Ex.No : 1	CONSTRUCTION OF NFA

AIM:

To write a program for constructing an NFA from given regular expression using C++.

ALGORITHM:

1. Create a menu for getting four regular expressions input as choice.
2. To draw NFA for a, a/b ,ab ,a* create a routine for each regular expression.
3. For converting from regular expression to NFA, certain transition had been made based on choice of input at the rumtime.
4. Each of the NFA will be displayed is sequential order.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
struct node
{
        char start;
        char alp;
        node *nstate;
}*p,*p1,*p2,*p3,*p4,*p5,*p6,*p7,*p8;
char e='e';
void disp();
void re1()
{
        p1=new(node);
        p2=new(node);
        p1->start='0';
        p1->alp='e';
        p1->nstate=p2;
        p2->start='1';
        p2->alp='a';
        p2->nstate=NULL;
        disp();
        getch();
}
void re2()
{
        p1=new(node);
        p2=new(node);
        p3=new(node);
        p4=new(node);
        p5=new(node);
        p6=new(node);
        p7=new(node);
        p8=new(node);
```

```
        p1->start='0';
        p1->alp='e';
        p1->nstate=p2;
        p2->start='1';
        p2->alp='a';
        p2->nstate=p3;
        p3->start='2';
        p3->alp='e';
        p3->nstate=p4;
        p4->start='5';
        p4->alp=' ';
        p4->nstate=p5;
        p5->start='0';
        p5->alp='e';
        p5->nstate=p6;
        p6->start='3';
        p6->alp='b';
        p6->nstate=p7;
        p7->start='4';
        p7->alp='e';
        p7->nstate=p8;
        p8->start='5';
        p8->alp=' ';
        p8->nstate=NULL;
        disp();
        getch();
}
void re3()
{
        p1=new(node);
        p2=new(node);
        p3=new(node);
        p1->start='0';
        p1->alp='a';
        p1->nstate=p2;
```

```
        p2->start='1';
        p2->alp='b';
        p2->nstate=p3;
        p3->start='2';
        p3->alp=' ';
        p3->nstate=NULL;
        disp();
        getch();
}
void re4()
{
        p1=new(node);
        p2=new(node);
        p3=new(node);
        p4=new(node);
        p5=new(node);
        p6=new(node);
        p7=new(node);
        p8=new(node);
        p1->start='0';
        p1->alp='e';
        p1->nstate=p2;
        p2->start='1';
        p2->alp='a';
        p2->nstate=p3;
        p3->start='2';
        p3->alp='e';
        p3->nstate=p4;
        p4->start='3';
        p4->alp=' ';
        p4->nstate=p5;
        p5->start='0';
        p5->alp='e';
        p5->nstate=p6;
        p6->start='3';
```

```
        p6->alp=' ';
        p6->nstate=p7;
        p7->start='2';
        p7->alp='e';
        p7->nstate=p8;
        p8->start='1';
        p8->alp=' ';
        p8->nstate=NULL;
        disp();
        getch();
}
void disp()
{
        p=p1;
        while(p!=NULL)
        {
                cout<<"\t"<<p->start<<"\t"<<p->alp;
                p=p->nstate;
        }
}
void main()
{
        p=new(node);
        clrscr();
        int ch=1;
        while(ch!=0)
        {
                cout<<"\nMenu"<<"\n1.a"<<"\n2.a/b"<<"\n3.ab"<<"\n4.a
        *";
                cout<<"\n Enter the choice:";
                cin>>ch;
                switch(ch)
                {
                        case 1:
                        {
```

