AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

LABORATORY MANUAL

COMPILER DESIGN

SUBJECT CODE: 2170701

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

B.E. 7th SEMESTER

NAME:_____
ENROLLMENTNO:_____
BATCHNO:_____
YEAR: _____

Amiraj College of Engineering and Technology,
Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

CERTIFICATE

This is to certify that Mr. / Ms. _____

Of class_____ Enrolment No _____has

Satisfactorily completed the course in _____as

by the Gujarat Technological University for ____ Year (B.E.) semester___of

Computer Science & Engineering in the Academic year_____.

Date of Submission:-

Faculty Name and Signature
(Prof. Rupali Patel)

Head of Department
(CSE)

AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

B.E. 7th SEMESTER

SUBJECT: COMPILER DESIGN

SUBJECT CODE: 2170701

List Of Practicals

Sr. No.	Title	Date of submission	Sign	Remark
1	Implement a Program for simple lexical analyzer using C language.			
2	Write a program to create Symbol Table for the given input file.			
3	Write a program to implement recursive decent parser for the following grammar. expr -> term + expr / term term -> factor * term / factor factor -> expr / id			
4	Write a C program to test whether a given identifier is valid or not.			
5	To Study about Lexical Analyzer Generator (LEX) and Fast Lexical Analyzer Generator (FLEX).			
6	Write a C program to find first and follow for the given grammar.			

7	Write a C program for implementing the functionalities of predictive parser for the mini language.			
8	Write a CPP program for constructing of LL (1) parsing.			
9	To Study about Yet Another Compiler-Compiler (YACC).			

Practical - 1

Aim: Implement a Program for simple lexical analyzer using C language.

//checking for identifier, constant, keyword, operator

//gives error in identifier, constant, and operator and for braces missing

Pseudo Code:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
	char *keyword[30]={"if","while","for","else","int","float","eof"};
char str[30]Implement a Program for simple lexical analyzer using C language.
{"\0"};
	char temp[30]={"\0"};
	int i=0,j=0,k,f=0;
	clrscr();
	printf("Enter a string");
	gets(str);

	for(i=0;str[i]!='\0';i++)
	{
		j=0;
		f=0;
		while(str[i]==' ')
		{
			i++;
		}
		if(isalpha(str[i]))
		{
```

```
        while(isalnum(str[i]))
        {
                temp[j]=str[i];
                i++;
                j++;
        }
        temp[j]='\0';
        i=i-1;
        for(k=0;strcmp(keyword[k],"eof")!=0;k++)
        {
                if(strcmp(keyword[k],temp)==0)
                {
                        printf("%s is a kwyword\n",temp);
                        f=1;
                        break;
                }
        }
        if(f!=1)
        {
                printf("%s is an identifier\n",temp);
        }
}
else if(isdigit(str[i]))
{
        while(isdigit(str[i]))
        {
                temp[j]=str[i];
                j++;
                i++;
        }
        temp[j]='\0';
```

```
            printf("%s is a number\n",temp);
            i=i-1;
    }
    else if(str[i]=='<' || str[i]=='>' || str[i]=='=' || str[i]=='!')
    {
            j=0;
            if(str[i+1]=='=')
            {
                    temp[j]=str[i];
                    temp[j+1]=str[i+1];
                    i++;
                    printf("%s is a relational operator\n",temp);
            }
            else if(str[i]!='=')
            {
                    printf("%c is a relational operator\n",str[i]);
            }
            else
            {
                    printf("%c is an assignment operator\n",str[i]);
            }
    }
    else if(str[i]=='+' || str[i]=='-' || str[i]=='/' || str[i]=='*')
    {
            if(str[i+1]=='+')
            {
                    temp[j]=str[i];
                    temp[j+1]=str[i+1];
                    i++;
                    printf("%s is increment operator\n",temp);
            }
```

```
                else if(str[i+1]=='-')
                {
                        temp[j]=str[i];
                        temp[j+1]=str[i+1];
                        i++;
                        printf("%s is decrement operator\n",temp);
                }
                else
                {
                        printf("%c is an arithmetic operator\n",str[i]);
                }
        }
        else
        {
                printf("%c is special symbol\n",str[i] );
        }
    }
  getch();
}
```

OUTPUT:

Enter the String: int a, b=5;

int is keyword

a is identifier

, is operator

b is identifier

= is assignment operator

; is operator

Practical - 2

Aim: Write a program to create Symbol Table for the given input file.

Input File:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
char c;
double d,e;
float f;
getch();
}
```

