# SRM UNIVERSITY

**Ramapuram, Chennai- 600089.**

## FACULTY OF ENGINEERING AND TECHNOLOGY

**Department of Computer Science & Engineering**

# LAB MANUAL

## COMPILER DESIGN LAB
## (CS0311)

**CLASS** : B.Tech. [U.G]

**YEAR / SEM.** : III Year / 5th Semester

**PREPARED BY**

| S.No. | Staff Name | Designation | Signature |
|---|---|---|---|
| 1. | Mr.S.Arunkumar | AP(OG) | |
| 2 | Ms.M.S.Antony vigil | AP(OG) | |

**Approved By**

**(HOD/CSE)**

`

# SYLLABUS:

| | | L | T | P | C |
|---|---|---|---|---|---|
| **PCS311** | **COMPILER DESIGN LAB** | **0** | **0** | **3** | **2** |
| | Prerequisite | | | | |
| | PCS204 | | | | |

**PURPOSE**

To Practice and implement the system software tools and compiler design techniques

**INSTRUCTIONAL OBJECTIVES**

1. To implement Loader, Linker, Assembler & Macro processor
2. To implement the NFA,DFA, First & Follow procedures
3. To implement Top down and Bottom up parsing techniques

**LIST OF EXPERIMENTS** **45**

1. Implementation of a Linker
2. Implementation of a Loader
3. Implementation of an Assembler
4. Implementation of Macro processor
5. Implementation of a Lexical Analyser
6. Converting a regular expression to NFA
7. Converting NFA to DFA
8. Computation of FIRST and FOLLOW sets
9. Construction of Predictive Parsing Table
10. Implementation of Shift Reduce Parsing
11. Computation of Follow and Trailing Sets
12. Computation of LR(0) items
13. Construction of DAG
14. Intermediate Code Generation
15. Design of Simple Compiler using Tamil words
16. Trace the execution of another program - debugger

**TOTAL** **45**

`

**TABLE OF CONTENTS:**

`

**EX.NO : 1**

# Implementation of a Linker

**AIM:** To write a C program to implement the linker concept

**ALGORITHM:**

1. Create file pointers
2. Accept the file name as the input
3. Open the file and check for #include
4. If present open that again and copy contents to linked.
5. Display names of files that are linked and total number of files linked
6. Terminate the program.

## SOURCE CODE:

### /*C Program to Illustrate the implementation of Linker*/

```
#include<stdio.h>
#include<conio.h>
#include<FILE.H>
#include<math.h>
void main()
{
        int n,p,a;
        clrscr();
        while(1)
        {
                printf("\n\n1.Factorial\n2.Fionacci series\n3.ODD/EVEN\n4.exit");
                printf("\n Enter your choice:");
                scanf("%d",&a);
                switch(a)
                {
                        case 1:printf("\nEnter the value of n:");
                               scanf("%d",&n);
                               fact(n);
                               break;
                        case 2:printf("\nEnter value of n:");
                               scanf("%d",&n);
                               fib(n);
                               break;
                        case 3:printf("\nEnter the value of n:");
                               scanf("%d",&n);
```

`

```
                        fun(n);
                        break;
                case 4:printf("\nProgram terminated");
                        exit(0);
                        break;
                default:printf("\nInvalid choice");
            }
        }
}
```

**OUTPUT-**

## /*C Program to Illustrate the implementation of Linker*/

```
1.Factorial
2.Fionacci series
3.ODD/EVEN
4.exit
 Enter your choice:1

Enter the value of n:3
Factorial of 3 = 6

1.Factorial
2.Fionacci series
3.ODD/EVEN
4.exit
 Enter your choice:2

Enter value of n:3

 Fibonnaci series:
0
1
1
2

1.Factorial
2.Fionacci series
3.ODD/EVEN
4.exit
 Enter your choice:3

Enter the value of n:3

Given num is Odd

1.Factorial
2.Fionacci series
```

`

3.ODD/EVEN
4.exit
Enter your choice:4

Program terminated

**RESULT:** Thus the Linker Program was executed and verified Successfully.

**EX.NO : 2**

# Implementation of a Loader

`

**AIM:** To write a C program to implement the loader

**ALGORITHM:**

1. Create file pointers
2. Accept the file name as the input
3. Open the input file and get length, starting address and location
4. Load the instruction from that location
5. Display the loaded code and its memory location
6. Terminate the program.

## SOURCE CODE:

### /*C Program to Illustrate the implementation of Loader*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,n;
        int irl[30],orl[20],line,sta,ard,rrl,len;
        char name[10],hex[30][5],rop[2];
        clrscr();
        printf("\nEnter name of the program:");
        scanf("%s",name);
        printf("\nEnter the length");
        scanf("%d",&len);
        printf("\nEnter the input program\n\nRelative addr\tHexcode\n");
        i=0;
        do
        {
                scanf("%d",&irl[i]);
                gets(hex[i]);
                i++;
        }
        while(irl[i-1]<len);
        line=i;
        printf("\nEnter the starting addess:");
        scanf("%d",&sta);
        printf("\nEnter the details of RLD:");
        printf("\nEnter the operator:");
        scanf("%s",rop);
        printf("\nEnter the relocative location:");
        scanf("%d",&rrl);
        for(i=0;i<line;i++)
        {
```

`

```
                if(irl[i]==rrl)
                {
                        if(strcmp(rop,"+")==0)
                        {
                                n=sta+atol(hex[i]);
                                sprintf(hex[i],"%d",n);
                        }
                        if(strcmp(rop,"-")==0)
                        {
                                n=sta-atol(hex[i]);
                                sprintf(hex[i],"%d",n);
                        }
                        break;
                }
        }
        printf("\nThe program %s is loaded as:\n",name);
        printf("\nMemory location   hexcode\n");
        for(i=0;i<line;i++)
        {
                orl[i]=irl[i]+sta;
        }
        for(i=0;i<line;i++)
        {
                printf("\n%d\t%s",orl[i],hex[i]);
        }
        getch();
}
```

**OUTPUT-**

## /*C Program to Illustrate the implementation of Loader*/

```
Enter  name of the Program     doc

Enter the length                3
Enter  the  input  program

Relative Address        HexaCode

    0                       10
    1                       20
    2                       40

Enter the starting address       =3000
Enter the Details of             RLD
Enter the Operator               +
Enter the relocative location    1
```

`

The program doc is loaded as:

| Memory location | Hexacode |
|---|---|
| 3000 | 10 |
| 3001 | 3020 |
| 3003 | 40 |

**RESULT:** Thus the above Loader Program was executed and verified Successfully.




**EX.NO : 3**

# Implementation of an Assembler

**AIM:** To write a C program for the Implementation of an Assembler

`

**ALGORITHM:**

1. Accept the starting address from the user
2. Now accept the program in assembly language from the user
3. Assign the correct opcodes for each of the commands
4. Generate address for each line and print the corresponding opcode
5. The final output is generated only when the command is entered which indicates ending of program
6. Stop the program

## SOURCE CODE:

**/*Program to Illustrate the implementation of Assembler*/**

```
#include<stdio.h>
#include<conio.h>
 #include<process.h>
 #include<stdlib.h>
 #include<string.h>
Int start id,id1;
Void main()
{
FILE *f1,*f2;
IntI,no;
Char str[100];
Clrscr();
Printf("\n enter the starting address:");
Scanf("%d",&start);
F1=fopen("z:\\sample.txt","w");
Printf("\n enter the assembly language program:");
While(1)
{
Scanf("%s",&str);
Fwrite(&str,sizeof(str=),1,f1);
If(strcmp(str,"hlt")==0)
{
Fwrite(&str,sizeof(str),1,f1);
Break;
}}
Fclose(f1);
F1=fopen("z:\\sample.txt","r");
Pritnf("\n");
While(1)
{
L1:
```

`

```
Fread(&str,sizeof(str),1,f1);
If(strcmp(str,”mov”)==0)
{
Fread(&str,sizeof(str),1,f1);
If(strcmp(str,”a,b”)==0)
{
Printf(“%d 3e\n”,start++);
Goto l1;
}
If(strcmp(str,”b,c”)==0)
{
Printf(“%d 25 \n”,start++);
Goto l1;
}}
If(strcmp(str,”sub”)==0)
{
Printf(“%d 3a\n”,start++);
Goto l1;
}
If(strcmp(str,”sta”)==0)
{
Printf(“%d 5e\n”,start++);
Fread(&str,sizeof(str),1,f1);
No=atoi(str);
Id=no;
Id=no%100;
No=no/100;
Id1=no;
If(id==0)
{
Printf(“%d%d%d\n”,start++,id,id);
Gotoabc;
}
Printf(“%d%d \n”,start++,id);
Abc:
Printf(“%d%d \n”,start++,id1);
}
If(strcmp(str,”lxi”)==0)
{
Printf(“%d 7d\n”,start++);
Fread(&str,sizeof(str),1,f1);
No=atoi(str);
Id=no;
Id=no%100;
No=no/100;
Id1=no;
If(id==0)
{
Printf(“%d%d%d\n”,start++,id,id);
Gotoab;
}
```

```
`

Printf("%d%d \n",start++,id1);
Ab:
Printf("%d%d \n",start++,id1);
}
If(strcmp(str,"hlt")==0)
{
Printf("%d 7b\n",start++);
Break;
}}
Fclose(f1);
getch();
}
```

**OUTPUT:**

## /*Program to Illustrate the implementation of Assembler*/

```
Enter the start address: 1000
Enter the assembly language program:
Mova,b
Sta @4200
Movb,c
Str @4300
Net