```
                re1();
                break;
            }
            case 2:
            {
                re2();
                break;
            }
            case 3:
            {
                re3();
                break;
            }
            case 4:
            {
                re4();
                break;
            }
            default:
            {
                exit(0);
            }
        }
    }
}
```

OUTPUT:

Menu

1. a
2. a/b
3. ab
4. a*

Enter the choice: 1

0 a 1

Menu

1. a
2. a/b
3. ab
4. a*

Enter the choice: 2

0 e 1 a 2 e 5 0 e 3 b 4 e 5

Menu

1. a
2. a/b
3. ab
4. a*

Enter the choice: 3

0 a 1 b 2

Menu

1. a
2. a/b
3. ab
4. a*

Enter the choice: 4

0 e 1 a 2 e 3 0 e 3 2 e 1

Menu

1. a
2. a/b
3. ab
4. a*

Enter the choice: 5

RESULT:

Thus the program for Construction of NFA from the given regular expression is executed and verified.

Ex.No : 2	CONSTRUCTION OF MINIMIZED DFA FROM A GIVEN REGULAR EXPRESSION

AIM:

To write a program for construction of minimized DFA from a given regular expression using C.

ALGORITHM:

1. Get the start state, final state, input symbols as input and also give the edge value for each state.
2. Maintain a stack required for transition from one state to other state.
3. Using Pop or push function perform the insertion and deletion of elements when required.
4. Finally conversion has been made to change from regular expression to minimized DFA and the output is displayed as DFA transition table.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define STATES 50
struct Dstate
{
        char name;
        char StateString[STATES+1];
        char trans[10];
        int is_final;
}Dstates[50];
struct tran
{
        char sym;
        int tostates[50];
        int notran;
};
struct state
{
        int no;
        struct tran tranlist[50];
};
int stackA[100],stackB[100],c[100],Cptr=-1,Aptr=-1,Bptr=-1;
struct state States[10];
char temp[STATES+1],inp[10];
int nos,noi,nof,j,k,nods=-1;

void pushA(int z)
{
        stackA[++Aptr]=z;
}
void pushB(int z)
{
        stackB[++Bptr]=z;
```

```
}

int popA()
{
        return stackA[Aptr--];
}
void copy(int i)
{
        char temp[STATES+1]=" ";
        int k=0;
        Bptr=-1;
        strcpy(temp,Dstates[i].StateString);
        while(temp[k]!='\0')
        {
        pushB(temp[k]-'0');
        k++;
        }
}
int popB()
{
        return stackB[Bptr--];
}
int peekA()
{
        return stackA[Aptr];
}
int peekB()
{
        return stackA[Bptr];
}
int seek(int arr[],int ptr,int s)
{
        int i;
        for(i=0;i<=ptr;i++)
        {
```

```
                if(s==arr[i])
                return 1;
        }
return 0;
}
void sort()
{
        int i,j,temp;
        for(i=0;i<Bptr;i++)
        {
                for(j=0;j<(Bptr-i);j++)
                {
                        if(stackB[j]>stackB[j+1])
                        {
                                temp=stackB[j];
                                stackB[j]=stackB[j+1];
                                stackB[j+1]=temp;
                        }
                }
         }
   }
void tostring()
{
        int i=0;
        sort();
        for(i=0;i<=Bptr;i++)
{
                temp[i]=stackB[i]+'0';
        }
        temp[i]='\0';
}
void display_DTran()
{
        int i,j;
```

```
        printf("\n\t\t DFA transition table");
        printf("\n\t\t ---------------------------------------- ");
        printf("\n States \tString \tInputs\n");
        for(i=0;i<noi;i++)
        {
                printf("\t %c",inp[i]);
        }
        printf("\n\t ---------------------------------------- ");
        for(i=0;i<nods;i++)
        {
        if(Dstates[i].is_final==0)
        printf("\n%c",Dstates[i].name);
        else
        printf("\n*%c",Dstates[i].name);
        printf("\t%s",Dstates[i].StateString);
        for(j=0;j<noi;j++)
        {
                printf("\t%c",Dstates[i].trans[j]);
        }
    }
     printf("\n");
}
void move(int st,int j)
{
        int ctr=0;
        while(ctr<States[st].tranlist[j].notran)
        {
                pushA(States[st].tranlist[j].tostates[ctr++]);
        }
}
void lambda_closure(int st)
{
        int ctr=0,in_state=st,curst=st,chk;
        while(Aptr!=-1)
        {
```

```
                curst=popA();
                ctr=0;
                in_state=curst;
        while(ctr<=States[curst].tranlist[noi].notran)
                {
                        chk=seek(stackB,Bptr,in_state);
                        if(chk==0)
                        pushB(in_state);
                        in_state=States[curst].tranlist[noi].tostates[ctr++];
                        chk=seek(stackA,Aptr,in_state);
                        if(chk==0 && ctr<=States[curst].tranlist[noi].notran)
                        pushA(in_state);
                }
            }
        }
Void main()
{
        int i,final[20],start,fin=0;
        char c,ans,st[20];
        printf("\n Enter no of states in NFA:");
        scanf("%d",&nos);
        for(i=0;i<nos;i++)
        {
                States[i].no=i;
        }
        printf("\n Enter the start states:");
        scanf("%d",&start);
        printf("Enter the no of final states:");
        scanf("%d",&nof);
        printf("Enter the final states:\n");
        for(i=0;i<nof;i++)
        scanf("%d",&final[i]);
        printf("\n Enter the no of input symbols:");
        scanf("%d",&noi);
        c=getchar();
```

```
        printf("Enter the input symbols:\n");
        for(i=0;i<noi;i++)
        {
                scanf("%c",&inp[i]);
                c=getchar();
        }
        g1inp[i]='e';
        printf("\n Enter the transitions:(-1 to stop)\n");
        for(i=0;i<nos;i++)
        {
        for(j=0;j<=noi;j++)
        {
        States[i].tranlist[j].sym=inp[j];
        k=0;
        ans='y';
        while(ans=='y')
        {
                printf("move(%d,%c);",i,inp[j]);
                scanf("%d",&States[i].tranlist[j].tostates[k++]);
                if((States[i].tranlist[j].tostates[k-1]==-1))
                {
                        k--;
                        ans='n';
                        break;
                   }
            }
                States[i].tranlist[j].notran=k;
            }
}
i=0;nods=0,fin=0;
pushA(start);
lambda_closure(peekA());
tostring();
Dstates[nods].name='A';
nods++;
```

```
strcpy(Dstates[0].StateString,temp);
while(i<nods)
{
        for(j=0;j<noi;j++)
        {
        fin=0;
        copy(i);
        while(Bptr!=-1)
        {
                move(popB(),j);
        }

while(Aptr!=-1)
lambda_closure(peekA());
tostring();
for(k=0;k<nods;k++)
{
        if((strcmp(temp,Dstates[k].StateString)==0))
        {
                Dstates[i].trans[j]=Dstates[k].name;
                break;
        }
}
if(k==nods)
{
        nods++;
        for(k=0;k<nof;k++)
        {
        fin=seek(stackB,Bptr,final[k]);
        if(fin==1)
        {
                Dstates[nods-1].is_final=1;
                break;
        }
   }
```

```
            strcpy(Dstates[nods-1].StateString,temp);
            Dstates[nods-1].name='A'+nods-1;
            Dstates[i].trans[j]=Dstates[nods-1].name;
            }
            }
            i++;
    }
    display_DTran();
    }
```