Pseudo Code:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char type[4][7]={"char","int","float","double"},*s=" ,;\n";
int size[4]={1,2,4,8};
void main()
{
  char *token,*token1,currentline[400];
  int i,j=3000;
  FILE *f1;
  clrscr();
  f1=fopen("e:\symbol.txt","r");
  printf("\nDatatype\tVariable\tSize\tAddress\n");
  while((fgets(currentline,1000,f1))!=NULL)
```

```
 {
   token=strtok(currentline,s);
    for(i=0;i<4;i++)
    {
        if(strcmp(token,type[i])==0)
        {
           while((token1=strtok(NULL,s))!=NULL)
           {
             printf("\n%s\t\t%s\t\t%d\t%d",type[i],token1,size[i],j);
             j=j+size[i];
           }
        }
    }
 }
 fclose(f1);
 getch();
}
```

OUTPUT:

```
Datatype        Variable        Size    Address

int             a               2       3000
int             b               2       3002
char            c               1       3004
double          d               8       3005
double          e               8       3013
float           f               4       3021
```

Practical - 3

Aim: Write a program to implement recursive decent parser for the following grammar.

expr -> term + expr / term

term -> factor * term / factor

factor -> expr / id

Pseudo Code:-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>

int expr();
int term();
int fact();
char str[30],next;
int i,l=0,cursor,l;
char get_char();

void main()
{
        clrscr();
        printf("Enter any string: ");
        gets(str);
        l=strlen(str);
        cursor=0;
        next=get_char();
```

```
	if(expr())
	{
		if(next=='#')
			printf("Valid..\n");
		else
			printf("Invalid..\n");
	}
	else
	{
		printf("Invalid..\n");
	}
	getch();
}

int expr()
{
	if(!term())
	{
		return 0;
	}
	if(next=='+' || next=='-')
	{
		next=get_char();
		if(next=='#')
		{
			return 0;
		}
		if(!expr())
			return 0;
		else
			return 1;
```

```
        }
        else
                return 1;
}

int term()
{
        if(!fact())
        {
                return 0;
        }
        if(next=='*' || next == '/')
        {
                next = get_char();
                if(next=='#')
                        return 0;
                else if(!term())
                        return 0;
                else
                        return 1;
        }
        else
                return 1;
}

int fact()
{
        if(next=='#')
        {
                return 0;
        }
```

```
	if(next=='(')
	{
		next=get_char();
		if(next=='#')
			return 0;
		if(!expr())
			return 0;
		if(next!=')')
			return 0;
		else
		{
			next=get_char();
			return 1;
		}
	}
	if(next!='i')
		return 0;
	else
	{
		next=get_char();
		return 1;
	}
}

char get_char()
{
	char chr=str[cursor];
	cursor=cursor+1;
	return chr;
}
```

OUTPUT:

Enter the String: a+a*a

Valid String

Practical - 4

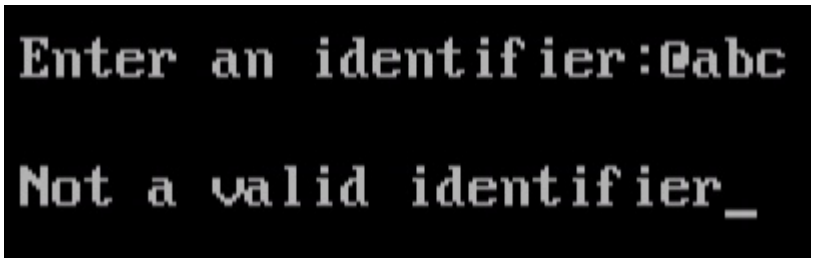
Aim: Write a C program to test whether a given identifier is valid or not

