(k=0 ;Result[k]!='\0';k++)
       if(Result[k]==val)
           return;
   Result[k]=val;
   Result[k+1]='\0';
}
```

Output :

```
How many number of productions ? :2
Enter productions Number 1 : E=F+id
Enter productions Number 2 : F=(id)|#

 Find the FIRST of  :E

 FIRST(E)= {  ( }
press 'y' to continue : y

 Find the FIRST of  :F

 FIRST(F)= {  ( }
press 'y' to continue :
```