OUTPUT:

Enter the no of input symbols:2

Enter the input symbols:

a ,b

Enter the transitions:(-1 to stop)

move(0,a);-1

move(0,b);-1

move(0,e);1

move(0,e);7

move(0,e);-1

move(1,a);-1

move(1,b);-1

move(1,e);2

move(1,e);4

move(1,e);-1

move(2,a);3

move(2,a);3

move(2,a);-1

move(2,b);-1

move(2,e);-1

move(3,a);-1

move(3,b);-1

move(3,e);6

move(3,e);-1

move(4,a);-1

move(4,b);-1

move(4,e);-1

move(5,a);-1

move(5,b);-1

move(5,e);6

move(5,e);1

move(5,e);-1

move(6,a);-1

move(6,b);-1

move(6,e);-1

move(7,a);-1

move(7,b);-1

move(7,e);-1

DFA transition table

States	String	Inputs	
		a	b
A	01247	B	C
B	36	C	C
C		C	C

RESULT:

Thus the program for Construction of minimized DFA from the given regular expression is executed and verified.

Ex.No:3	IMPLEMENTATION OF A LEXICAL ANALYZER USING LEX

AIM:

To write a program for implementing a Lexical analyser using LEX tool in Linux platform.

ALGORITHM:

1. Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%.
   The format is as follows:
   definitions
   %%
   rules
   %%
   user_subroutines
2. In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.
3. In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.
4. Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.
5. When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

6. In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.
7. The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

3.(a) The program replaces the substring abc by ABC from the given input string:

```
%{
#include<stdio.h>
#include<string.h>
int i;
%}
%%
[a-z A-Z]* {
for(i=0;i<=yyleng;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",yytext);
}
[\t]* return;
.* {ECHO;}
\n {printf("%s",yytext);}
%%
main()
```

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

OUTPUT:

```
[CSE@localhost ~]$ lex lex1.l
[CSE@localhost ~]$ cc lex.yy.c
[CSE@localhost ~]$. /a.out

abc
ABC
```

3.(b) Well formedness of brackets

```
%{
/*input is b.c*/
#include<stdio.h>
#include<string.h>
char temp[10];
int i=0,openbracket=0,closebracket=0;
extern FILE *yyin;
%}

%%
"("[()]*")"";"{
strcpy(temp,yytext);
printf("%s\n",temp);
i=0;
```

```
while(temp[i]!=';')
{
if(temp[i]=='(')
openbracket++;
if(temp[i]==')')
closebracket++;
else ;
i++;
}
if(openbracket=closebracket)
printf("Well formed input!\n");
else
printf("not well formed!\n");
}
%%
main(int argc,char *argv[])
{
FILE *fp=fopen(argv[1],"r");
yyin=fp;
yylex();
fclose(fp);
}
```

OUTPUT:

```
[CSE@localhost ~]$ lex lex2.l
[CSE@localhost ~]$ gcc lex.yy.c -ll
[CSE@localhost ~]$. /a.out b.c
(());
Well formed input
[CSE@localhost ~]$ lex lex2.l
[CSE@localhost ~]$ gcc lex.yy.c -ll
[CSE@localhost ~]$. /a.out b.c
());
Not well formed
```

3.(c) Finding vowels and consonant in a string:

```
%{
int vowel_cnt=0,consonant_cnt=0;
%}
vowel [aeiou]+
consonant [^aeiou]
eol \n
%%
{eol} return 0;
[\t]+ ;
{vowel} {vowel_cnt++;}
{consonant} {consonant_cnt++;}

%%
int main()
{
printf("\n Enter some input string:\n");
yylex();
printf("Vowels=%d and consonant=%d\n",vowel_cnt,consonant_cnt);
return 0;
}
int yywrap()
{
return 1;
}
```

OUTPUT:

```
[CSE@localhost ~]$ lex lex2.l
[CSE@localhost ~]$ cc lex.yy.c
[CSE@localhost ~]$. /a.out
Enter some input string: parth
Vowels=1 and consonant=4
```

3.(d) Finding the capital:

```
%{
%}
%%
[A-Z]+[ \t\n\.\,] {printf("%s",yytext);}
.          ;
%%
main()
{
printf("\n Enter some input with capital words in between:\n");
yylex();
}
int yywrap()
{
return 1;
}
```

OUTPUT:

```
[CSE@localhost ~]$ lex lex2.l
[CSE@localhost ~]$ cc lex.yy.c
[CSE@localhost ~]$. /a.out
Enter some input with capital words in between:
I am PROUD of you – INDIA
        I PROUD INDIA
```