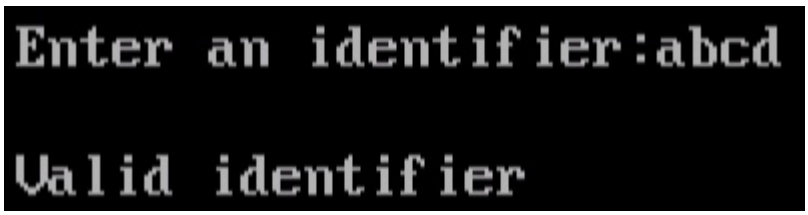
Pseudo Code:-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
char a[10];
int flag, i1=1;
clrscr();
printf("\n Enter an identifier:");
gets(a);
if(isalpha(a[0])) flag=1;
else
printf("\n Not a valid identifier");
while(a[i1]!='\0')
{
if(!isdigit(a[i1])&&!isalpha(a[i1]))
 { flag=0;
 break;
 }
 i1++;
 }
 if(flag==1)
 printf("\n Valid identifier");
 if(flag==0)
 printf("\n invalid identifier");
 getch();
```

}

OUTPUT:


Enter an identifier:@abc
Not a valid identifier_


Enter an identifier:abcd
Valid identifier

Practical - 5

Aim: To Study about Lexical Analyzer Generator (LEX) and Fast Lexical Analyzer Generator (FLEX)

Theory:-

Lexical Analyzer Generator

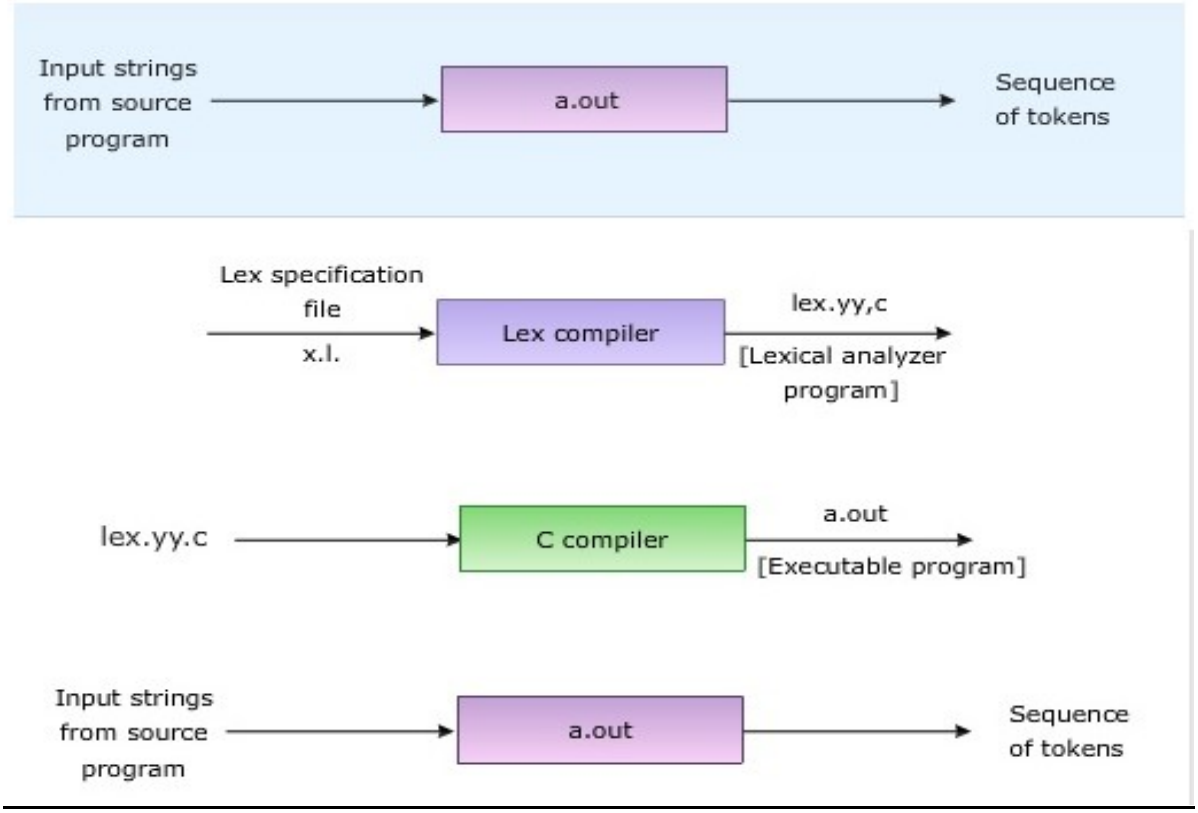
Lexical Analyzer Generator introduce a tool called Lex, which allows one to specify a lexical analyzer by specifying regular expressions to describe pattern for tokens.

The input for the lex tool is lex language.

A program which is written in lex language will compile through lex compiler and produce a C Code called *lex.yy.c* always i.e.,

Now, C code is compiled by C compiler and produce a file called a.out as always i.e.,

The C compiler output is a working lexical analyzer that can take a stream of input character and produce a stream of tokens i.e.,



FLEX

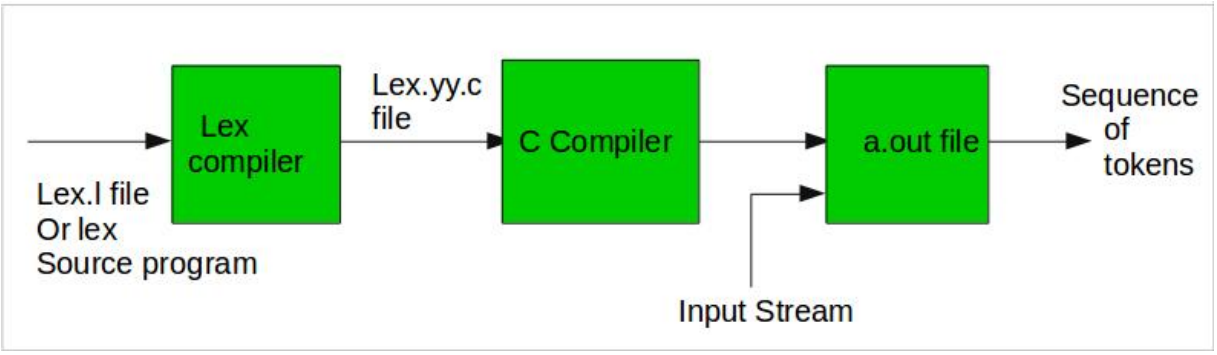
Flex (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produces faster code.
Bison produces parser from the input file provided by the user. The function **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.

Installing Flex on Ubuntu:

sudo apt-get update

sudo apt-get install flex

Given image describes how the Flex is used:



Step 1: An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.

Step 2: The C complier compile lex.yy.c file into an executable file called a.out.

Step 3: The output file a.out take a stream of input characters and produce a stream of tokens.

Program Structure:

In the input file, there are 3 sections:

1. Definition Section: The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in "%{ %}" brackets. Anything written in this brackets is copied directly to the file lex.yy.c

Syntax:

```
%{
  // Definitions
%}
```

2. Rules Section: The rules section contains a series of rules in the form: *pattern action* and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in "%% %%".

Syntax:

```
%%
pattern  action
%%
```

3. User Code Section: This section contain C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

Basic Program Structure:

```
%{
```

// Definitions
%}

%%
Rules
%%
User code section

How to run the program:

To run the program, it should be first saved with the extension .l or .lex. Run the below commands on terminal in order to run the program file.

Step 1: lex filename.l or lex filename.lex depending on the extension file is saved with

Step 2: gcc lex.yy.c

Step 3: ./a.out

Step 4: Provide the input to program in case it is required

Practical - 6

Aim: Write a C program to find first and follow for the given grammar.

Pseudo Code:-

```
#include<conio.h>
#include<stdio.h>
#include<math.h>

char g[5][2][10]={
                        {"e->tE"},              //treat E as e'
                        {{"E->+tE"},{"E->N"}},  //treat N as null
                        {"t->fT"},              //treat T as t'
                        {{"T->*fT"},{"T->N"}},
                        {{"f->(e)"},{"f->i"}},
                };
char resf[5][2];
char resfollow[5][5];
int track[5];
void first(char);
void follow(void);

void main()
{
        int i,k;
        clrscr();
        //find first
        for(k=0;k<5;k++)
        {
                for(i=0;i<5;i++)
                        track[i]=0;
```

```
            if(resf[k][0]=='\x0')
                first(g[k][0][0]);
        }
        follow();
        printf("\tFirst\tFollow\n");
        for(i=0;i<5;i++)
        {
            printf("\n\n%c\t",g[i][0][0]);
            printf("%c,%c %10s",resf[i][0],resf[i][1],resfollow[i]);
        }
        getch();
}

void first(char ref)
{
        int i,j,production,k;
        char c;
        for(i=0;i<5;i++)
        {
            if(g[i][0][0]==ref)
            {
                production=i;
                track[production]=1;
            }
        }
        for(i=0;i<2;i++)
        {
            c=g[production][i][0+3];
            if(c=='+'||c=='*'||c=='('||c==')'||c=='i'||c=='N')
            {
                for(k=0;k<5;k++)
```