1000 3e
1001 5e
1002 00
1003 0
1004 25
1005 5e
1006 00
1007 0
1008 76
```

| LOCATION | MNEMONICS | OBJECT CODE | LENGTH |
|---|---|---|---|
| 2003 | MUL 3 | 80 | 4 |
| 2001 | ADD 1 | 40 | 2 |
| 2004 | DIV 4 | 100 | 5 |
| 2000 | LDX 0 | 20 | 1 |

`

| | | | |
|---|---|---|---|
| 2005 | TIX 5 | 120 | 6 |
| 2002 | SUB 2 | 60 | 3 |

**RESULT:**Thus the assembler program was successfully executed




**EX.NO : 4**

# Implementation of Macro Processor

`

**AIM:** To write a C program to implement the Macro Processor concept

**ALGORITHM:**

1. Start the program
2. Create 2 file pointers
3. Get the program from user and compare whether it's a macro
4. If yes replace the macro with its definition
5. Write the new program in a new file
6. Display the contents of the new file.
7. Stop the program

## SOURCE CODE:

## /*C Program to Illustrate the implementation of Macro Processor*/

## Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char a,name[20],array[100],ch[50];
        FILE *f1,*f2;
        f1=fopen("in2.txt","r");
        f2=fopen("out2.txt","w");
        fscanf(f1,"%s",&ch);
        clrscr();
        printf("%s",ch);
        strcpy(array," ");
        while(strcmp(ch,"mend")!=0)
        {
                if(strcmp(ch,"marco")==0)
                        fscanf(f1,"%s",&name);
                fscanf(f1,"%s",&ch);
                if(strcmp(ch,"mend")!=0)
                        strcat(array,ch);
                a=fgetc(f1);
                if(a=='\n')
                strcat(array,"\n");
        }
        printf("%s",array);
        fscanf(f1,"%s",ch);
        while(strcmp(ch,"end")!=0)
```

```
`
        {
                if(strcmp(ch,name)==0)
                        fprintf(f2,"%s",array);
                else
                        fprintf(f2,"%s",array);
                a=fgetc(f1);
                if(a=='\n')
                        fprintf(f2,"\n");
                fscanf(f1,"%s",ch);
        }
        fprintf(f2,"%s",ch);
}

    .
```

*****IN2.TXT*****

```
macro
fruits
mango
grapes
orange
pineapple

mend
start
strawbeery
friuts
end
```

**/*C Program to Illustrate the implementation of Macro Processor*/**

*****OUTPUT*****

```
macro fruits
mango
grapes
orange
pineapple
```

**RESULT:**

Thus the assembler program was successfully executed

**EX.NO : 5**

# Implementation of a Lexical Analyzer

`

**AIM:** To write a C program to Implement the Lexical Analyer

**ALGORITHM:**

1. Start the program
2. Accept the input from the user
3. Check using the function it is alphabet or not
4. If it then check for the identifier
5. Stop the program

## SOURCE CODE:

### /*C Program to Illustrate the implementation of Lexical Analyzer*/

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
void main()
{
        int i=1,l;
        char a[15];
        clrscr();
        printf("Enter the string:")
        scanf("%s",&a);
        l=strlen(a);
        for(i=0;i<l;i++)
        {
                if(isalpha(a[i]))
                        printf("\n%c is a character",a[i]);
                else if(isdigit(a[i]))
                        print("\n%c is a digit",a[i]);
                else if((a[i]>='a'&&a[i]<='z')|| (a[i]>='A'&&a[i]<='Z'))
                        printf("\n%c is character",a[i]);
                else if(a[i]=='*'||a[i]='+'||a[i]='-'||a[i]='/')
                        printf("\n%c is an operator",a[i]);
                else if(a[i]=='@'||a[i]=='$'||a[i]=='%')
                        printf("\n%c is a Special character",a[i]);
                else
                        printf("\n%c is number",a[i]);
        }
        getch();
}
```

`

**OUTPUT-**

## /*C Program to Illustrate the implementation of Lexical Analyzer*/

Enter the string: hello+wel
h is a character
e is a character
l is a character
l is a character
o is a character
+ is a operator
w is a character
e is a character
l is a character

**RESULT:** Thus the **Lexical Analyzer** was executed and verified successfully.

**EX.NO : 6**

# Finding epsilon closure from regular expression

**AIM:** To write a program to find the e-closure of all states from RE

`

**ALGORITHM:**

1. Start the program
2. Declare and initialise variables and arrays
3. Check whether it is a kleen closure or positive closure
4. If kleen closure perform the necessary steps also perform positive closure
5. Show the various states of e-closure
6. Stop the program

## SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Char ip[10],p,s[10][10];
Int i=0,j=0,n=0;c=0;
Clrscr();
Printf("\n enter the input:");
For(i=0;i<3;i++)
{
Scanf("%c",&ip[i]);
C=n++;
}
S[0][1]='e';
P=ip[1];
Printf("\n no of states are: %d",n+1);
If(c==2)
{for(i=o;i<n;i++)
for(j=o;j<n;j++)
s[i][j]='0';
switch(p)
{
Case'*':
S[0][1]='e';
S[1][2]=ip[0];
S[0][3]='e';
S[2][1]='e';
Break;
Case '+':
S[0][1]='e';
S[1][2]=ip[0];
S[0][3]='e';
S[2][1]='e';
Break;
}
For(i=0;i<4;i++)
```

`

```
{printf(“\n e-closure of(%d)”,i,i);
For(j=0;j<4;j++)
If(s[i][j]==’e’)
Printf(“&d”,j)
}}
Getch();}
```

**OUTPUT:**

Enter the input:a*bb

no of states: 4

e-closure of (0)=0,0,1,2,3

e-closure of (1)=1,0,1,2,3

e-closure of (2)=2,0,1,2,3

e-closure of (3)=3,0,1,2,3

**RESULT:** Thus the Epsilon closure program is successfully executed

**EX.NO : 7**

# Converting NFA to DFA

**AIM:** To write a program to convert NFA to DFA

`

**ALGORITHM:**

1. Start the program
2. Assign an input string terminated by end of file, DFA with start
3. The final state is assigned to F
4. Assign the state to S
5. Assign the input string to variable C
6. While C!=e of do
   S=move(s,c)
   C=next char
7. If it is in ten return yes else no
8. Stop the program

## SOURCE CODE:

```
#include<conio.h>
#include<string.h>
#include<process.h>
#include<math.h>
Int n[11],I,j,c,k,h,l,h1,f,h2,temp1[12],temp2[12],count=0,ptr=0;
Char a[20][20],s[5][8];
Inttr[5][2],ecl[5][8],str[5],flag;
Inttr[5][2],ecl[5][8],st[5],flag;
Void ecls(int b[10],int x)
{
I=0;
K=-1;flag=0;
While(l<x)
{
N[++k]=b[l];
I=b[l];
h=k+1;
a:
for(j=I;j<=11;j++)
{
If(a[i][j]==’e’)
{n[++k]=j;
}
If(j==11&&h<=k)
{
I=n[h];
H++;
Goto a;
}}
L++;
}for(i=0;i<k;i++)
```

```
`

for(j=i+1;j<k;j++)
if(n[i]>n[j])
{
C=n[i];
N[i]=n[j];
N[j]=c;
}for(i=0;i<ptr;i++)
for(j=0;j<k;j++)
{
If(ecl[i][j]!=n[j])
{
If(i<count)
{i++;
J=0;
}
Else
Goto b;
}
Else if((ecl[i][j]==n[j])&&(j==k))
{tr[ptr][f]=st[i];
Flag=1;
Break;
}}
B:
If(flag==0)
{for(i=0;i<=k;i++)
Ecl[count][i]=n[i];
st[count]=count+65;
tr[ptr][f]=st[count];
count++;
}}
Void mova(int g)
{h1=0;
for(i=0;i<7;i++)
{if(ecl[g][i]==3)
Temp1[h1++]=4;
if(ecl[g][i]==8)
Temp1[h1++]=9;
}
Printf("\n move(%c,a):",st[g]);
For(i=0;i<h1;i++)
Printf("%d",temp1[i]);
F=0;
Ecls(temp1,h1);
}
Void movb(int g)
{
H2=0;
For(i=0;i<7;i++)
{
If(ecl[g][i]==5)
```

`

```
Temp2[h2++]=6;
If(ecl[g][i]==9)
Temp2[h2++]=10;
If(ecl[g][i]==10)
Temp2[h2++]=11;}
Printf("move(%c,b):"st[g]);
For(i=0;i<h2;i++)
Printf("%d",temp2[i]);
F=1;
Ecls(temp2,h2);
}
Void main()
{
Clrscr();
Printf("\n the no. of states in NFA (a/b)*abb are:11");
For(i=0;i<=11;i++)
For(j=0;j<=11;j++)
A[i][j]='\0';
A[1][2]='e';
A[1][8]='e';
A[2][3]='e';
A[2][5]='e';
A[3][4]='a';
A[5][6]='b';
A[4][7]='e';
A[6][7]='e';
A[7][8]='e';
A[7][2]='e';
A[8][9]='a';
A[9][10]='b';
A[10][11]='b';
Printf("\n the transmission table is as Follows");
Printf("\n states 1 2 3 4 5 6 7 8 9 10 11");
Getch();
For(i=1;i<=11;i++)
{
Printf("\n  %d \t",i);
For(j=1;j<=11;j++)
Printf("%c",a[i][j]);
}
Getch();
Printf("\n \n press any key to continue");
Clrscr();
I=1;k=1;h=1;
N[0]=I;
Printf("\n");
A:
For(j=1;j<=11;j++)
{if(a[i][j]=='e')
{
N[k++]=j;
```

```
`

}
If(j==11&&h<k)
{
I=n[h];
H++;
Goto a;
}}
For(i=1;j<k;i++)
For(j=i+1;j<k;j++)
If(n[i]>n[j])
{c=n[i];
N[i]=n[j];
N[j]=c;
}
Count++;
St[0]=65;
For(i=0;i<k;i++)
Ecl[0][i]=n[i];
Printf("the moves are of the Following manner");
Mova(ptr);
Movb(ptr);
Ptr++;}
Printf("\n the new states of DFA are as Follows");
For(i=0;i<5;i++)
{printf("\n %c",st[i]);
For(j=0;j<7;j++)
Printf"("%d",ecl[i][j]);
}
Printf(" the transition table are as Follows");
Printf("\n a \n b \n");
For(i=0;i<5;i++)
{
Printf("%c",st[i]);
For(j=0;j<2;j++)
Printf("%c \t",tr[i][j]);
}
Getch();
}
```

**OUTPUT:**

The no. of states in NFA(a/b)*abb are:11

The transition table is as Follows

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | e | | e | | | | | | | | |
| 2 | | e | | e | | | | | | | |
| 3 | | a | | | | | | | | | |
| 4 | | | | e | | | | | | | |