3.(e) It is used to display the Keywords and identifiers in the given program:

```
%{
        int COMMENT=0;
%}
        identifier[a-z|A-Z][a-z|A-Z|0-9]*
%%
        #.* {printf("\n%s is a preprocesor directive",yytext);}
        int {printf("\n\t%s is a keyword",yytext);}
        float {printf("\n\t%s is a keyword",yytext);}
        double {printf("\n\t%s is a keyword",yytext);}
        char {printf("\n\t%s is a keyword",yytext);}
        if {printf("\n\t%s is a keyword",yytext);}
        else {printf("\n\t%s is a keyword",yytext);}
        while {printf("\n\t%s is a keyword",yytext);}
        do {printf("\n\t%s is a keyword",yytext);}
        return {printf("\n\t%s is a keyword",yytext);}
        break {printf("\n\t%s is a keyword",yytext);}
        continue {printf("\n\t%s is a keyword",yytext);}
        void {printf("\n\t%s is a keyword",yytext);}
        switch {printf("\n\t%s is keyword",yytext);}
        for {printf("\n\t%s is a keyword",yytext);}
        typedef {printf("\n\t%s is a keyword",yytext);}
        struct {printf("\n\t%s is a keyword",yytext);}
        goto {printf("\n\t%s is a keyword",yytext);}
        "/*" {COMMENT=1;}
        "*/" {COMMENT=0;}
        {identifier}\( {if(!COMMENT)
        printf("\nFUNCTIONS\n\t%s",yytext);}
        \{ {if(!COMMENT) printf("\nBLOCK BEGINS");}
        \} {if(!COMMENT) printf("\nBLOCK ENDS");}
        {identifier} {if(!COMMENT) printf("\n%sIDENTIFIER",yytext);}
        {identifier}(\[[0-9]*\])?\( {if(!COMMENT)
```

```
        printf("\n%sIDENTIFIER",yytext);}
        \".*\" {if(COMMENT)printf("\n\t%s is a string",yytext);}
        [0-9]+ {if(COMMENT)printf("\n\t%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);}
        \\n
%%
int main(int argc,char **argv)
{
        if(argc>1)
        {
                FILE *file;
                file=fopen(argv[1],"r");
                if(!file)
                {
                        printf("COULD NOT OPEN %s\n",argv[1]);
                        exit(1);
                }
                yyin=file;
         }
        yylex();
        printf("\n\n");
        return 0;
 }
        int yywrap()
        {
                return 0;
        }
```

OUTPUT:

```
[CSE@localhost ~]$ lex lex5.l
[CSE@localhost ~]$ gcc lex.yy.c -II
[CSE@localhost ~]$. /a.out
FUNCTIONS
main()
{
BLOCK BEGINS
int a,b,c;
int is a keyword
aIDENTIFIER,
bIDENTIFIER,
cIDENTIFIER;
c=sum(a+b);
```

RESULT:

Thus the program for implementing Lexical analyser using LEX tool is executed and verified.

Ex.No:4	IMPLEMENTATION OF SYMBOL TABLE

AIM:

To write a program for implementing Symbol Table using C.

ALGORITHM:

1. Start the program for performing insert, display, delete, search and modify option in symbol table
2. Define the structure of the Symbol Table
3. Enter the choice for performing the operations in the symbol Table
4. If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays "Duplicate Symbol". Else, insert the symbol and the corresponding address in the symbol table.
5. If the entered choice is 2, the symbols present in the symbol table are displayed.
6. If the entered choice is 3, the symbol to be deleted is searched in the symbol table.
7. If it is not found in the symbol table it displays "Label Not found". Else, the symbol is deleted.
8. If the entered choice is 5, the symbol to be modified is searched in the symbol table. The label or address or both can be modified.

PROGRAM:

```
# include <stdio.h>
# include <conio.h>
# include <alloc.h>
# include <string.h>
# define null 0
int size=0;
void insert();
void del();
int search(char lab[]);
void modify();
void display();
struct symbtab
{
        char label[10];
        int addr;
        struct symtab *next;
};
struct symbtab *first,*last;
void main()
{
        int op;
        int y;
        char la[10];
        clrscr();
        do
        {
                printf("\nSYMBOL TABLE IMPLEMENTATION\n");
                printf("1. INSERT\n");
                printf("2. DISPLAY\n");
                printf("3. DELETE\n");
                printf("4. SEARCH\n");
                printf("5. MODIFY\n");
                printf("6. END\n");
```

```
printf("Enter your option : ");
scanf("%d",&op);
switch(op)
{
        case 1:
        insert();
        display();
        break;
        case 2:
        display();
        break;
        case 3:
        del();
        display();
        break;
        case 4:
        printf("Enter the label to be searched : ");
        scanf("%s",la);
        y=search(la);
        if(y==1)
        {
        printf("The label is already in the symbol Table");
        }
        else
        {
        printf("The label is not found in the symbol table");
        }
        break;
        case 5:
        modify();
        display();
        break;
        case 6:
        break;
}
```

```
        }
        while(op<6);
        getch();
}
void insert()
{
        int n;
        char l[10];
        printf("Enter the label : ");
        scanf("%s",l);
        n=search(l);
        if(n==1)
        {
        printf("The label already exists. Duplicate cant be inserted\n");
        }
        else
        {
                struct symbtab *p;
                p=malloc(sizeof(struct symbtab));
                strcpy(p->label,l);
                printf("Enter the address : ");
                scanf("%d",&p->addr);
                p->next=null;
                if(size==0)
                {
                        first=p;
                        last=p;
                }
                else
                {
                        last->next=p;
                        last=p;
                }
                size++;
        }
```

```
}
void display()
{
        int i;
        struct symbtab *p;
        p=first;
        printf("LABEL\tADDRESS\n");
        for(i=0;i<size;i++)
        {
                printf("%s\t%d\n",p->label,p->addr);
                p=p->next;
        }

}
int search(char lab[])
{
        int i,flag=0;
        struct symbtab *p;
        p=first;
        for(i=0;i<size;i++)
        {
                if(strcmp(p->label,lab)==0)
                {
                        flag=1;
                }
                p=p->next;
        }
        return flag;
}
void modify()
{
        char l[10],nl[10];
        int add, choice, i, s;
        struct symbtab *p;
        p=first;
```

```
printf("What do you want to modify?\n");
printf("1. Only the label\n");
printf("2. Only the address of a particular label\n");
printf("3. Both the label and address\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
        case 1:
        printf("Enter the old label\n");
        scanf("%s",l);
        printf("Enter the new label\n");
        scanf("%s",nl);
        s=search(l);
        if(s==0)
        {
                printf("NO such label");
        }
        else
        {
                for(i=0;i<size;i++)
                {
                        if(strcmp(p->label,l)==0)
                        {
                                strcpy(p->label,nl);
                        }
                        p=p->next;
                }
        }
        break;
        case 2:
        printf("Enter the label whose address is to modified\n");
        scanf("%s",l);
        printf("Enter the new address\n");
        scanf("%d",&add);
```

```
s=search(l);
if(s==0)
{
	printf("NO such label");
}
else
{
	for(i=0;i<size;i++)
	{
		if(strcmp(p->label,l)==0)
		{
			p->addr=add;
		}
		p=p->next;
	}
}
break;
case 3:
printf("Enter the old label : ");
scanf("%s",l);
printf("Enter the new label : ");
scanf("%s",nl);
printf("Enter the new address : ");
scanf("%d",&add);
s=search(l);
if(s==0)
{
	printf("NO such label");
}
else
{
	for(i=0;i<size;i++)
	{
		if(strcmp(p->label,l)==0)
		{
```

```
                        strcpy(p->label,nl);
                        p->addr=add;
                    }
                    p=p->next;
                }
            }
            break;
        }
}
void del()
{
    int a;
    char l[10];
    struct symbtab *p,*q;
    p=first;
    printf("Enter the label to be deleted\n");
    scanf("%s",l);
    a=search(l);
    if(a==0)
    {
        printf("Label not found\n");
    }
    else
    {
        if(strcmp(first->label,l)==0)
        {
            first=first->next;
        }
        else if(strcmp(last->label,l)==0)
        {
            q=p->next;
            while(strcmp(q->label,l)!=0)
            {
                p=p->next;
                q=q->next;
```

```
                }
                p->next=null;
                last=p;
        }
        else
        {
                q=p->next;
                while(strcmp(q->label,l)!=0)
                {
                        p=p->next;
                        q=q->next;
                }
                p->next=q->next;
        }
        size--;
    }
}
```