```
                {
                        if(track[k]!=0)
                                resf[k][i]=c;
                }
        }
        else if(c !='\x0')

                first(c);
    }
}
void follow(void)
{
    int i,j,k,l,pbeta,pB,pA;
    int index[5]={0,0,0,0,0};
    int iA,iB,ibeta;
    int rule3;
    int length,redundant;
    char start,a,c,B,beta,A;
    //rule 1
    for(i=0;i<5;i++)
    {
        for(j=0;j<2;j++)
        {
            start=g[i][j][3];
            if(start != '+' && start != '*' && start != '(' && start != ')' && start != 'N'
&& start != 'i' && start!='\x0')
                resfollow[i][index[i]++]='$';
        }
        //defination
        length=strlen(g[i][0]);
        a=g[i][0][length-1];
```

```
        A=g[i][0][length-2];
        if((a=='+'||a=='*'||a=='('||a==')'||a=='i')&&a!='N')
        {
        //search for the index value of A
                for(k=0;k<5;k++)
                        if(g[k][0][0]==A)
                                pA=k;
                resfollow[pA][index[pA]++]=a;
        }
}
//rule 2
for(j=0;j<5;j++)
{
        A=g[j][0][0];
        length=strlen(g[j][0]);
        B=g[j][0][length-2];
        beta=g[j][0][length-1];
        for(i=0;i<5;i++)
        {
                if(g[i][0][0]==beta)
                        pbeta=i;
                if(g[i][0][0]==A)
                        pA=i;
                if(g[i][0][0]==B)
                        pB=i;
        }
        if(!(beta=='+' || beta=='*' || beta=='(' || beta==')' || beta=='i') && beta !='N')
        {
                for(i=0;i<2;i++)
                {
                        if((resf[pbeta][i])!='N')
```

```
                        {
                                //check for redundant element in follow
                                redundant=0;
                                for(k=0;k<5;k++)
                                {
                                        if(resfollow[pB][k]== resf[pbeta][i])
                                                redundant=1;
                                }
                                if(redundant!=1)
                                        (resfollow[pB][index[pB]++])=(resf[pbeta][i]);
                        }
                }
        }
}
//rule 3 A->(alpha)(B)(beta)
for(j=0;j<5;j++)
{
        A=g[j][0][0];
        length=strlen(g[j][0]);
        B=g[j][0][length-2];
        beta=g[j][0][length-1];
        if(!(beta=='+'||beta=='*'||beta=='('||beta==')'||beta=='i'||beta=='N'))
        {
                for(i=0;i<5;i++)
                {
                        if(g[i][0][0]==beta)
                                pbeta=i;
                        if(g[i][0][0]==A)
                                pA=i;
                        if(g[i][0][0]==B)
                                pB=i;
```

```
                }
                for(i=0;i<2;i++)
                {
                        if(resf[pbeta][i]=='N')
                        {
                                for(k=0;k<5;k++)
                                {
                                        //check for redundant element in follow
                                        redundant=0;
                                        for(l=0;l<5;l++)
                                        {
                                                if(resfollow[pB][l]== resfollow[pA][k])
                                                        redundant=1;
                                        }
                                        if(redundant!=1)

(resfollow[pB][index[pB]++])=(resfollow[pA][k]);
                                }

                        }

                }
        }
}
//rule 3 A->(alpha)(B)
for(j=0;j<5;j++)
{
        A=g[j][0][0];
        length=strlen(g[j][0]);
        B=g[j][0][length-1];
        if(A!=B)
```

