Mahatma Gandhi Mission's

College of Engineering & Technology

A-9, Sector-62, NOIDA

Subject

Name: _____ Roll No.: _____

Year: Ist ☐ IInd ☐ IIIrd ☐ IVth ☐

Semester: _____ Branch : _____

Academic Year: _____

INDEX

Ex. No.	Experiment Name	Date	Page No.	Signature
1	Implementation of Lexical Analyzer for If statement			
2	Implementation of Lexical Analyzer for Arithmetic Expression			
3	Construction of NFA from Regualr Expression			
4	Construction of DFA from NFA			
5	Implementation of Shift Reducing Parser Algorithm			
6	Implementation of Operator Precedence Parser			
7	Implementation of Code Optimization Technique			
8	Implementation of Code Generator			

Experiment No: 1

AIM: Implementation of Lexical Analyzer for 'if' Statement

PRORGAM

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<string.h>
char vars[100][100];
int vcnt;
char input[1000],c;
char token[50],tlen;
int state=0,pos=0,i=0,id;
char*getAddress(char str[])
{
for(i=0;i<vcnt;i++)
if(strcmp(str,vars[i])==0)
return vars[i];
strcpy(vars[vcnt],str);
return vars[vcnt++];
}
intisrelop(char c)
{
if(c=='>'||c=='<'||c=='|'||c=='=')
return 1;
else
return 0;
}
int main(void)
{
clrscr();
printf("Enter the Input String:");
gets(input);
do
{
c=input[pos];
putchar(c);
switch(state)
{
case 0:
if(c=='i')
state=1;
break;
case 1:
if(c=='f'
)
{
printf('\t<1,1>\n");
state =2;
}
```

```
break;
case 2:
if(isspace(c))
printf("\b");
if(isalpha(c))
{
token[0]=c;
tlen=1;
state=3;
}
if(isdigit(c))
state=4;
if(isrelop(c))
state=5;
if(c==';')printf("\t<4,4>\n");
if(c=='(')printf("\t<5,0>\n");
if(c==')')printf("\t<5,1>\n");
if(c=='{') printf("\t<6,1>\n");
if(c=='}') printf("\t<6,2>\n");
break;
case 3:
 if(!isalnum(c))
{
token[tlen]='\0';
printf("\b\t<2,%p>\n",getAddress(token));
state=2;
pos--;
}
else
token[tlen++]=c;
break;
case 4:
if(!isdigit(c))
{
printf("\b\t<3,%p>\n",&input[pos]);
state=2;
pos--;
}
break;
case 5:
id=input[pos-1];
if(c=='=')
printf("\t<%d,%d>\n",id*10,id*10);
else
{
printf("\b\t<%d,%d>\n",id,id);
pos--;
}
state=2;
break;
}
pos++;
}
```

```
while(c!=0);
getch();
return 0;
}
```

Sample Input & Output:

```
Enter the file name : input.c
                LEXICAL ANALYSIS

line : 1
                #       :       preprocessor
                include :       keyword
                "       :       doublequote
                stdio.h :       keyword
                "       :       doublequote

line : 2
                #       :       preprocessor
                include :       keyword
                "       :       doublequote
                conio.h :       keyword
                "       :       doublequote

line : 3
                void    :       keyword
                main    :       keyword
                (       :       openpara
                )       :       closepara

line : 4
                {       :       openbrace

line : 5
                int     :       keyword
                a       :       identifier
                =       :       equal
                10      :       constant
                ,       :       identifier
                b       :       identifier
                ,       :       identifier
                c       :       identifier
                ;       :       semicolon

line : 6
                a       :       identifier
                =       :       equal
                b       :       identifier
                *       :       star
                c       :       identifier
                ;       :       semicolon

line : 7
                getch   :       identifier
                (       :       openpara
                )       :       closepara
                ;       :       semicolon