OUTPUT:

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:1

Enter the label: A

Enter the address: 10

LABEL ADDRESS

A 10

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:1

Enter the label: B

Enter the address: 11

LABEL ADDRESS

A 10

B 11

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:2

LABEL ADDRESS

A 10

B 11

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:3

Enter the label to be deleted: A

LABEL ADDRESS

B 11

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:4

Enter the label to be searched: A

The label is not found in the symbol table.

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:4

Enter the label to be searched: B

The label is already in the symbol table.

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:5

what do you want to modify?

1.only the label

2.only the address of a particular label

3.both the label and address

Enter your choice:1

Enter the old label

B

Enter the new label

A

LABEL	ADDRESS
a	11

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option:6

RESULT:

Thus the program for implementation of Symbol Table is executed and verified.

Ex.No:5	CONSTRUCTION OF OPERATOR PRECEDENCE PARSE TABLE

AIM:

To write a program for construction of Operator Precedence Parse Table using C++.

ALGORITHM:

1. Get the input as expression.
2. Check the preference of each symbol in the given expression with all other symbols in the input expression.
3. After checking display the Operator Precedence matrix with corresponding symbol comparison and specify it using the relational operators< (Less than), > (Greater than) and =(equals).
4. Also display the postfix expression format of the given expression as tree structure.

PROGRAM:

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#define node struct tree1
int getOperatorPosition(char);
int matrix[5][5]=
                    {     {1,0,0,1,1},
                          {1,1,0,1,1},
                          {0,0,0,2,3},
                          {1,1,3,1,1},
                          {0,0,0,3,2}       };
int tos=-1;
void matrix_value(void);
void show_tree(node *);
int isOperator(char);
struct tree1
{
      char data;
      node *lptr;
      node *rptr;
}*first;
struct opr
{
      char op_name;
      node *t;
}
oprate[50];
```

```
char cur_op[5]={'+','*','(',')','['};
char stack_op[5]={'+','*','(',')',']'};
void main()
{
        char exp[10];
        int ssm=0,row=0,col=0;
        node *temp;
        clrscr();
        printf("enter exp:");
        scanf("%s",exp);
        matrix_value();
        while(exp[ssm]!='\0')
        {
                if(ssm==0)
                {
                        tos++;
                        oprate[tos].op_name=exp[tos];
                }
                else
                {
                        if(isOperator(exp[ssm])==-1)
                        {
                                oprate[tos].t=(node*)malloc(sizeof(node));
                                oprate[tos].t->data=exp[ssm];
                                oprate[tos].t->lptr='\0';
                                oprate[tos].t->rptr='\0';
                        }
```

```
else
{
        row=getOperatorPosition(oprate[tos].op_name);
        col=getOperatorPosition(exp[ssm]);
        if(matrix[row][col]==0)
        {
                tos++;
                oprate[tos].op_name=exp[ssm];
        }
        else if(matrix[row][col]==1)
        {
                temp=(node*)malloc(sizeof(node));
                temp->data=oprate[tos].op_name;
                temp->lptr=(oprate[tos-1].t);
                temp->rptr=(oprate[tos].t);
                tos--;
                oprate[tos].t=temp;
                ssm--;
        }
        else if(matrix[row][col]==2)
        {
                temp=oprate[tos].t;
                tos--;
                oprate[tos].t=temp;
        }
        else if(matrix[row][col]==3)
        {
```