`

| | | | |
|---|---|---|---|
| 5 | | b | e |
| 6 | | | e |
| 7 | e | | e |
| 8 | | | |
| 9 | | | a |
| 10 | | | b |
| 11 | | | b |

**RESULT:**

Thus the above conversion program is successfully executed

**EX.NO : 8**

# Computation of FIRST and FOLLOW sets

**AIM:** To calculate the first and Follow of the given expression

`

**ALGORITHM:**

1. Start the program
2. In the production the first terminal on R.H.S becomes the first of it
3. If the first character is non-terminal then its first is taken else Follow of left is taken
4. To find Follow find where all the non terminals appear. the first of its Follows is its Follow
5. If the Follow is t then the Follow of left is taken
6. Finally print first and its Follow
7. Stop the program

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
Void main()
{
Intnop,x=0,y=0,l=0,k=0,c=0,s=0,z=0;
Char p[10][10],o,fi[10][10],fo[10][10];
Clrscr();
For(x=0;x<10;x++)
{
For(y=0;y<10;y++)
{
P[x][y]='0';
Fi[x][y]='0';
Fo[x][y]='0';
}}
Printf("enter the no of productions:");
Scanf("%d",&nop);
Printf("\n enter the number of production");
For(x=0;x<nop;x++)
{
Scanf("%s",&p[x]);
If(p[x][0]==p[x][2])
{
Printf("\a production is not free from left recursion");
Printf("\a program has to be terminated");
Printf("\a press any key");
Getch();
Exit(0);
}}
Printf("\n\n")
Printf("first \n");
For(y=0;y<nop;y++)
{
```

`

```
Printf(“first(%c)=”,p[y][0]);
If(p[y][2]>=’A’&&p[y][2]<=’z’)
{
O=p[y][2];
For(x=y+1;x<nop;x++)
{
If(p[x][0]==o)
{
If(p[x][2]>=’A’&&p[x][2]<=’z’)
O=p[x][2];
Else if(p[x][2]<’A’||p[x][2]>’z’)
{
Printf(“%c”,p[x][2]);
Fi[y][k++]=p[x][2]);
For(l=0;l<strlen(p[x]);l++)
{
If(p[x][l]==’/’)
{
Printf(“%c”,p[x][l+1]);
Fi[y][k++]=p[x][l+1];
Break;
}}}}}
Else if(p[y][2]<’A’ || p[y][2]>’Z’)
{
Printf(“%c”,p[y][2]);
Fi[y][k++]=p[y][2];
}
L=strlen(p[y]);
For(c=0;c<l;c++)
{
If(p[y][c]==’/’)
{
Printf(“%c”,p[y][c+1]);
Fi[y][k++]=p[y][c+1];
}}
Printf(“\n”);
K=0;
Printf(“Follow \n”);
For(x=0;x<nop;x++)
{
For(y=0;y<nop;y++)
For(l=2;l<strlen(p[x]);l++)
If(p[y][l]==p[x][0])
{
If(p[y][l+1]<’A’||p[y][l+1]>’Z’||p[y][l+1]!=’/’||p[y][l+1]!=’0’)
{
Fo[x][k++]=p[y][l+1];
If(x==0)
{
Fo[x][k++]=’$’;
}}
```

```
`

C=k;
If(p[y][l+1]==’0’||p[y][l+1]==’/’)
{
For(s=0;s<c+10;s++)
{
Fo[x][k++]=fo[x-1][s];
}}
C=k;
If(p[y][l+1]>=’A’&&p[y][l+1]<=’z’)
{
For(s=0;s<=c;s++)
{
Fo[x][k++]=fo[x-1][s];
Fo[x][k++]=fo[x-2][s];
Fo[x][k++]=fi[x-1][s];
}}
C=k;
}
Printf(“Follow(%c)=”,p[x][0]);
For(z=0;z<=k+10;z++)
If(fo[x][z]==’*’||fo[x][z]==’+’||fo[x][z]==’$’||fo[x][z]==’)’||fo[x][z]==’(‘||fo[x][z]==’-‘||fo[x]
[z]==’%’)
Printf(“%c”,fo[x][z]);
K=0;
Printf(“\n”);
}
Getch();
}
```

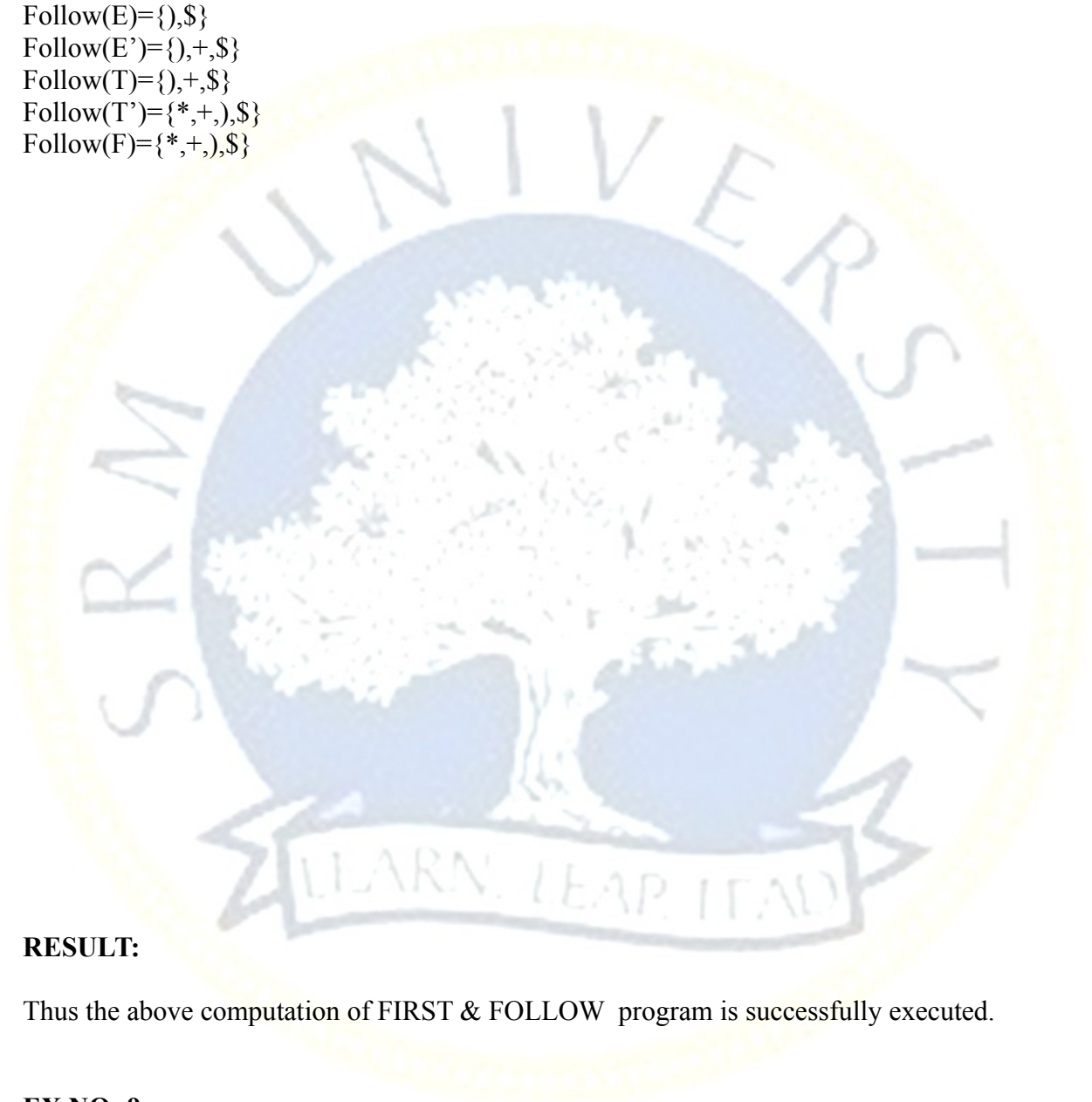
**OUTPUT:**

```
Enter the no. of production:5
E->TE’
E’->+TE’/e
T->FT’
T’->*FT’/e
F->(E) /id
```

`

First
First(E)={(,id}
First(E')={+,e}
First(T)={(,id}
First(T')={*,e}
First(F)={(,id}

Follow
Follow(E)={),$}
Follow(E')={),+,$}
Follow(T)={),+,$}
Follow(T')={*,+,),$}
Follow(F)={*,+,),$}



**RESULT:**

Thus the above computation of FIRST & FOLLOW program is successfully executed.

**EX.NO :9**

# Construction of Predictive Parsing Table

**AIM:** To write a C program for the implementation of predictive parsing table

`

**ALGORITHM:**

1. start the program
2. Assign an input string and parsing table in for then G.
3. Set ip to point to the first symbol of to $
4. Repeat if X is a terminal of $ then if n=a,then pop X from the stack
5. Push Y into the stack with Y,on top
6. Output the production x->x,y
7. End else error until x=$
8. Terminate the program

## SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char str[10],out,in,output[10],input[10],temp;
char tl[10]={'x','+','*','(',')','$','@'};
char ntl[10]={'e','e','t','t','f'};
int err=0,flag=0,i,j,k,l,m;
char c[10][10][7]={{{"te"},{"error!"},{"error!"},{"te"},{"error!"},{"error!"},},
{"error!","te","error!","error","@","@"},{"ft","error!","error","ft""error!","error"},
{"error","@","*ft","error!","@","@"},{"x","error!","error!","(e)","error!","error!"}};
struct stack
{
char sic[10];
int top;
};
void push(struct stack *s,char p)
{
s->sic[++s->top]=p;
s->sic[s->top+1]='\0';
}
char pop(struct stack *s)
{
char a;
a=s->sic[s->top];
s->sic[s->top--]='\0';
return(a);
}
char stop(struct stack *s)
{
return(s->sic[s->top]);
}
void pobo(struct stack *s)
{
```

```
`

m=o;
while(str[m]!='\0')
m++;
m--;
while(m!=-1)
{
if(str[m]!='@')
push(s,str[m]);
m--;
}}
void search(int l)
{
for(k=0;k<7;k++)
if(in==tl[k])
break;
if(l==0)
strcpy(str,c[l][k]);
else if(l==1)
strcpy(str,c[l][k]);
else if(l==2)
strcpy(str,c[l][k]);
else if(l==3)
strcpy(str,c[l][k]);
else strcpy(str,c[l][k]);}
void main()
{
struct stack s1;
struct stack *s;
s=&s1;
s->top=-1;
clrscr();
printf("\t\t parsing table \t\t");
for(i=0;i<5;i++)
{
printf("%c\t",ntl[i];
for(j=0;j<6;j++)
if(strcmp(c[i][j],"error!")==0)
printf("error!\t");
else
printf("%c->%s" \t",ntl[i],c[i][j]);
}
push(s,'$');
push(s,'e');
printf("enter the input string");
scanf("%s",input);

printf("\n\n the behaviour of the parser for given input string is: \n\ ");
printf("\n stack\n input\n output");
i=0;
in=input[i];
printf("%s\t",s->sic);
```