```
            {
                for(i=0;i<5;i++)
                {
                    if(g[i][0][0]==A)
                        pA=i;
                    if(g[i][0][0]==B)
                        pB=i;
                }
                if(!(B=='+'||B=='*'||B=='('||B==')'||B=='i'||B=='N'))
                    for(i=0;i<5;i++)
                        resfollow[pB][index[pB]++]=resfollow[pA][i];
            }
        }
}
```

OUTPUT:

	First	Follow
e	(,i	$,)
E	+,N	$,)
t	(,i	$,+,)
T	*,N	$,+,)
f	(,i	$,+,*,)

Practical - 7

Aim: Write a C program for implementing the functionalities of predictive parser for the mini language.

Pseudo Code:-

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
char g[5][2][10]={
                        {"e->tE"},              //treat E as e'
                        {{"E->+tE"},{"E->N"}},  //treat N as null
                        {"t->fT"},              //treat T as t'
                        {{"T->*fT"},{"T->N"}},
                        {{"f->(e)"},{"f->i"}},
                  };
char resf[5][2];
char resfollow[5][5];
int track[5];
char table[5][6][7];
char ind_nt[]={'i','+','*','(',')','$'};
void ptable(void);
void first(char);
void follow(void);
void main()
{
        int i,k,j;
        clrscr();
        //find first
        for(k=0;k<5;k++)
```

```
        {
                for(i=0;i<5;i++)
                        track[i]=0;
                if(resf[k][0]=='\x0')
                        first(g[k][0][0]);
        }
        follow();
        ptable();
        for(i=0;i<6;i++)
                printf("%10c",ind_nt[i]);
        printf("\n\n");
        for(i=0;i<5;i++)
        {
                printf("%c",g[i][0][0]);
                for(k=0;k<6;k++)
                {
                printf(" %10s",table[i][k]);
                }
                printf("\n\n");
        }
        getch();
}
void first(char ref)
{
        int i,j,production,k;
        char c;
        for(i=0;i<5;i++)
        {
                if(g[i][0][0]==ref)
                {
                        production=i;
```

```
                track[production]=1;
            }
        }
        for(i=0;i<2;i++)
        {
            c=g[production][i][0+3];
            if(c=='+'||c=='*'||c=='('||c==')'||c=='i'||c=='N')
            {
                for(k=0;k<5;k++)
                {
                    if(track[k]!=0)
                        resf[k][i]=c;
                }
            }
            else if(c !='\x0')
                first(c);
        }
}
void follow(void)
{
        int i,j,k,l,pbeta,pB,pA;
        int index[5]={0,0,0,0,0};
        int iA,iB,ibeta;
        int rule3;
        int length,redundant;
        char start,a,c,B,beta,A;
        //rule 1
        for(i=0;i<5;i++)
        {
            for(j=0;j<2;j++)
            {
```

```
                start=g[i][j][3];
                if(start != '+' && start != '*' && start != '(' && start != ')' && start != 'N'
&& start != 'i' && start!='\x0')
                        resfollow[i][index[i]++]='$';
        }
        //defination
        length=strlen(g[i][0]);
        a=g[i][0][length-1];
        A=g[i][0][length-2];
        if((a=='+'||a=='*'||a=='('||a==')'||a=='i')&&a!='N')
        {
        //search for the index value of A
                for(k=0;k<5;k++)
                        if(g[k][0][0]==A)
                                pA=k;
                resfollow[pA][index[pA]++]=a;
        }
    }
    //rule 2
    for(j=0;j<5;j++)
    {
        A=g[j][0][0];
        length=strlen(g[j][0]);
        B=g[j][0][length-2];
        beta=g[j][0][length-1];
        for(i=0;i<5;i++)
        {
                if(g[i][0][0]==beta)
                        pbeta=i;
                if(g[i][0][0]==A)
                        pA=i;
```

```
                if(g[i][0][0]==B)
                        pB=i;
        }
        if(!(beta=='+' || beta=='*' || beta=='(' || beta==')' || beta=='i') && beta !='N')
        {
                for(i=0;i<2;i++)
                {
                        if((resf[pbeta][i])!='N')
                        {
                                //check for redundant element in follow
                                redundant=0;
                                for(k=0;k<5;k++)
                                {
                                        if(resfollow[pB][k]== resf[pbeta][i])
                                                redundant=1;
                                }
                                if(redundant!=1)
                                        (resfollow[pB][index[pB]++])=(resf[pbeta][i]);
                        }
                }
        }
}
//rule 3 A->(alpha)(B)(beta)
for(j=0;j<5;j++)
{
        A=g[j][0][0];
        length=strlen(g[j][0]);
        B=g[j][0][length-2];
        beta=g[j][0][length-1];
        if(!(beta=='+'||beta=='*'||beta=='('||beta==')'||beta=='i'||beta=='N'))
        {
```

```
                for(i=0;i<5;i++)
                {
                        if(g[i][0][0]==beta)
                                pbeta=i;
                        if(g[i][0][0]==A)
                                pA=i;
                        if(g[i][0][0]==B)
                                pB=i;
                }
                for(i=0;i<2;i++)
                {
                        if(resf[pbeta][i]=='N')
                        {
                                for(k=0;k<5;k++)
                                {
                                        //check for redundant element in follow
                                        redundant=0;
                                        for(l=0;l<5;l++)
                                        {
                                                if(resfollow[pB][l]== resfollow[pA][k])
                                                        redundant=1;
                                        }
                                        if(redundant!=1)

(resfollow[pB][index[pB]++])=(resfollow[pA][k]);
                                }
                        }
                }
        }
}
//rule 3 A->(alpha)(B)
```

```
	for(j=0;j<5;j++)
	{
		A=g[j][0][0];
		length=strlen(g[j][0]);
		B=g[j][0][length-1];
		if(A!=B)
		{
			for(i=0;i<5;i++)
			{
				if(g[i][0][0]==A)
					pA=i;
				if(g[i][0][0]==B)
					pB=i;
			}
			if(!(B=='+'||B=='*'||B=='('||B==')'||B=='i'||B=='N'))
				for(i=0;i<5;i++)
					resfollow[pB][index[pB]++]=resfollow[pA][i];
		}
	}
}
void ptable(void)
{
	int i,j,k,l,m;
	char A;
	for(i=0;i<5;i++)
	{
//		A=g[i][0][0];
		for(j=0;j<2;j++)
		{
			//step 2
			if(resf[i][j]!='N')
```

```
        {
            for(k=0;k<6;k++)
            {
                if(ind_nt[k]==resf[i][j])
                {
                    for(m=0;m<2;m++)
                    {
                        if(ind_nt[k]==g[i][m][3])
                        {
                            strcpy(table[i][k],g[i][m]);
                        }
                        else
                        {
                            strcpy(table[i][k],g[i][0]);
                        }
                    }
                }
            }
        }
    }
    //step 3
    for(j=0;j<2;j++)
    {
        if(resf[i][j]=='N')
        {
            for(k=0;k<3;k++)
            {
                for(l=0;l<6;l++)
                {
                    if(ind_nt[l]==resfollow[i][k])
                        strcpy(table[i][l],g[i][1]);
```

```
                                    }
                                }
                            }
                        }
                    }
                }
```