```
                                printf("\b expression is invalid...\n");
                                printf("%c %c can not occur
                                        simuktaneously\n",
                                                oprate[tos].op_name,exp[ss
                                        m]);
                                break;
                        }
                }
        }ssm++;
    }
    printf("show tree \n\n\n");
    show_tree(oprate[tos].t);
    printf("over");
    getch();
}
int isOperator(char c)
{
    int i=0;
    for(i=0;i<5;i++)
    {
        if(c==cur_op[i]||c==stack_op[i])
        break;
    }
    if(i==5)
        return (-1);
    else
        return 1;
}
```

```
int getOperatorPosition(char c)
{
        int i;
        for(i=0;i<=5;i++)
        {
                if(c==cur_op[i]||c==stack_op[i])
                        break;
        }
        return i;
}
void show_tree(node *start)
{
        if(start->lptr !=NULL)
                show_tree(start->lptr);
                if(start->rptr !=NULL)
                        show_tree(start->rptr);
                        printf("%c \n",start->data);
}
void matrix_value(void)
{
        int i,j;
        printf("operator precedence matrix\n");
        printf("============================\n");
        for(i=0;i<5;i++)
        {
                printf("%c",stack_op[i]);
        }
```

```
        printf("\n");
        for(i=0;i<5;i++)
        {
                printf("%c",cur_op[i]);
                for(j=0;j<5;j++)
                {
                        if(matrix[i][j]==0)
                                printf("<");
                        else if(matrix[i][j]==1)
                                printf(">");
                        else if(matrix[i][j]==2)
                                printf("=");
                        else if(matrix[i][j]==3)
                                printf(" ");
                }
                printf("\n");
        }
}
```

OUTPUT:

Enter the Expression: [a+b*c]

Operator Precedence Matrix

+	*	<	>	]
+	>	<	<	>
*	>	>	<	>
<	<	<	<	=
>	>	>	>	>
[	<	<	<	>

Show Tree

a

b

c

*

+

Over

RESULT:

Thus the program for construction of Operator precedence parse table is executed and verified.

Ex.No:6	IMPLEMENTATION OF CALCULATOR USING YACC

AIM:

To write a program for implementing a calculator for computing the given expression using semantic rules of the YACC tool.

ALGORITHM:

1. A Yacc source program has three parts as follows:

   Declarations

   %%

   translation rules

   %%

   supporting C routines

2. Declarations Section:

   This section contains entries that:

   i. Include standard I/O header file.

   ii. Define global variables.

   iii. Define the list rule as the place to start processing.

   iv. Define the tokens used by the parser.

   v. Define the operators and their precedence.

3. Rules Section:

   The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

4. Programs Section:

   The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

5. Main- The required main program that calls the yyparse subroutine to start the program.

6. yyerror(s) -This error-handling subroutine only prints a syntax error message.

7. yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.
8. calc.lex contains the rules to generate these tokens from the input stream.

PROGRAM:

Cal.l

```
%{
        #include<stdio.h>
        #include<math.h>
        #include"y.tab.h"
%}
%%
([0-9]+|([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?)
{yylval.dval=atof(yytext);
        return NUMBER;}
        MEM {return MEM;}
        [\t];
        \$ {return 0;}
        \n {return yytext[0];}
        . {return yytext[0];}
%%
```

Cal.y

```
%{
        #include<stdio.h>
        #include<math.h>
        double memvar;
%}
        %union
        {
                double dval;
```

```
        }
        %token<dval> NUMBER
        %token<dval> MEM
        %left '-' '+'
        %left '*' '/'
        %nonassoc UMINUS
        %type<dval> expression
%%
        start:statement '\n'
        |start statement '\n'
        statement:MEM '=' expression {memvar=$3;}
        |expression {printf("answer=%g\n",$1);}
        ;
        expression:expression'+'expression {$$=$1+$3;}
        |expression'-'expression {$$=$1-$3;}
        |expression'*'expression {$$=$1*$3;}
        |expression'/'expression {if($3==0)
        yyerror("divide by zero");
        else
        $$=$1/$3;
        };
        expression:'-'expression %prec UMINUS {$$= -$2;}
        |'('expression')' {$$=$2;}
        |NUMBER {$$=$1;}
        |MEM {$$=memvar;};
%%
int main(void)
{
        printf("Enter the expression");
        yyparse();
        printf("\n\n");
        return 0;
}
int yywrap()
{
```

```
        return 0;
}
int yyerror(char *error)
{
        printf("%s\n",error);
        return 0;
}
```

OUTPUT:

```
[CSE@localhost ~]$ lex cal.l
[CSE@localhost ~]$ yacc -d cal.y
[CSE@localhost ~]$ cc lex.yy.c y.tab.c -ll
[CSE@localhost ~]$ ./a.out
Enter the expression5+3
answer=8