line : 8
                }       :       closebrace

line : 9
                $       :       identifier
```

Result:

The above C program was successfully executed and verified.

Experiment No: 2

AIM: Implementation of Lexical Analyzer for Arithmetic Expression

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<string.h>
char vars[100][100];
int vcnt;
char input[1000],c;
char token[50],tlen;
int state=0,pos=0,i=0,id;
char *getAddress(char str[])
{
for(i=0;i<vcnt;i++)
if(strcmp(str,vars[i])==0)
return vars[i];
strcpy(vars[vcnt],str);
return vars[vcnt++];
}
intisrelop(char c)
{
if(c=='+'||c=='-'||c=='*'||c=='/'||c=='%'||c=='^')
return 1;
else
return 0;
}
int main(void)
{
clrscr();
printf("Enter the Input String:");
gets(input);
do
{
c=input[pos];
putchar(c);
switch(state)
{
case 0:
if(isspace(c))
printf("\b");
if(isalpha(c))
{
token[0]=c;
tlen=1;
state=1;
}
if(isdigit(c))
state=2;
```

```
if(isrelop(c))
state=3;
if(c==';')
printf("\t<3,3>\n");
if(c=='=')
printf("\t<4,4>\n");
break;
case 1:
if(!isalnum(c))
{
token[tlen]='\o';
printf("\b\t<1,%p>\n",getAddress(token));
state=0;
pos--;
}
else
token[tlen++]=c;
break;
case 2:
if(!isdigit(c))
{
printf("\b\t<2,%p>\n",&input[pos]);
state=0;
pos--;
}
break;
case 3:
id=input[pos-1];
if(c=='=')
printf("\t<%d,%d>\n",id*10,id*10);
else
{
printf("\b\t<%d,%d>\n",id,id);
pos--;
}
state=0;
break;
}
pos++;
}
while(c!=0);
getch();
return 0;
```

Sample Input & Output:

Enter the Input String: a=a*2+b/c; a

=	<4,4>
a	<1,08CE>
*	<42,42>
2	<2,04E9>
+	<43,43>
b	<1,0932>
/	<47,47>
c	<1,0996>
;	<3,3>

Result:

The above C program was successfully executed and verified.

Experiment No: 3

AIM: Construction of NFA from Regular Expression

Program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
#include<graphics.h>
#include<math.h>
#include<process.h>
int
minx=1000,miny=0;
void star(int *x1,int *y1,int *x2,int *y2)
{
char pr[10];
ellipse(*x1+(*x2-*x1)/2,*y2-10,0,180,(*x2-
*x1)/2,70); outtextxy(*x1-2,*y2-17,"v");
line(*x2+10,*y2,*x2+30,*y2);
outtextxy(*x1-15,*y1-
3,">"); circle(*x1-
40,*y1,10); circle(*x1-
80,*y1,10); line(*x1-
30,*y2,*x1-10,*y2);
outtextxy(*x2+25,*y2-
3,">"); sprintf(pr,"%c",238);
outtextxy(*x2+15,*y2-9,pr);
outtextxy(*x1-25,*y1-9,pr);
outtextxy((*x2-*x1)/2+*x1,*y1-30,pr);