OUTPUT:

	i	+	*	(	)	$
e	e->tE			e->tE		
E		E->+tE			E->N	E->N
t	t->fT			t->fT		
T		T->N	T->*fT		T->N	T->N
f	f->i			f->(e)		

Practical - 8

Aim: Write a CPP program for constructing of LL (1) parsing

Pseudo Code:-

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void main()
{
   clrscr();
   int i=0,j=0,k=0,m=0,n=0,o=0,o1=0,var=0,l=0,f=0,c=0,f1=0;
   char     str[30],str1[40]="E",temp[20],temp1[20],temp2[20],tt[20],t3[20];
   strcpy(temp1,'\0');
   strcpy(temp2,'\0');
   char t[10];
   char array[6][5][10] = {
              "NT", "<id>","+","*",";",
              "E", "Te","Error","Error","Error",
              "e", "Error","+Te","Error","\0",
              "T", "Vt","Error","Error","Error",
              "t", "Error","\0","*Vt","\0",
              "V", "<id>","Error","Error","Error"
               };
   cout << "\n\tLL(1)  PARSER  TABLE \n";
   for(i=0;i<6;i++)
   {
      for(j=0;j<5;j++)
      {
```

```
            cout.setf(ios::right);
            cout.width(10);
            cout<<array[i][j];
          }
          cout<<endl;
      }
      cout << endl;
      cout << "\n\tENTER THE STRING :";
      gets(str);
      if(str[strlen(str)-1] != ';')
      {
          cout << "END OF STRING MARKER SHOULD BE ';'";
          getch();
          exit(1);
      }
      cout << "\n\tCHECKING VALIDATION OF THE STRING ";
      cout <<"\n\t" << str1;
      i=0;
  while(i<strlen(str))
      {
       again:
          if(str[i] == ' ' && i<strlen(str))
          {
              cout << "\n\tSPACES IS NOT ALLOWED IN SOURSE STRING ";
              getch();
              exit(1);
          }
          temp[k]=str[i];
          temp[k+1]='\0';
          f1=0;
       again1:
```

```
if(i>=strlen(str))
{
    getch();
    exit(1);
}
for(int l=1;l<=4;l++)
{
 if(strcmp(temp,array[0][l])==0)
 {
    f1=1;
    m=0,o=0,var=0,o1=0;
    strcpy(temp1,'\0');
    strcpy(temp2,'\0');
    int len=strlen(str1);
    while(m<strlen(str1) && m<strlen(str))
    {
        if(str1[m]==str[m])
        {
            var=m+1;
            temp2[o1]=str1[m];
            m++;
            o1++;
        }
        else
        {
            if((m+1)<strlen(str1))
            {
               m++;
               temp1[o]=str1[m];
               o++;
            }
```