[cse@NFSSERVER ~]$ ./a.out
Enter the expression5+-5
answer=0

[cse@NFSSERVER ~]$ ./a.out
Enter the expression+5/
syntax error
```

RESULT:

Thus the program for implementation of Calculator using YACC tool is executed and verified.

Ex.No:7	IMPLEMENTATION OF SHIFT-REDUCED PARSING ALGORITHMS

AIM:

To write a program for implementing Shift Reduce Parsing using C.

ALGORITHM:

1. Get the input expression and store it in the input buffer.
2. Read the data from the input buffer one at the time.
3. Using stack and push & pop operation shift and reduce symbols with respect to production rules available.
4. Continue the process till symbol shift and production rule reduce reaches the start symbol.
5. Display the Stack Implementation table with corresponding Stack actions with input symbols.

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<string.h>

char ip_sym[15],stack[15];

int ip_ptr=0,st_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main()

{

        clrscr();

        printf("\n\t\t SHIFT REDUCE PARSER\n");

        printf("\n GRAMMER\n");

        printf("\n E->E+E\n E->E/E");

        printf("\n E->E*E\n E->a/b");

        printf("\n enter the input symbol:\t");

        gets(ip_sym);

        printf("\n\t stack implementation table");

        printf("\n stack \t\t input symbol\t\t action");

        printf("\n________\t\t_____________\t\t_____________\n");

        printf("\n $\t\t%s$\t\t\t--",ip_sym);

        strcpy(act,"shift");
```

```
        temp[0]=ip_sym[ip_ptr];
        temp[1]='\0';
        strcat(act,temp);
        len=strlen(ip_sym);
        for(i=0;i<=len-1;i++)
        {
                stack[st_ptr]=ip_sym[ip_ptr];
                stack[st_ptr+1]='\0';
                ip_sym[ip_ptr]=' ';
                ip_ptr++;
                printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
                strcpy(act,"shift");
                temp[0]=ip_sym[ip_ptr];
                temp[1]='\0';
                strcat(act,temp);
                check();
                st_ptr++;
        }
        st_ptr++;
        check();
}
void check()
{
```

```
        int flag=0;
        temp2[0]=stack[st_ptr];
        temp2[1]='\0';
        if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
        {
                stack[st_ptr]='E';
                if(!strcmpi(temp2,"a"))
                        printf("\n $%s\t\t%s$\t\t\tE->a",stack,ip_sym);
                else
                        printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip_sym);
                        flag=1;
        }
    if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))
        {
                flag=1;
        }
if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E*E")))
    {
        strcpy(stack,"E");
        st_ptr=0;
        if(!strcmpi(stack,"E+E"))
                printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
        else
```

```
                if(!strcmpi(stack,"E\E"))
                        printf("\n $%s\t\t%s$\t\t\tE->E\E",stack,ip_sym);
                else
                        if(!strcmpi(stack,"E*E"))
                        printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
                        else
                        printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
                                flag=1;
        }
        if(!strcmpi(stack,"E")&&ip_ptr==len)
        {
                printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);
                getch();
                exit(0);
        }
        if(flag==0)
        {
                printf("\n%s\t\t\t%s\t\t reject",stack,ip_sym);
                exit(0);
        }
        return;
}
```

OUTPUT:

SHIFT REDUCE PARSER

GRAMMER

E->E+E

E->E/E

E->E*E

E->a/b

Enter the input symbol: a+b

Stack Implementation Table

Stack	Input Symbol	Action
$	a+b$	--
$a	+b$	shift a
$E	+b$	E->a
$E+	b$	shift +
$E+b	$	shift b
$E+E	$	E->b
$E	$	E->E+E
$E	$	ACCEPT

RESULT:

Thus the program for implementation of Shift Reduce parsing algorithm is executed and verified.

Ex.No:8	CONSTRUCTION OF LR-PARSING TABLE

AIM:

To write a program for construction of LR Parsing table using C.

ALGORITHM:

1. Get the input expression and store it in the input buffer.
2. Read the data from the input buffer one at the time and convert in to corresponding Non Terminal using production rules available.
3. Perform push & pop operation for LR parsing table construction.
4. Display the result with conversion of corresponding input symbols to production and production reduction to start symbol. No operation performed on the operator.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
char stack[30];
int top=-1;
void push(char c)
{
        top++;
        stack[top]=c;
}
char pop()
{
        char c;
        if(top!=-1)
        {
                c=stack[top];
                top--;
                return c;
        }
        return'x';
}
void printstat()
{
        int i;
        printf("\n\t\t\t $");
        for(i=0;i<=top;i++)
        printf("%c",stack[i]);
}
void main()
{
```

```
        int i,j,k,l;
        char s1[20],s2[20],ch1,ch2,ch3;
        clrscr();
        printf("\n\n\t\t LR PARSING");
        printf("\n\t\t ENTER THE EXPRESSION");
        scanf("%s",s1);
        l=strlen(s1);
        j=0;
        printf("\n\t\t $");
        for(i=0;i<l;i++)
{
        if(s1[i]=='i' && s1[i+1]=='d')
        {
                s1[i]=' ';
                s1[i+1]='E';
                printstat(); printf("id");
                push('E');
                printstat();
        }
        else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='*' ||s1[i]=='/' ||s1[i]=='d')
        {
                push(s1[i]);
                printstat();
        }
   }
        printstat();
        l=strlen(s2);
        while(l)
        {
                ch1=pop();
                if(ch1=='x')
                {
                        printf("\n\t\t\t $");
                        break;
                }
```