outtextxy((*x2-*x1)/2+*x1,*y1+30,pr);
ellipse(*x1+(*x2-*x1)/2,*y2+10,180,360,(*x2-
*x1)/2+40,70); outtextxy(*x2+37,*y2+14,"^");
if(*x1-40<minx)minx=*x1-40;
miny=*y1;
}
void star1(int *x1,int *y1,int *x2,int *y2)
{
char pr[10];
ellipse(*x1+(*x2-*x1)/2+15,*y2-10,0,180,(*x2-
*x1)/2+15,70); outtextxy(*x1-2,*y2-17,"v");
line(*x2+40,*y2,*x2+60,*y2
); outtextxy(*x1-15,*y1-
3,">"); circle(*x1-
40,*y1,10); line(*x1-
30,*y2,*x1-10,*y2);
outtextxy(*x2+25,*y2-
3,">"); sprintf(pr,"%c",238);
```

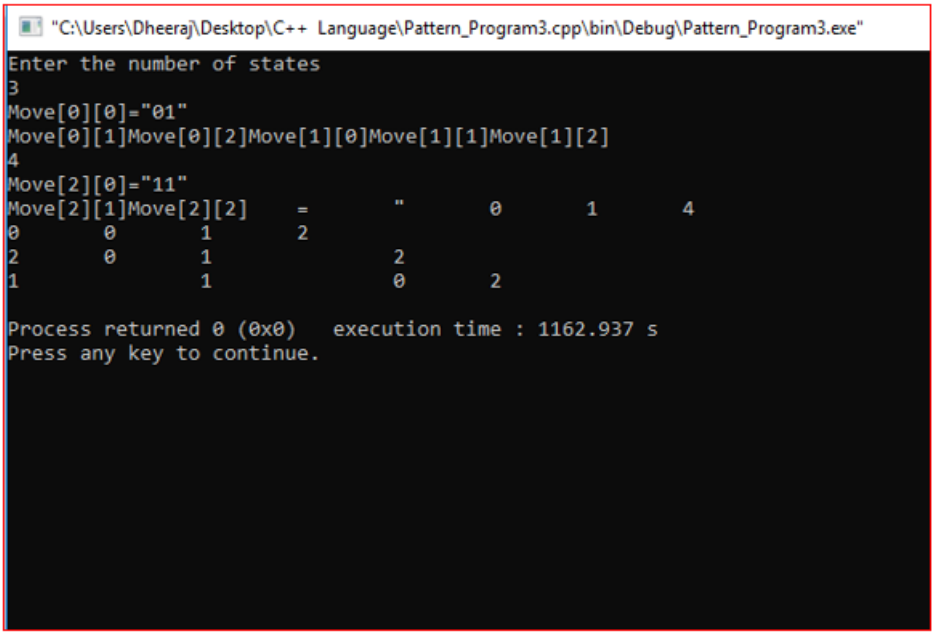
```
outtextxy(*x2+15,*y2-9,pr);
outtextxy(*x1-25,*y1-9,pr);
outtextxy((*x2-*x1)/2+*x1,*y1-
30,pr); outtextxy((*x2-
*x1)/2+*x1,*y1+30,pr);
ellipse(*x1+(*x2-*x1)/2+15,*y2+10,180,360,(*x2-*x1)/2+50,70);
outtextxy(*x2+62,*y2+13,"^");
if(*x1-40<minx)minx=*x1-40;
miny=*y1;
}
void basis(int *x1,int *y1,char x)
{
char pr[5];
circle(*x1,*y1,10);
line(*x1+30,*y1,*x1+10,*y1
); sprintf(pr,"%c",x);
outtextxy(*x1+20,*y1-10,pr);
outtextxy(*x1+23,*y1-
3,">");
circle(*x1+40,*y1,10);
if(*x1<minx)minx=*x1;
miny=*y1;
}
void slash(int *x1,int *y1,int *x2,int *y2,int *x3,int *y3,int *x4,int *y4)