```
            else
                m++;
        }

}
temp2[o1] = '\0';
temp1[o] = '\0';
t[0] = str1[var];
t[1] = '\0';
for(n=1;n<=5;n++)
{
   if(strcmp(array[n][0],t)==0)
      break;
}
strcpy(str1,temp2);
strcat(str1,array[n][l]);
strcat(str1,temp1);
cout << "\n\t" <<str1;
getch();

if(strcmp(array[n][l],'\0')==0)
{
   if(i==(strlen(str)-1))
   {
        int len=strlen(str1);
        str1[len-1]='\0';
        cout << "\n\t"<<str1;
        cout << "\n\n\tENTERED STRING IS                    VALID";
        getch();
        exit(1);
    }
```

```
      strcpy(temp1,'\0');
      strcpy(temp2,'\0');
      strcpy(t,'\0');
      goto again1;
}
if(strcmp(array[n][l],"Error")==0)
{
      cout << "\n\tERROR IN YOUR SOURCE STRING";
      getch();
      exit(1);
}
strcpy(tt,'\0');
strcpy(tt,array[n][l]);
strcpy(t3,'\0');
f=0;
for(c=0;c<strlen(tt);c++)
{
    t3[c]=tt[c];
    t3[c+1]='\0';
    if(strcmp(t3,temp)==0)
    {
        f=0;
        break;
    }
    else
        f=1;
 }

 if(f==0)
 {
   strcpy(temp,'\0');
```

```
                strcpy(temp1,'\0');
                strcpy(temp2,'\0');
                strcpy(t,'\0');
                i++;
                k=0;
                goto again;
              }
              else
              {
                strcpy(temp1,'\0');
                strcpy(temp2,'\0');
                strcpy(t,'\0');
                goto again1;
              }
            }
          }
          i++;
          k++;
    }
    if(f1==0)
         cout << "\nENTERED STRING IS INVALID";
    else
        cout << "\n\n\tENTERED STRING IS VALID";
    getch();
}
```

OUTPUT:-

LL(1) PARSER TABLE

NT	<id>	+	*	;

E	Te	Error	Error	Error
e	Error	+Te	Error	
T	Vt	Error	Error	Error
t	Error	*Vt		
V	<id>	Error	Error	Error

ENTER THE STRING :<id>+<id>*<id>;

CHECKING VALIDATION OF THE STRING

E

Te

Vte

<id>te

<id>e

<id>+Te

<id>+Vte

<id>+<id>te

<id>+<id>*Vte

<id>+<id>*<id>te

<id>+<id>*<id>e

<id>+<id>*<id>

ENTERED STRING IS VALID

Practical - 9

Aim: To Study about Yet Another Compiler-Compiler (YACC)

Theory:-

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

YACC (yet another compiler-compiler) is an LALR(1) (LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.

Input File:

YACC input file is divided in three parts.

```
/* definitions */
 ....

%%
/* rules */
....
%%

/* auxiliary routines */
....
```

Input File: Definition Part:

- The definition part includes information about the tokens used in the syntax definition:
- %token NUMBER
- %token ID

- Yacc automatically assigns numbers for tokens, but it can be overridden by

%token NUMBER 621

- Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should no overlap ASCII codes.
- The definition part can include C code external to the definition of the parser and variable declarations, within %{ and %} in the first column.
- It can also include the specification of the starting symbol in the grammar:

%start nonterminal

Input File: Rule Part:

- The rules part contains grammar definition in a modified BNF form.
- Actions is C code in { } and can be embedded inside (Translation schemes).

Input File: Auxiliary Routines Part:

- The auxiliary routines part is only C code.
- It includes function definitions for every function needed in rules part.
- It can also contain the main() function definition if the parser is going to be run as a program.
- The main() function must call the function yyparse().

Input File:

- If yylex() is not defined in the auxiliary routines sections, then it should be included:

#include "lex.yy.c"

- YACC input file generally finishes with:

.y

Output Files:

- The output of YACC is a file named y.tab.c
- If it contains the main() definition, it must be compiled to be executable.
- Otherwise, the code can be an external function definition for the function int yyparse()
- If called with the –d option in the command line, Yacc produces as output a header file y.tab.h with all its specific definition (particularly important are token definitions to be included, for example, in a Lex input file).

- If called with the –v option, Yacc produces as output a file y.output containing a textual description of the LALR(1) parsing table used by the parser. This is useful for tracking down how the parser solves conflicts.