```
                if(ch1=='+'||ch1=='/'||ch1=='*'||ch1=='-')
                {
                        ch3=pop();
                if(ch3!='E')
                {
                        printf("errror");
                        exit();
                }
                else
                {
                        push('E');
                        printstat();
                }
        }
                ch2=ch1;
}
getch();
}
```

OUTPUT:

LR PARSING
ENTER THE EXPRESSION
id+id*id-id
$
$id
$E
$E+
$E+id
$E+E
$E+E*
$E+E*id
$E+E*E
$E+E*E-
$E+E*E-id
$E+E*E-E
$E+E*E-E
$E+E*E
$E
$

RESULT:

Thus the program for construction of LR Parsing table is executed and verified.

Ex.No:9	GENERATION OF CODE FOR A GIVEN INTERMEDIATE CODE

AIM:

To write a program for the generation of assembly language code of relational operator.

ALGORITHM:

1. The three address code using the relational operator is get from the user.
2. Identifying the addressing mode of the given three address code.
3. Identify the relational operator used in the statement.
4. Generate and display the assembly language code.

PROGRAM:

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
void forswitch(char,int);
void conv(int);
char arg[10][10],op[5][2],ch[10],go[3][3],c[10];
void main()
{
        int i=-1,m=0,k=10;
        clrscr();
        cout<<"\t\t\tTHREE ADDRESS CODE";
        gotoxy(15,7);
        cout<<"OPERATOR";
        gotoxy(30,7);
        cout<<"ARGUMENT-1";
        gotoxy(45,7);
        cout<<"ARGUMENT-2";
        gotoxy(60,7);
        cout<<"GOTO";
        gotoxy(15,8);
        cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~";
        do
        {
                i++;
                gotoxy(2,k);
                printf("[%d]",i);
                gotoxy(18,k);
                scanf("%s",&op[i]);
                forswitch(op[i][0],i);
                gotoxy(33,k);
```

```
                scanf("%s",&arg[m+i]);
                gotoxy(48,k);
                scanf("%s",&arg[m+1+i]);
                gotoxy(61,k);
                scanf("%s",&go[i]);
                conv(m+i);
                conv(m+1+i);
                k++;
                m++;
        }while(i!=3);
        clrscr();
        printf("ASSEMBLY LANGUAGE CODE");
        printf("\n\n100\tMOV %s,RO",arg[0]);
        printf("\n101\tMOV %s,R1",arg[1]);
        printf("\n102\tCMP R0,R1");
        printf("\n103\t%s 107",ch);
        printf("\n104\tMOV%s,R2",arg[3]);
        printf("\n105\tMOV R2,%s",arg[2]);
        printf("\n106\tJUMP 109");
        printf("\n107\tMOV %s,R2",arg[5]);
        printf("\n109\tend");
        getch();
}
void conv(int x)
{
        if(isdigit(arg[x][0]))
        {
                strcpy(c,"#");
                strcat(c,arg[x]);
                strcpy(arg[x],c);
        }
}
void forswitch(char sh,int t)
{
        if(t<1)
```

```
        switch(sh)
        {
                case '<':
                        strcpy(ch,"JC");
                        break;
                case '>':
                        strcpy(ch,"JNC");
                        break;
                case '=':
                        strcpy(ch,"JZ");
                        break;
                case '-':
                        break;
                default:
                        gotoxy(8,40);
                        cout<<"\n\t\tINVALID ENTRY";
                        getch();
                        exit(0);
                        break;
        }
}
```

OUTPUT:

THREE ADDRESS CODE

	OPERATOR	ARGUMENT1	ARGUMENT2	GOTO
[0]	>	date	5	[2]
[1]	=	fine	50	[3]
[2]	=	fine	0	----
[3]	-	----	----	end

ASSEMBLY LANGUAGE CODE

```
100  MOV date,R0
101  MOV #5,R1
102  CMP  R0,R1
103  JNC   107
104  MOV #50,R2
105  MOV R2,fine
106  JUMP 109
107   MOV #0,R2
108  MOV R2,fine
109  END
```

RESULT:

Thus the program for generation of Machine Code for the given Intermediate code is executed and verified.

Ex.No:10	IMPLEMENTATION OF CODE OPTIMIZATION TECHNIQUES

AIM:

To write a program for implementation of Code Optimization Technique in for and do-while loop using C++.

ALGORITHM:

1. Generate the program for factorial program using for and do-while loop to specify optimization technique.
2. In for loop variable initialization is activated first and the condition is checked next. If the condition is true the corresponding statements are executed and specified increment / decrement operation is performed.
3. The for loop operation is activated till the condition failure.
4. In do-while loop the variable is initialized and the statements are executed then the condition checking and increment / decrement operation is performed.
5. When comparing both for and do-while loop for optimization do-while is best because first the statement execution is done then only the condition is checked. So, during the statement execution itself we can fin the inconvenience of the result and no need to wait for the specified condition result.
6. Finally when considering Code Optimization in loop do-while is best with respect to performance.

PROGRAM:

Before:

Using for:

```
#include<iostream.h>
#include <conio.h>
int main()
{
        int i, n;
        int fact=1;
        cout<<"\nEnter a number: ";
        cin>>n;
        for(i=n ; i>=1 ; i--)
        fact = fact * i;
        cout<<"The factoral value is: "<<fact;
        getch();
        return 0;
}
```

OUTPUT:

Enter the number: 5

The factorial value is: 120

After:

Using do-while:

```
#include<iostream.h>
#include<conio.h>
void main()
{
        clrscr();
        int n,f;
        f=1;
        cout<<"Enter the number:\n";
        cin>>n;
        do
        {
        f=f*n;
        n--;
        }while(n>0);
        cout<<"The factorial value is:"<<f;
        getch();
}
```

OUTPUT:

Enter the number: 3

The factorial value is: 6

RESULT:

Thus the program for implementation of Code Optimization technique is executed and verified.