{
char
pr[10]; int
c1,c2;
c1=*x1;
if(*x3>c1)c1=*x3;
c2=*x2;
if(*x4>c2)c2=*x4;
line(*x1-10,*y1,c1-40,(*y3-*y1)/2+*y1-10);
endx[pos-1]=endx[pos-1]+40;
x1=x1+40;
}
if(str[i]=='(')
{
int s;
s=i;
while(str[s]!=')')s++;
if((str[s+1]=='*')&&(pos!=0))x1=x1+40;
op[par]=pos;
par++;
}
if(str[i]==')')
{
cx2=endx[pos-
1];
cy2=endy[pos-
1]; l=op[par-1];
cx1=stx[1];
cx2=sty[1];
```

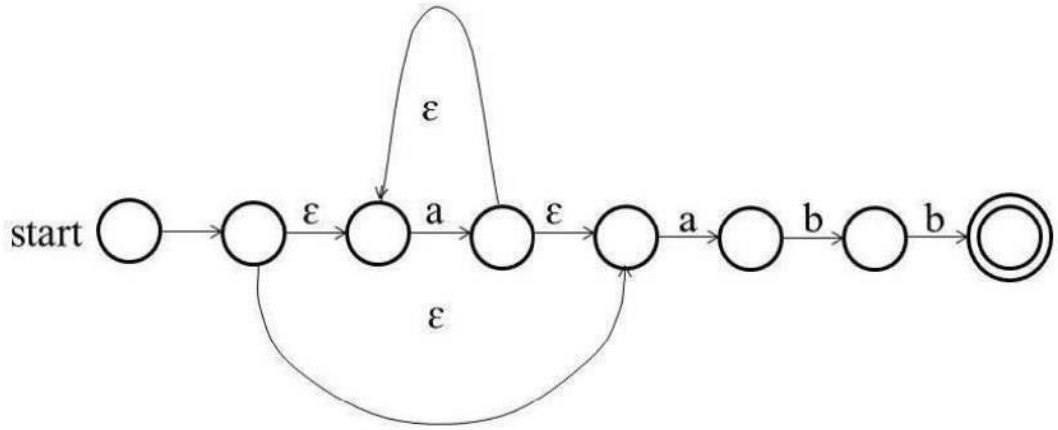
```
par--;
if(str[i+1]=='*')
{
i++;
star1(&cx1,&cy1,&cx2,&cy2
); cx1=cx1-40;
cx2=cx2+40;
stx[1]=stx[1]-
40;
endx[pos-1]=endx[pos-1]+40;
x1=x1+40;
}
if(d==1)
{
slash(&cx3,&cy3,&cx4,&cy4,&cx1,&cy1,&cx2,&cy2);
if(cx4>cx2)x1=cx4+40;
else x1=cx2+40;
y1=(y1-
cy4)/2.0+cy4; d=0;
}
}
if(str[i]=='/')
{
cx2=endx[pos-
1];
cy2=endy[pos-
1]; x1=200;
y1=y1+100;
if(str[i+1]=='(')
{
d=1;
cx3=cx1
;
cy3=cy1
;
cx4=cx2
;
cy4=cy2
;
}
if(isalpha(str[i+1]))
{
i++;
basis(&x1,&y1,str[i]);
stx[pos]=x1;
endx[pos]=x1+40;
sty[pos]=y1;
endy[pos]=y1;
if(str[i+1]=='*')
{
i++;
star(&stx[pos],&sty[pos],&endx[pos],&endy[pos]
); stx[pos]=stx[pos]-40;
```