`

```
for(k=i;k<strlen(input);k++)
printf("%c",input[k]);
if(strcmp(str,' ')!=0)
printf("\t%c->%s"ntl[j],str);
while((s->sic[s->top]!='$')&&err!=1&&strcmp(str,"error!")!=0)
{
strcpy(str," ");
flag=0;
for(j=0;j<7;j++)
if(in==tl[j])
{
flag=1;
break;
}
if(flag==0)
in='x';
flag=0;
out=stop(s);
for(j=0;j<7;j++)
if(out==tl[j])
{
flag=1;
break;
}
if(flag==1)
{
if(out==in)
{
temp=pop(s);
in=input[++i];
if(str=='@')
temp=pop(s);
}
else
{
strcpy(sstr,"error!");
err=1;
}}
else
{
flag=0;
for(j=0;j<5;j++)
if(out==ntl[j])
{
flag=1;
break;
}
if(flag==1)
{
search(j);
temp=pop(s);
```

```
`

pobo(s);
}
else
{
strcpy(str,"error!");
err=1;
}}
if(strcmp(str,"error!")!=0)
{
printf("%s\t",s->sic);
for(k=i;k<strlen(input);k++)
printf("%c",input[k]);
if((strcmp(str," ")!=0)&&(strcmp(str,"error!")!=0))
printf("\t %c->%s",ntl[j],str);
}}
if(strcmp(str,"error!")==0)
printf("\n the string is not accepted!!");
else
printf("\t \t accept \n\n\n the string is accepted");
getch();
}
```

**OUTPUT:**

Parsing table

```
X       +       *       (       )       $
E.E->Te ERROR! ERROR! E->teERROR! ERROR!
E ERROR! E->+teERROR! ERROR! E->@ e->@
T T->Ft ERROR! ERROR! T->ftERROR! ERROR!
T ERROR! T->@ t->*ft ERROR t->@ t->@
F.F->x ERROR! ERROR! F-> (E) ERROR! ERROR!

Enter the input string: x+X$
The behaviour of the parser for given input string is

Stack                   input                   output
SE                              X+X$
SeT                             X+X$                    E->Te
Set F                   X+X$                    T->Ft
Set X                   X+X$                    F->X
Set                             +X$
Se                              +X$                             t->@
SeT+                    +X$                     e->+Te
seT                             X$
SetF                            X$                              T->Ft
```

`

**RESULT:**
Thus the Predictive Parser program is executed successfully.
**EX.NO : 10**

# Implementation of Shift Reduce Parsing

**AIM:** To write a C program for shift reduce parsing

`

**ALGORITHM:**

1. start the program
2. read the expression and declare the variables
3. set $ symbol to indicate the start of stack
4. Repeat for i=0 to n where n is the no. of productions

    A. read prod(i)
    B.set problem[i]=strlen(prod[i])

5. check for the non-terminla which corresponds to the terminal
6. If it equals then replace the terminal with non-terminal
7. Repeat this until terminals are replaced by non-terminals

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void push(char);
char pop();
void printstack();
struct grammar
{
char lpr,rpr[10];
};char stack[20];
int top=-1;
void main()
{
grammar gr[10];
char buffer[10];
char ch,ch1,temp[10],start;
int i,j,k,s,t,len;
clrscr();
cout<<"enter the no of productions:";
cin>>n;
for(i=0;i<n;i++)
{
cout<<"\n enter the left side of productions"<<i+1<<":";
cin>>gr[i].lpr;
cout<<"\n enter the right side of productions"<<":";
cin>>gr[i].rpr;
}
cout<<"\n enter the input string:";
```

```
`

cin>>buffer;
cout<<"\n the grammaris:\n";
for(i=0;i<n'i++)
cout<<gr[i].lpr<<"-->"<<gr[i].rpr<<endl;
cout<<"\n input string is :"<<buffer;
push('$');
start=gr[0].lpr;
len=strlen(buffer);
buffer[len]='$';
buffer[len+1]='\0';
cout<<"\n \n stack \t\t buffer\t\t\t action\n";
cout<<stack<<"\t\t"<<buffer<<endl;
getch();
while(1)
{
ch=buffer[i];
lab:t=0;
for(k=top;k>0;k--)
{
temp[t++]=stack[t]='\0';
strrev(temp);
for(j=0;j<n;j++)
{
if(strcmp(temp,gr[j].rpr)==0)
{
for(s=0;s<t;s++)
ch1=pop();
push(gr[j].lpr);
printstack();
cout<<"\t\t\t"<<&buffer[i]<<"\t\tReduce"<<endl;
getch();
goto lab;
}}
strrev(temp);
}
ch1=pop();
if(ch!='$')
{push(ch1);
push(ch);
printstack();
cout<<"\t\t\t"<<&buffer[i+1]<<"\t\tshift"<<endl;
getch();
i++;
}
else if(ch=='$' && ch1==start && top==0)
{
cout<<"\n string is accepted";
getch();
exit(0);
}
else
```

```
`

{
cout<<"\n string is not accepted";
getch();
exit(0);
}
}
}
void push(char a)
{
stack[++top]=a;
}
char pop(){
return stack[top--];}
void printstack()
for(int i=0;i<top;i++)
cout<<stack[i];
}
```

**OUTPUT:**

```
Enter the no. of production: 3
Enter the production: S->aABC
A->Abc/b
B->d

The production are
S>>aABC
A>>Abc/b
B>>d
ILR
S>>aABC
u>>bc/bu
B>>d
u>x
the first symbols are X
```

**RESULT:**

Thus the shift reduce parser program is successfully executed

**EX.NO : 11**

# Computation of Follow and Trailing Sets

**AIM:** To write a C program to Compute of Follow and Trailing Sets

**ALGORITHM:**

`

Step1 : Start the Program

Step2: Get the no of production and calculate the length of each production.

Step3: With a variable val for checking the valid non terminals if they are duplicated get all Non terminals in an array.

Step4: In each production check the first accurate of terminals and take that terminal and add it to Follow of non terminal in array and exitthe loop.

Step5: Scan the production and find the last terminal and add it to respective trailing array of associated non terminal &exit the loop.

Step6: Consider production with a non terminal on right side and check the Follow of that non terminal associated production and also trailing and add it to the Follow and trailing of left side non terminal.

Step7: Write the Follow and trailing terminals for each non terminal.

## SOURCE CODE:

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

char nt[5],p[5],q[5],a[5][5][5],b[5][5][5],fi[5][5],left[5],right[5],lead[5][10],trail[5][10];
int n1,n[5],c[5][5],m,l[5],f[5],k,a1;

void leading(char,int);
void trailing(char,int);

void main()
{
    clrscr();
    cout<<"Enter the number of non-terminals ";
    cin>>n1;
    cout<<"Enter the set of non-terminals ";
    for(int i=0;i<n1;i++)
          cin>>nt[i];
    for(i=0;i<n1;i++)
    {
      cout<<endl;
      cout<<"Enter the number of productions for "<<nt[i]<<" ";
      cin>>n[i];
```

```
`
  for(int j=0;j<n[i];j++)
  {
        cout<<"\nEnter the productions ";
        gets(p);
        //q=strrev(p);
        c[i][j]=strlen(p);
        for(int k=0;k<c[i][j];k++)
         {
           a[i][j][k]=p[k];
           b[i][j][k]=p[c[i][j]-k-1];
  } }
}
for(i=0;i<n1;i++)
{
      cout<<endl;
      cout<<nt[i]<<"--->";
      for(int j=0;j<n[i];j++)
      {
        for(int k=0;k<c[i][j];k++)
         cout<<a[i][j][k];
           cout<<"/";
}  }
cout<<"\n\n";
char x;
for(i=0;i<n1;i++)
{
l[i]=0;
f[i]=0;
x=nt[i];
k=i;
leading(x,k);
trailing(x,k);
}
/*for(int mn=0;mn<=n1;mn++)
{
cout<<right[mn]<<" ";
} */
int count=0;
char z;
for(int mn=0;mn<n1;mn++)
{
for(i=0;i<n1;i++)
{
        z=right[i];
        for(int k=0;k<n1;k++)
        {
              if(z==nt[k])
              {
                for(int d=0;d<l[k];d++)
                 {
                       for(int g=0;g<l[i];g++)
```

```
`
                        {
                          if(lead[i][g]==lead[k][d])
                          count=1;
                        }
                        if(count==0)
                        {
                          lead[i][l[i]-1]=lead[k][d];
                          l[i]++;
                        }
                        count=0;
                    }
                     for(d=0;d<f[k];d++)
                    {
                        for(int g=0;g<f[i];g++)
                        {
                          if(trail[i][g]==trail[k][d])
                          count=1;
                        }
                        if(count==0)
                        {
                          trail[i][f[i]-1]=trail[k][d];
                          f[i]++;
                        }
                        count=0;
                    }

                }
            }
           count=0;
 }
 }

 for(i=0;i<n1;i++)
 {
 cout<<"Leading("<<nt[i]<<")= {";
 for(int j=0;j<l[i];j++)
{
cout<<lead[i][j]<<" ";
}

 cout<<"}\t\t\t\t";

 cout<<"Trailing("<<nt[i]<<")= {";
 for(j=0;j<f[i];j++)
{
cout<<trail[i][j]<<" ";
}

 cout<<"}\n";
 }
 getch();
```

```
`

}

void leading(char x,int i)
{
   char z;
   for(int j=0;j<n[i];j++)
    {
    if(isupper(a[i][j][0]))
    {
    left[i]=nt[i];
    right[i]=a[i][j][0];
    if(!isupper(a[i][j][1]))
    {
    //cout<<a[i][j][1]<<" ";
    lead[i][l[i]]=a[i][j][1];
    l[i]++;
    }
    }
    else
     {
      //cout<<a[i][j][0]<<" ";
      lead[i][l[i]]=a[i][j][0];
      l[i]++;
     }
  }
}


void trailing(char x,int i)
{
   char z;
   for(int j=0;j<n[i];j++)
   {
    if(isupper(b[i][j][0]))
    {
    if(!isupper(b[i][j][1]))
    {
    //cout<<a[i][j][1]<<" ";
    trail[i][f[i]]=b[i][j][1];
    f[i]++;
    }
    }
    else
     {
      //cout<<a[i][j][0]<<" ";
      trail[i][f[i]]=b[i][j][0];
      f[i]++;
     }
  }
}
```

`

**OUTPUT-**

Enter the no. of production:5
E->TE'
E'->+TE'/e
T->FT'
T'->*FT'/e
F->(E) /id

Leading
Leading (E)={(,id}
Leading (E')={+,e}
Leading (T)={(,id}
Leading (T')={*,e}
Leading (F)={(,id}

Trailing
Trailing (E)={),$}
Trailing (E')={),+,$}
Trailing (T)={),+,$}
Trailing (T')={*,+,),$}
F Trailing (F)={*,+,),$}

**RESULT:** Thus the Leading and Trailing was executed and verified Successfully

**EX.NO : 12**

# Computation of LR (0) items

**AIM:** **To write a code for LR(0) Parser for Following Production:**
**E->E+T**
**T->T*F/F**
**F->(E)/char**

`

**ALGORITHM:**

**1.Initialize the stack with the start state.**
**2. Read an input symbol**
**3. while true do**
**3.1 Using the top of the stack and the input symbol determine the next state.**
**3.2 If the next state is a stack state**
**then3.2.1 stack the state**
**3.2.2 get the next input symbol**
**3.3 else if the next state is a reduce state**
**then**
**3.3.1 output reduction number, k**
**3.3.2 pop RHSk -1 states from the stack where RHSk is the right hand side of production k.**
**3.3.3 set the next input symbol to the LHSk**
**3.4 else if the next state is an accept state**
**then**
**3.4.1 output valid sentence**
**3.4.2 return**
**else**
**3.4.3 output invalid sentence**
**3.4.4 return**

## SOURCE CODE:

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

int axn[][6][2]={
                {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
                {{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},
                {{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},
                {{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},
                {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
                {{100,5},{101,6},{101,6},{-1,-1},{101,6},{101,6}},
                {{100,5},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}},
                {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
                {{-1,-1},{100,6},{-1,-1},{-1,-1},{100,11},{-1,-1}},
                {{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
                {{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
                {{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
};

int gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1,-1,
                  9,3,-1,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

int a[10];
char b[10];
```

```
`

int top=-1,btop=-1,i;
void push(int k)
{
        if(top<9)
        a[++top]=k;
}
void pushb(char k)
{
        if(btop<9)
        b[++btop]=k;
}
char TOS()
{
        return a[top];
}
void pop()
{
        if(top>=0)
        top--;
}

void popb()
{
        if(btop>=0)
        b[btop--]='\0';
}

void display()
{
        for(i=0;i<=top;i++)
                printf("%d%c",a[i],b[i]);
}

void display1(char p[],int m)
{
        int l;
        printf("\t\t");
        for(l=m;p[l]!='\0';l++)
                printf("%c",p[l]);
                printf("\n");
}

void error()
{
        printf("\n\nSyntax Error");
}
void reduce(int p)
{
        int len,k,ad;
        char src,*dest;
        switch(p)
        {
```

```
`
                case 1:dest="E+T";
                        src='E';
                        break;
                case 2:dest="T";
                        src='E';
                        break;
                case 3:dest="T*F";
                        src='T';
                        break;
                case 4:dest="F";
                        src='T';
                        break;
                case 5:dest="(E)";
                        src='F';
                        break;
                case 6:dest="i";
                        src='F';
                        break;
                default:dest="\0";
                        src='\0';
                        break;
        }
        for(k=0;k<strlen(dest);k++)
        {
                pop();
                popb();
        }
        pushb(src);
        switch(src)
        {
                case 'E': ad=0;
                        break;
                case 'T': ad=1;
                        break;
                case 'F': ad=2;
                        break;
                default: ad=-1;
                        break;
        }
        push(gotot[TOS()][ad]);
}
int main()
{
        int j,st,ic;
        char ip[20]="\0",an;
        clrscr();
        printf("Enter any String :- ");
        gets(ip);
        push(0);
        display();
        printf("\t%s\n",ip);
```

```
`
        for(j=0;ip[j]!='\0';)
        {
                st=TOS();
                an=ip[j];
                if(an>='a'&an<='z')
                        ic=0;
                else if(an=='+')
                        ic=1;
                else if(an=='*')
                        ic=2;
                else if(an=='(')
                        ic=3;
                else if(an==')')
                        ic=4;
                else if(an=='$')
                        ic=5;
                else
                {
                        error();
                        break;
                }
                if(axn[st][ic][0]==100)
                {
                        pushb(an);
                        push(axn[st][ic][1]);
                        display();
                        j++;
                        display1(ip,j);
                }
                if(axn[st][ic][0]==101)
                {
                        reduce(axn[st][ic][1]);
                        display();
                        display1(ip,j);
                }
                if(axn[st][ic][1]==102)
                {
                        printf("Given String is Accepted");
                        break;
                }
        }
        getch();
        return 0;
}
```

**OUTPUT**

```
Enter any String :-  a+b*c

0               a+b*c
0a5             +b*c
```

`

**0F3 +b*c**
**0T2 +b*c**
**0E1 +b*c**
**0E1+6 b*c**
**0E1+6b5 *c**
**0E1+6F3 *c**
**0E1+6T9 *c**
**0E1+6T9*7 c**
**0E1+6T9*7c5**

**RESULT:** Thus the LR(0) Program was executed and verified Successfully.

**EX.NO : 13**

# Construction of DAG

**AIM:** To write a C program to perform the

**ALGORITHM:**

`

```
/******************************************************************************
 *  Compilation:  javac DirectedCycle.java
 *  Execution:    java DirectedCycle < input.txt
 *  Dependencies: Digraph.java Stack.java StdOut.java In.java
 *  Data files:   http://algs4.cs.princeton.edu/42directed/tinyDG.txt
 *                http://algs4.cs.princeton.edu/42directed/tinyDAG.txt
 *
 *  Finds a directed cycle in a digraph.
 *  Runs in O(E + V) time.
 *
 *  % java DirectedCycle tinyDG.txt
 *  Cycle: 3 5 4 3
 *
 *  %  java DirectedCycle tinyDAG.txt
 *  No cycle
 *
 ******************************************************************************/
```

## SOURCE CODE:_

```
public class DirectedCycle {
    private boolean[] marked;        // marked[v] = has vertex v been
marked?
    private int[] edgeTo;            // edgeTo[v] = previous vertex on path
to v
    private boolean[] onStack;       // onStack[v] = is vertex on the
stack?
    private Stack<Integer> cycle;    // directed cycle (or null if no such
cycle)

    public DirectedCycle(Digraph G) {
        marked  = new boolean[G.V()];
        onStack = new boolean[G.V()];
        edgeTo  = new int[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) dfs(G, v);

        // check that digraph has a cycle
        assert check(G);
    }


    // check that algorithm computes either the topological order or finds
a directed cycle
    private void dfs(Digraph G, int v) {
        onStack[v] = true;
        marked[v] = true;
        for (int w : G.adj(v)) {

            // short circuit if directed cycle found
            if (cycle != null) return;

            //found new vertex, so recur
            else if (!marked[w]) {
                edgeTo[w] = v;
                dfs(G, w);
```

`

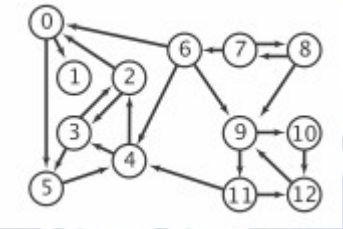
```
            }

            // trace back directed cycle
            else if (onStack[w]) {
                cycle = new Stack<Integer>();
                for (int x = v; x != w; x = edgeTo[x]) {
                    cycle.push(x);
                }
                cycle.push(w);
                cycle.push(v);
            }
        }

        onStack[v] = false;
    }

    public boolean hasCycle()        { return cycle != null;    }
    public Iterable<Integer> cycle() { return cycle;            }


    // certify that digraph is either acyclic or has a directed cycle
    private boolean check(Digraph G) {

        if (hasCycle()) {
            // verify cycle
            int first = -1, last = -1;
            for (int v : cycle()) {
                if (first == -1) first = v;
                last = v;
            }
            if (first != last) {
                System.err.printf("cycle begins with %d and ends with
%d\n", first, last);
                return false;
            }
        }


        return true;
    }

    public static void main(String[] args) {
        In in = new In(args[0]);
        Digraph G = new Digraph(in);