```
endx[pos]=endx[pos]+40;
}
slash(&cx1,&cy1,&cx2,&cy2,&stx[pos],&sty[pos],&endx[pos],&endy[pos]);
if(cx2>endx[pos])x1=cx2+40;
else
x1=endx[pos]+40;
y1=(y1-cy2)/2.0+cy2;
cx1=cx1-40;
cy1=(sty[pos]-cy1)/2.0+cy1;
cx2=cx2+40;
cy2=(endy[pos]-
cy2)/2.0+cy2; l=op[par-1];
stx[1]=cx1;
sty[1]=cy1;
endx[pos]=cx2;
endy[pos]=cy2;
pos++;
}
}
i++;
}
circle(x1,y1,13);
line(minx-30,miny,minx-10,miny);
outtextxy(minx-100,miny-
10,"start"); outtextxy(minx-
15,miny-3,">");

getch();
closegraph()
;
}
```

Sample Input & Output:





Result:

The above C program was successfully executed and verified.

Experiment No: 4

AIM : Construction of DFA from NFA

Program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<process.h>
typedef struct
{
int num[10],top;
}
stack;
stack
s;
int mark[16][31],e_close[16][31],n,st=0;
char data[15][15];
void push(int a)
{
s.num[s.top]=a;
s.top=s.top+1;
}
int pop()
{
int a;
if(s.top==0)
return(-1);
s.top=s.top-1;
a=s.num[s.top]
;

return(a);
}
void epi_close(int s1,int s2,int c)
{
int i,k,f;
for(i=1;i<=n;i++
)
{
if(data[s2][i]=='e')
{
f=0;
for(k=1;k<=c;k++)
if(e_close[s1][k]==i)
f=1;
if(f==0)
{
c++;
```

```
e_close[s1][c]=i;
push(i);
}
}
}
while(s.top!=0) epi_close(s1,pop(),c);
}
int move(int sta,char c)
{
int i;
for(i=1;i<=n;i++
)
{
if(data[sta][i]==c
) return(i);
}
return(0);
}
void e_union(int m,int n)
{
int i=0,j,t;
for(j=1;mark[m][i]!=-
1;j++)
{
while((mark[m][i]!=e_close[n][j])&&(mark[m][i]!=-
1)) i++;
if(mark[m][i]==-1)mark[m][i]=e_close[n][j];
}
}
void main()
{
int i,j,k,Lo,m,p,q,t,f;
clrscr();

printf("\n enter the NFA state table entries:");
scanf("%d",&n);
printf("\n");
for(i=0;i<=n;i++
) printf("%d",i);
printf("\n");
for(i=0;i<=n;i++
) printf(" --- ");
printf("\n");
for(i=1;i<=n;i++
)
{
printf("%d|",i);
fflush(stdin);
for(j=1;j<=n;j++)
scanf("%c",&data[i][j]);
}
for(i=1;i<=15;i++)
for(j=1;j<=30;j++)
```

```
{
e_close[i][j]=-1;
mark[i][j]=-1;
}
for(i=1;i<=n;i++)
{
e_close[i][1]=i
; s.top=0;
epi_close(i,i,1)
;
}
for(i=1;i<=n;i++)
{
for(j=1;e_close[i][j]!=-1;j++)
for(k=2;e_close[i][k]!=-1;k++)
if(e_close[i][k-1]>e_close[i][k])
{
t=e_close[i][k-1];
e_close[i][k-
1]=e_close[i][k];
e_close[i][k]=t;
}
}
printf("\n the epsilon closures are:");
for(i=1;i<=n;i++)
{
printf("\n E(%d)={",i);
for(j=1;e_close[i][j]!=-1;j++)
printf("%d",e_close[i][j])
; printf("}");
}

j=1;
while(e_close[1][j]!=-1)
{
mark[1][j]=e_close[1][j];
j++;
}
st=1;
printf("\n DFA Table is:");
printf("\n              a              b      ");
printf("\n.............................................. ");

for(i=1;i<=st;i++)
{
printf("\n{");
for(j=1;mark[i][j]!=-
1;j++)
printf("%d",mark[i][j])
; printf("}");
while(j<7)
{
printf("
```

```
"); j++;
}
for(Lo=1;Lo<=2;Lo++
)
{
for(j=1;mark[i][j]!=-1;j++)
{
if(Lo==1)
t=move(mark[i][j],'a')
; if(Lo==2)
t=move(mark[i][j],'b')
; if(t!=0)
e_union(st+1,t);
}
for(p=1;mark[st+1][p]!=-1;p++)
for(q=2;mark[st+1][q]!=-1;q++)
{
if(mark[st+1][q-1]>mark[st+1][q])
{
t=mark[st+1][q];
mark[st+1][q]=mark[st+1][q-
1]; mark[st+1][q-1]=t;
}
}
f=1;
for(p=1;p<=st;p++)
{
j=1;

while((mark[st+1][j]==mark[p][j])&&(mark[st+1][j]!=-
1)) j++;
if(mark[st+1][j]==-1 && mark[p][j]==-
1) f=0;
}
if(mark[st+1][1]==-1)
f=0;
printf("\t{");
for(j=1;mark[st+1][j]!=-
1;j++)
{
printf("%d",mark[st+1][j]);
}
printf("}\t");
if(Lo==1)
printf(" ");
if(f==1)
st++;
if(f==0)
{
for(p=1;p<=30;p++
) mark[st+1][p]=-1;
}
}
```

```
}
getch();
}
```

Sample Input & Output:

```
"C:\Users\Dheeraj\Desktop\C++ Language\Pattern_Program3.cpp\bin\Debug\Pattern_Program3.exe"
Enter the number of states
3
Move[0][0]="01"
Move[0][1]Move[0][2]Move[1][0]Move[1][1]Move[1][2]
4
Move[2][0]="11"
Move[2][1]Move[2][2]    =       "       0       1       4
0       0       1       2
2       0       1               2
1               1               0       2

Process returned 0 (0x0)   execution time : 1162.937 s
Press any key to continue.
```

Result:

The above C program was successfully executed and verified.

EXPERIMENT NO : 5

AIM : Implementation of Shift Reduce Parsing Algorithm

To write a C program to implement the shift-reduce parsing algorithm.