        DirectedCycle finder = new DirectedCycle(G);
        if (finder.hasCycle()) {
            StdOut.print("Cycle: ");
            for (int v : finder.cycle()) {
                StdOut.print(v + " ");
            }
            StdOut.println();
        }

        else {
            StdOut.println("No cycle");
        }
    }

}
```

`

**Ref:**

**Digraphs.**

A *directed graph* (or *digraph*) is a set of *vertices* and a collection of *directed edges* that each connects an ordered pair of vertices. We say that a directed edge *points from* the first vertex in the pair and *points to* the second vertex in the pair. We use the names 0 through V-1 for the vertices in a V-vertex graph.



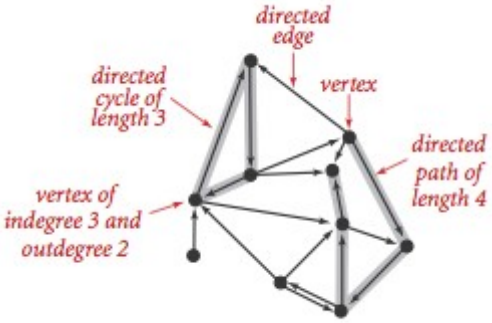

**Glossary.**

Here are some definitions that we use.

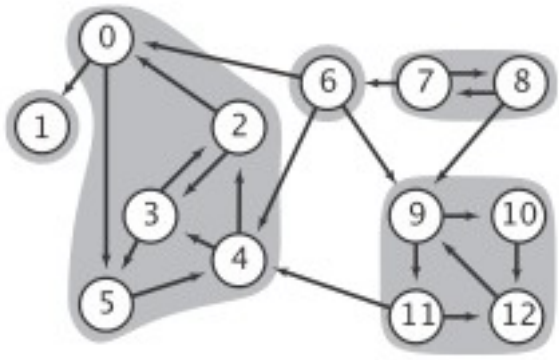
- A *self-loop* is an edge that connects a vertex to itself.
- Two edges are *parallel* if they connect the same ordered pair of vertices.
- The *outdegree* of a vertex is the number of edges pointing from it. The *indegree* of a vertex is the number of edges pointing to it.
- A *subgraph* is a subset of a digraph's edges (and associated vertices) that constitutes a digraph.
- A *directed path* in a digraph is a sequence a sequence of vertices in which there is a (directed) edge pointing from each vertex in the sequence to its successor in the sequence. A *simple path* is one with no repeated vertices.
- A *directed cycle* is a directed path (with at least one edge) whose first and last vertices are the same. A *simple cycle* is a cycle with no repeated edges or vertices (except the requisite repetition of the first and last vertices).
- The *length* of a path or a cycle is its number of edges.
- We say that a vertex `w` is *reachable from* a vertex `v` if there exists a directed path from `v` to `w`.
- We say that two vertices `v` and `w` are *strongly connected* if they are mutually reachable: there is a directed path from `v` to `w` and a directed path from `w` to `v`.
- A digraph is *strongly connected* if there is a directed path from every vertex to every other vertex.

`

- A digraph that is not strongly connected consists of a set of *strongly-connected components*, which are maximal strongly-connected subgraphs.
- A *directed acyclic graph* (or DAG) is a digraph with no directed cycles.



Anatomy of a digraph



A digraph and its strong components

**Digraph graph data type.**

We implement the following digraph API.

| public class | Digraph | |
|---|---|---|
| | Digraph(int V) | *create a V-vertex digraph with no edges* |
| | Digraph(In in) | *read a digraph from input stream in* |
| int | V() | *number of vertices* |
| int | E() | *number of edges* |
| void | addEdge(int v, int w) | *add edge v->w to this digraph* |
| Iterable<Integer> | adj(int v) | *vertices connected to v by edges pointing from v* |
| Digraph | reverse() | *reverse of this digraph* |
| String | toString() | *string representation* |

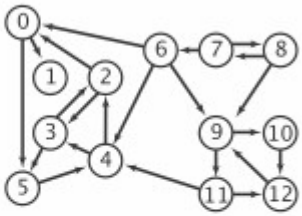
The key method `adj()` allows client code to iterate through the vertices adjacent from a given vertex.

We prepare the test data tinyDG.txt and tinyDAG.txt using the following input file format.

`

tinyDG.txt

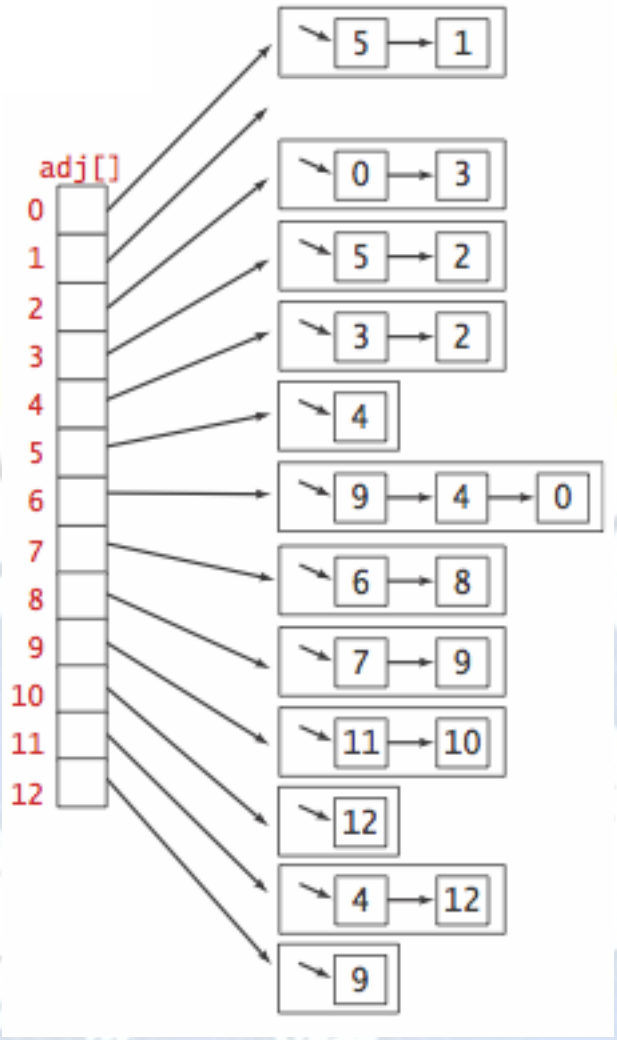
V → 13
22 ← E
4 2
2 3
3 2
6 0
0 1
2 0
11 12
12 9
9 10
9 11
8 9
10 12
11 4
4 3
3 5
7 8
8 7
5 4
0 5
6 4
6 9
7 6




**Graph representation.**

We use the *adjacency-lists representation*, where we maintain a vertex-indexed array of lists of the vertices connected by an edge to each vertex.

`




Digraph.java implements the digraph API using the adjacency-lists representation. AdjMatrixDigraph.java implements the same API using the adjacency-matrix representation.

**Reachability in digraphs.**

Depth-first search and breadth-first search are fundamentally digraph-processing algorithms.

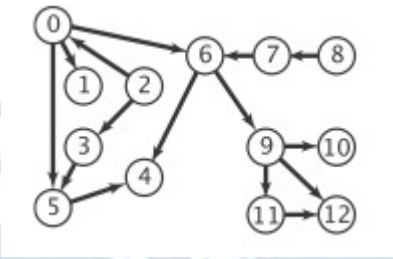
- *Single-source reachability:* Given a digraph and source `s`, is there a directed path from s to v? If so, find such a path. DirectedDFS.java uses depth-first search to solve this problem.
- *Multiple-source reachability:* Given a digraph and a *set* of source vertices, is there a directed path from *any* vertex in the set to v? DirectedDFS.java uses depth-first search to solve this problem.
- *Single-source directed paths:* given a digraph and source `s`, is there a directed path from s to v? If so, find such a path.DepthFirstDirectedPaths.java uses depth-first search to solve this problem.

`

- *Single-source shortest directed paths*: given a digraph and source `s`, is there a directed path from s to v? If so, find a shortest such path.BreadthFristDirectedPaths.java uses breadth-first search to solve this problem.

**Cycles and DAGs.**

Directed cycles are of particular importance in applications that involve processing digraphs.

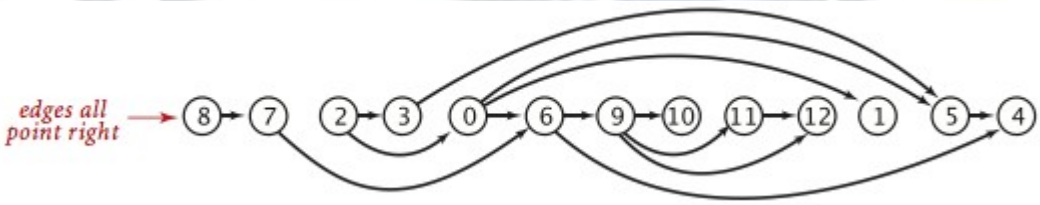



- *Directed cycle detection*: does a given digraph have a directed cycle? If so, find such a cycle. DirectedCycle.java solves this problem using depth-first search.
- *Depth-first orders*: Depth-first search search visits each vertex exactly once. Three vertex orderings are of interest in typical applications:
    - *Preorder*: Put the vertex on a queue before the recursive calls.
    - *Postorder*: Put the vertex on a queue after the recursive calls.
    - *Reverse postorder*: Put the vertex on a stack after the recursive calls.

  DepthFirstOrder.java computes these orders.

`


- *Topological sort*: given a digraph, put the vertices in order such that all its directed edges point from a vertex earlier in the order to a vertex later in the order (or report that doing so is not possible). Topological.java solves this problem using depth-first search. Remarkably, a reverse postorder in a DAG provides a topological order.




**Proposition.**

A digraph has a topological order if and only if it is a DAG.

**Proposition.**

Reverse postorder in a DAG is a topological sort.

`

**Proposition.**

With depth-first search, we can topologically sort a DAG in time proportional to V + E.

**Strong connectivity.**

Strong connectivity is an equivalence relation on the set of vertices:

- *Reflexive*: Every vertex v is strongly connected to itself.
- *Symmetric*: If v is strongly connected to w, then w is strongly connected to v.
- *Transitive*: If v is strongly connected to w and w is strongly connected to x, then v is also strongly connected to x.

Strong connectivity partitions the vertices into equivalence classes, which we refer to as *strong components* for short. We seek to implement the following API:

| public class | SCC | |
|---|---|---|
| | SCC(Digraph G) | *preprocessing constructor* |
| boolean | stronglyConnected(int v, int w) | *are v and w strongly connected?* |
| int | count() | *number of strong components* |
| int | id(int v) | *component identifier for v (between 0 and* `count()-1`*)* |

Remarkably, KosarajuSharirSCC.java implements the API with just a few lines of code added to CC.java, as follows:

- Given a digraph G, use DepthFirstOrder.java to compute the reverse postorder of its reverse, $G^R$.
- Run standard DFS on G, but consider the unmarked vertices in the order just computed instead of the standard numerical order.
- All vertices reached on a call to the recursive `dfs()` from the constructor are in a strong component (!), so identify them as in CC.

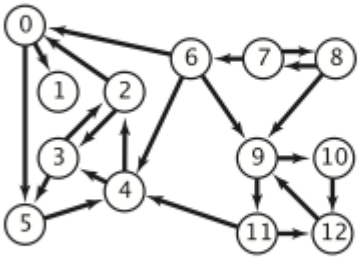
**Proposition.**

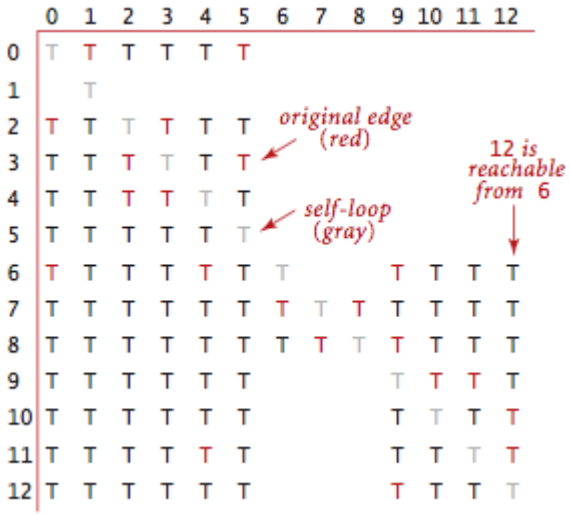
The Kosaraju-Sharir algorithm uses preprocessing time and space proportional to V + E to support constant-time strong connectivity queries in a digraph.

**Transitive closure.**

The *transitive closure* of a digraph G is another digraph with the same set of vertices, but with an edge from v to w if and only if w is reachable from v in G.

`




TransitiveClosure.java computes the transitive closure of a digraph by running depth-first search from each vertex and storing the results. This solution is ideal for small or dense digraphs, but it is not a solution for the large digraphs we might encounter in practice because the constructor uses space proportional to V^2 and time proportional to V (V + E).

**RESULT:** Thus the DAG was executed and verified Successfully.

**EX.NO : 14**

`

# Intermediate Code Generation

**AIM:** To write a C program to implementation of code generation

**ALGORITHM:**

step 1: Start.
Step 2: Enter the three address codes.
Step 3: If the code constitutes only memory operands they are moved to the register and according to the operation the corresponding assembly code is generated.
Step 4: If the code constitutes immediate operands then the code will have a # symbol proceeding the number in code.
Step 5: If the operand or three address code involve pointers then the code generated will constitute pointer register. This content may be stored to other location or vice versa.
Step 6: Appropriate functions and other relevant display statements are executed.
Step 7: Stop.

## SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10],exp[10],exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
clrscr();
while(1)
{
printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the expression with assignment operator:");
scanf("%s",exp);
l=strlen(exp);
exp2[0]='\0';
i=0;
while(exp[i]!='=')
{
```

```
`

i++;
}
strncat(exp2,exp,i);
strrev(exp);
exp1[0]='\0';
strncat(exp1,exp,l-(i+1));
strrev(exp1);
printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);
break;

case 2:
printf("\nEnter the expression with arithmetic operator:");
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';

for(i=0;i<l;i++)
{
if(exp[i]=='+'||exp[i]=='-')
{
if(exp[i+2]=='/'||exp[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(exp[i]=='/'||exp[i]=='*')
{
div();
break;
}
}
break;

case 3:
printf("Enter the expression with relational operator");
scanf("%s%s%s",&id1,&op,&id2);
if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)||(strcmp(op,">=")==0)||
(strcmp(op,"==")==0)||(strcmp(op,"!=")==0))==0)
printf("Expression is error");
else
{
printf("\n%d\tif %s%s%s goto %d",addr,id1,op,id2,addr+3);
addr++;
printf("\n%d\t T:=0",addr);
```

```
`

addr++;
printf("\n%d\t goto %d",addr,addr+2);
addr++;
printf("\n%d\t T:=1",addr);
}
break;
case 4:
exit(0);
}
}
}
void pm()
{
strrev(exp);
j=l-i-1;
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

OUTPUT :

Example Generation of Three Address Project Output Result

```
1. assignment
2. arithmetic
3. relational
4. Exit
Enter the choice:1
Enter the expression with assignment operator:
a=b
Three address code:
temp=b
a=temp

1.assignment
2.arithmetic
3.relational
```

`

4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:
a+b-c
Three address code:
temp=a+b
temp1=temp-c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:
a-b/c
Three address code:
temp=b/c
temp1=a-temp

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:
a*b-c
Three address code:
temp=a*b
temp1=temp-c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:a/b*c
Three address code:
temp=a/b
temp1=temp*c
1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:3
Enter the expression with relational operator
a
<=
b

100 if a<=b goto 103
101 T:=0

`

102 goto 104
103 T:=1

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:4

**RESULT:** Thus the Generation of Three Address was executed and verified Successfully.

**EX.NO.:** 15

`

## OPERATOR PRECEDENCE

**AIM:**
To write a C program to implement the concept of operator precedence.

**ALGORITHM**:

1. Start the program.
2. Include the required header files and start the declaration of main method.
3. Declare the required variable and define the function for pushing and poping the characters.
4. The operators are displayed in coliumn and row wise and stored it in a queue.
5. Using a switch case find the values of the operators.
6. Display the precedence of the operator and generate the code for precedence of operator for the given expression.
7. Compile and execute the program for the output.
8. Stop the program

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
char str[20],stk[20],pstk[20];
int tos=-1,flag=0,ptr=0,rm=-1,i,j;
char q[9][9]={{'>','>','<','<','<','<','>','<','>'},{'>','>','<','<','<','<','>','<','>'},{'>','>','>','>','<','<','>','<','>'},
{'>','>','>','>','<','<','>','<','>'},{'>','>','<','<','<','<','>','<','>'},{'<','<','<','<','<','<','=','<','E'},
{'>','>','>','>','>','E','>','E','>'},{'>','>','>','>','>','E','>','E','>'},{'<','<','<','<','<','<','E','<','A'},};
char c[9]={'+','-','*','/','^','a','(',')','$'};
void pushin(char a)
{
tos++;
stk[tos]=a;
}
char popout()
{
char a;
a=stk[tos];
tos--;
return(a);
}
int find(char a)
{
switch(a)
{
case'+':return 0;
case'-':return 1;
```

```
case'*':return 2;
case'/':return 3;
case'^':return 4;
case'(':return 5;
case')':return 6;
case'a':return 7;
case'$':return 8;
}
return-1;
}
void display(char a)
{
printf("\n SHIFT %c",a);
}
void display1(char a)
{
if(a!='(')
{
if(isalpha(a))
printf("\n REDUCE E--> %c",a);
else if(a==')')
printf("\n REDUCE E-->(E)");
else
printf("\n REDUCE E-->E %c E",a);
}}
int rel(char a,char b,char d)
{
if(isalpha (a))
a='a';
if(isalpha(b))
b='a';
if(q[find(a)][find(b)]==d)
return 1;
else
return 0;
}
void main()
{
clrscr();
printf("\n\n\t The productions used are:\n\t");
printf("E-->E*E/E+E/E^E/E*E/E-E\n\tE-->E/E \n\tE-->a/b/c/d/e.../z");
printf("\n\tEnteranexpressionthatterminalswith$:");
fflush(stdin);
i=-1;
while(str[i]!='$')
{
i++;
scanf("%c",&str[i]);
}
for(j=0;j<i;j++)
{
```

```
`
    if((str[j]=='(')||(str[j]==')')||(str[j+1]=='(')||(str[j+1]==')'))
    {
    }
    else if(((isalpha(str[j])==0)&&(isalpha(str[j+1])==0))||((isalpha(str[j])!=0)&&(isalpha(str[j+1])!=0)))
    {
    printf("ERROR");
    getch();
    exit(0);
    }}
    if((((isalpha(str[0]))!=0)||(str[0]=='('))&&(((isalpha(str[i-1]))!=0)||(str[i-1]==')')))
    {
    pushin('$');
    printf("\n\n\n\t+\t-\t*\t/\t^\ta\t(\t)\t$\n\n");for(i=0;i<9;i++)
    {
    printf("%c",c[i]);for(j=0;j<9;j++)
    printf("\t%c",q[i][j]);printf("\n");}getch();while(1){if(str[ptr]=='$' && stk[tos]=='$'){printf("\n\n\t
    ACCEPT!");break;}else if(rel(stk[tos],str[ptr],'<')||rel(stk[tos],str[ptr],'='))
    {display(str[ptr]);pushin(str[ptr]);ptr++;}else if(rel(stk[tos],str[ptr],'>')){do{rm+
    +;pstk[rm]=popout();display1(pstk[rm]);}while(!rel(stk[tos],pstk[rm],'<'));}
    else{printf("\n\n\t NOT ACCEPTED!!!!!!!");
    getch();
    exit(1);
    }}
    getch();
    }
    else {
    printf("ERROR");
    getch();
    }}
```

**OUTPUT:**

The productions used are:

E-->E*E/E+E/E^E/E*E/E-E

E-->E/E

E-->a/b/c/d/e.../z

Enter an expression that terminals with $: a+b*c$

| | + | - | * | / | ^ | a | ( | ) | $ |
|---|---|---|---|---|---|---|---|---|---|
| + | > | > | < | < | < | < | > | < | > |
| - | > | > | < | < | < | < | > | < | > |
| * | > | > | > | > | < | < | > | < | > |
| / | > | > | > | > | < | < | > | < | > |
| ^ | > | > | < | < | < | < | > | < | > |
| a | < | < | < | < | < | < | = | < | E |
| ( | > | > | > | > | > | E | > | E | > |
| ) | > | > | > | > | > | E | > | E | > |
| $ | < | < | < | < | < | < | E | < | A |

`

SHIFT a
REDUCE E--> a
SHIFT +
SHIFT b
REDUCE E--> b
SHIFT *
SHIFT c
REDUCE E--> c
REDUCE E-->E * E
REDUCE E-->E + E

ACCEPT!