Program:

```
#include<conio.h
>
#include<stdio.h>
#include<stdlib.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\n\t Shift Reduce Parser\
n"); printf(" n t***** ******
******\"); printf(" n \Grammar n n");
printf("E->E+E\ nE->E/E\
n"); printf("E->\E*E nE-
>a/b");
printf("\n Enter the Input Symbol\ t");
gets(ip_sym);
printf("\n\n\t Stack Implementation Table");
printf("\n Stack\ t\ t Input Symbol\ t\ t Action");
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();
getch();
}

void check()
{
int flag=0;
temp2[0]=stack[st_ptr]
; temp[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->a",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$%s\t\t\t%s$\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",ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n %s\t\t\t %s\t\t\Reject",stack,ip_sym);
```

```
}
return;
}
```

Sample Input & Output:

```
                    SHI  "RE DUC E PQRS ER

GRP[-][-]E R


E->E; E
E->E' E
E->a/b
enter the input symbol:          a-b

        sta:t implementa-ion table
                input symbol            ac -ion


$               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                                      ACCEP'
```

```
                    SH?F   REDUCE PARSER

GRP[-][-]E R

E->E-E
E->E; E
E->E' E
E->a/b
enter ':he input symbol:          a/b

        sta:  implementation table
                input symbol            action


$               a/b$
$a               /b$                    shlf:: a
$E               /b$                    E->a
$E¿               b$                    shift /
$E/b               $                    shift b
$E¿ E              $                    E->b
Process netunned 1 f0x1,    execution time : 7.5s4 s
Pness an; ke; to continue.
```

Result:

The above C program was successfully executed and verified.

Experiment No: 6

Implementation of Operator Precedence Parser

Aim:

To write a C program to implement Operator Precedence Parser

Program:

```
#include<stdio.h>
#include<conio.>
#include<string.>
#include<ctype.>
char q[9][9]={
{'>','>','<','<','<','<','>','<','>' },
{'>','>','<','<','<','<','>','<','>' },
{'>','>','>','>','<','<','>','<','>' },
{'>','>','>','>','<','<','>','<','>' },
{'>','>','<','<','<','<','>','<','>' },

{'<','<','<','<','<','<','=','<','E' },
{'>','>','>','>','>','E','>','E','>' },
{'>','>','>','>','>','E','>','E','>' },
{'<','<','<','<','<','<','E','<','A' }
};
char
s[30],st[30],qs[30]; int
top=-1,r=-1,p=0; void
push(char a)
{
top++;
st[top]=a
;
}
char pop()
{
char a;
a=st[top]
; top--;
return a;
}
int find(char a)
{
switch(a)
{
case '+':return 0;
case '-':return 1;
case '*':return 2;
case '/':return 3;
```