**RESULT:**

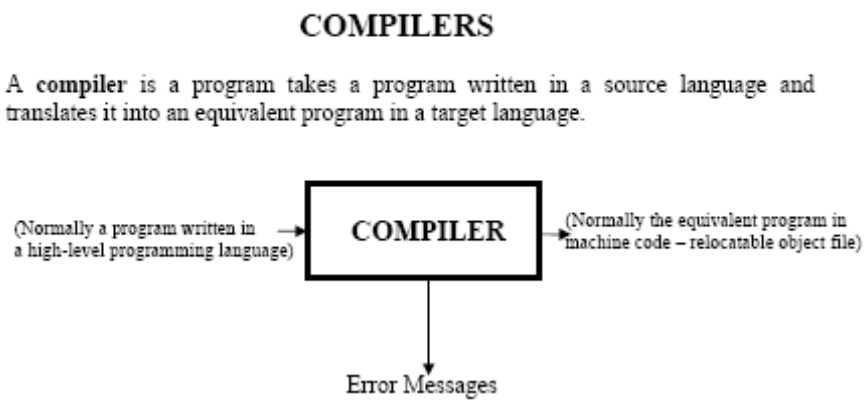
Thus the C program implementation for operator precedence is executed and verified.

**Ex. NO :16 Study Experiments**

## Principles of Compiler Design

`

**What is a compiler?**




Simply stated, a compiler is a program that reads a program written in one language-the source language-and translates it into an equivalent program in another language-the target language (see fig.1) As an important part of this translation process, the compiler reports to its user the presence of errors in the source program.

Compilers are sometimes classified as single-pass, multi-pass, load-and-go, debugging, or optimizing, depending on how they have been constructed or on what function they are supposed to perform. Despite this apparent complexity, the basic tasks that any compiler must perform are essentially the same.

**1) What are the phases of a compiler?**

**Phases of a Compiler**

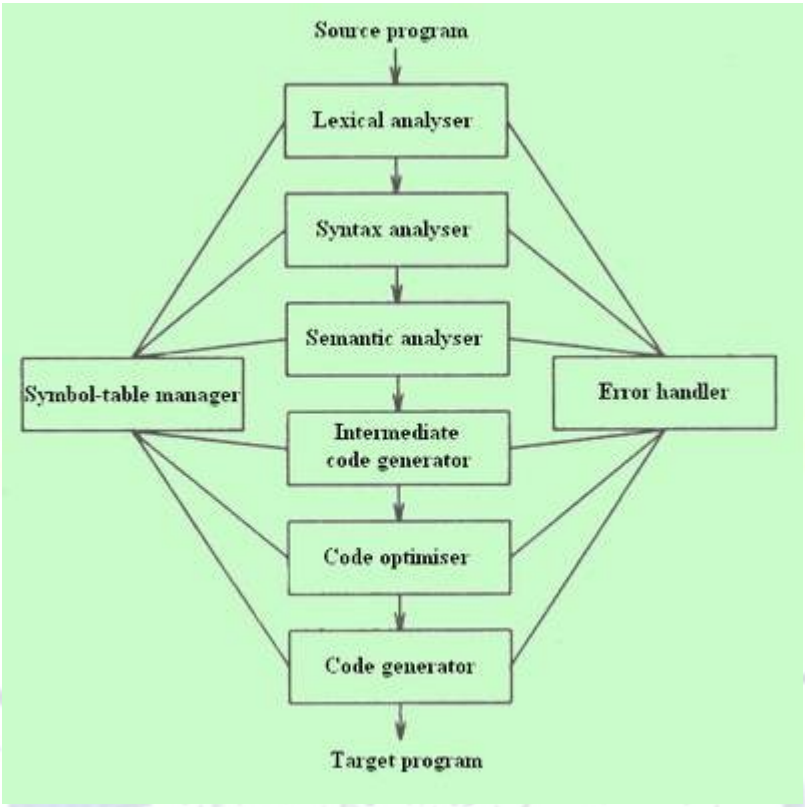
1. Lexical analysis ("scanning")
   - Reads in program, groups characters into "tokens"
2. Syntax analysis ("parsing")
   - Structures token sequence according to grammar rules of the language.
3. Semantic analysis
   - Checks semantic constraints of the language.
4. Intermediate code generation
   - Translates to "lower level" representation.
5. Program analysis and code optimization
   - Improves code quality.
6. Final code generation.

**2) Explain in detail different phases of a compiler.**

**THE DIFFERENT PHASES OF A COMPILER**

Conceptually, a compiler operates in *phases*, each of which transforms the source program from one representation to another.

`




The first three phases, forms the bulk of the analysis portion of a compiler. Symbol table management and error handling, are shown interacting with the six phases.

**Symbol table management**

An essential function of a compiler is to record the identifiers used in the source program and collect information about various attributes of each identifier. A *symbol table* is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly. When an identifier in the source program is detected by the lex analyzer, the identifier is entered into the symbol table.

**Error Detection and Reporting**

`

Each phase can encounter errors. A compiler that stops when it finds the first error is not as helpful as it could be.

The syntax and semantic analysis phases usually handle a large fraction of the errors detectable by the compiler. The lexical phase can detect errors where the characters remaining in the input do not form any token of the language. Errors when the token stream violates the syntax of the language are determined by the syntax analysis phase. During semantic analysis the compiler tries to detect constructs that have the right syntactic structure but no meaning to the operation involved.

**The Analysis phases**

As translation progresses, the compiler's internal representation of the source program changes. Consider the statement,

position := initial + rate * 10

The lexical analysis phase reads the characters in the source pgm and groups them into a stream of tokens in which each token represents a logically cohesive sequence of characters, such as an identifier, a keyword etc. The character sequence forming a token is called the *lexeme* for the token. Certain tokens will be augmented by a 'lexical value'. For example, for any identifier the lex analyzer generates not only the token id but also enter s the lexeme into the symbol table, if it is not already present there. The lexical value associated this occurrence of id points to the symbol table entry for this lexeme. The representation of the statement given above after the lexical analysis would be:

id1: = id2 + id3 * 10

Syntax analysis imposes a hierarchical structure on the token stream, which is shown by syntax trees (fig 3).

**Intermediate Code Generation**

After syntax and semantic analysis, some compilers generate an explicit intermediate representation of the source program. This intermediate representation can have a variety of forms.

`

In three-address code, the source pgm might look like this,

temp1: = inttoreal (10)

temp2: = id3 * temp1

temp3: = id2 + temp2

id1: = temp3

**Code Optimisation**

The code optimization phase attempts to improve the intermediate code, so that faster running machine codes will result. Some optimizations are trivial. There is a great variation in the amount of code optimization different compilers perform. In those that do the most, called 'optimising compilers', a significant fraction of the time of the compiler is spent on this phase.

**Code Generation**

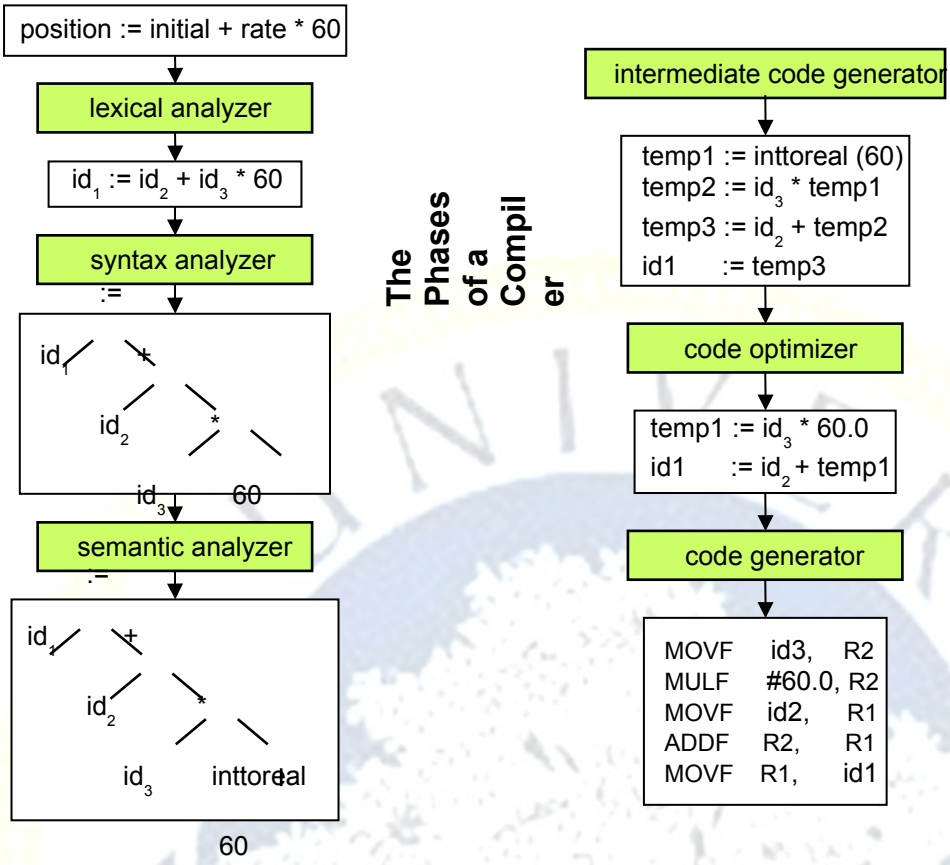
The final phase of the compiler is the generation of target code, consisting normally of relocatable machine code or assembly code. Memory locations are selected for each of the variables used by the program. Then, intermediate instructions are each translated into a sequence of machine instructions that perform the same task. A crucial aspect is the assignment of variables to registers.

**3) What is grouping of phases?**

**Grouping of Phases**

- *Front end* : machine independent phases
  - Lexical analysis
  - Syntax analysis
  - Semantic analysis
  - Intermediate code generation
  - Some code optimization
- *Back end* : machine dependent phases
  - Final code generation
  - Machine-dependent optimizations

**4) Explain with diagram how a statement is compiled .**




**RESULT:** Thus the above Study Experiments is studied .