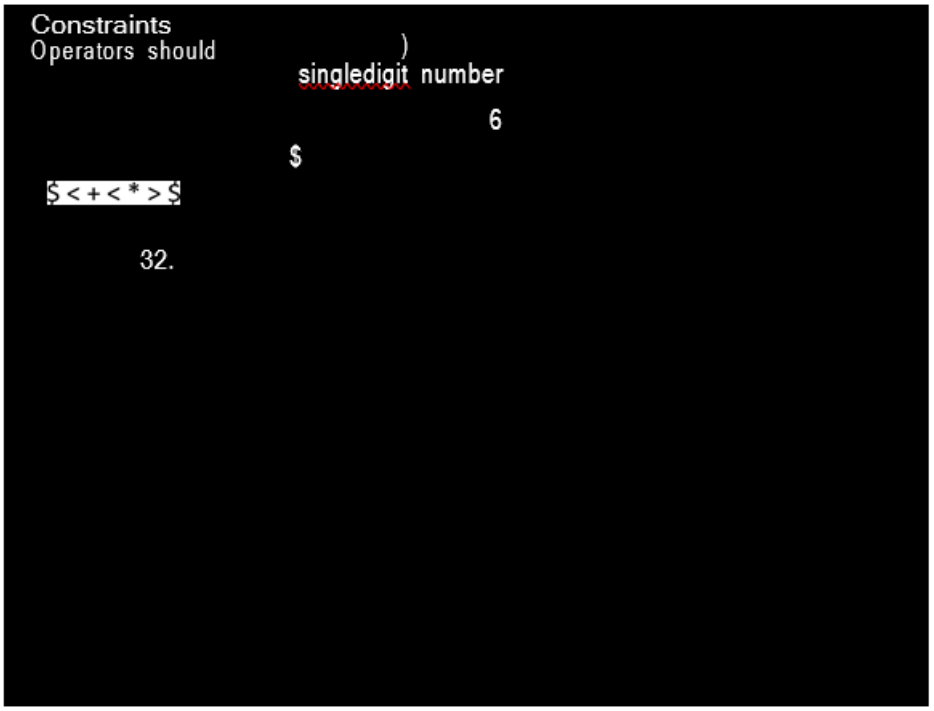
```
case '^':return 4;
case '(':return 5;
case ')':return 6;
case 'a':return 7;
case '$':return 8;
default :return -1;
}
}
void display(char a)
{
printf("\n Shift %c",a);
}
void display1(char a)
{
if(isalpha(a))
printf("\n Reduce E->%c",a);
else if((a=='+')||(a=='-')||(a=='*')||(a=='/')||(a=='^'))
printf("\n Reduce E->E%cE",a);
else if(a==')')

printf("\n Reduce E->(E)");


}
intrel(char a,char b,char d)
{
if(isalpha(a)!=0
) a='a';
if(isalpha(b)!=0
) b='a';
if(q[find(a)][find(b)]==d)
return 1;
else
return 0;
}
void main()
{
char s[100];
int i=-1;
clrscr();
printf("\n\t Operator Preceding Parser\ n");
printf("\n Enter the Arithmetic Expression End with $..");
gets(s);
push('$')
;
while(i)
{
if((s[p]=='$')&&(st[top]=='$'))
{
printf("\n\nAccepted");
break;
}
else if(rel(st[top],s[p],'<')||rel(st[top],s[p],'='))
{
display(s[p]);
push(s[p])
```

```
; p++;
}
else if(rel(st[top],s[p],'>'))
{
do
{
r++;
qs[r]=pop();
display1(qs[r]);
}
while(!rel(st[top],qs[r],'<'));
}
}
getch();
}
```

Sample Input & Output:



Result:

The above C program was successfully executed and verified.

Experiment No: 7

AIM: Implementation of Code Optimization Techniques

To write a C program to implement Code Optimization Techniques.

Program:

```
#include<stdio.h>
#include<conio.>
#include<string.>
struct op
{
char l;
char
r[20];
}
op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q;
char *p,*l;
char
temp,t;
char *tem;
clrscr();
printf("Enter the Number of Values:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("left: ");
op[i].l=getche();
printf("\tright: ");
scanf("%s",op[i].r);
}
printf("Intermediate Code\n") ;
for(i=0;i<n;i++)
{
printf("%c=",op[i].l);
printf("%s\n",op[i].r);
}
```

```
for(i=0;i<n-1;i++)
{
temp=op[i].l;
for(j=0;j<n;j++
)
{
p=strchr(op[j].r,temp);
if(p)
{
pr[z].l=op[i].l;
strcpy(pr[z].r,op[i].r);
z++;
}
}
}
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r);
z++;
printf("\nAfter Dead Code Elimination\n");
for(k=0;k<z;k++)
{
printf("%c\t=",pr[k].l);
printf("%s\n",pr[k].r);
}
for(m=0;m<z;m++)
{
tem=pr[m].r;
for(j=m+1;j<z;j++)
{
p=strstr(tem,pr[j].r);
if(p)
{
t=pr[j].l;
pr[j].l=pr[m].l;
for(i=0;i<z;i++
)
{
l=strchr(pr[i].r,t)
; if(l)
{
a=l-pr[i].r;
printf("pos: %d",a);
pr[i].r[a]=pr[m].l;
}
}
}
}
}
```

```
printf("Eliminate Common Expression\n");
for(i=0;i<z;i++)
{
printf("%c\t=",pr[i].l);
printf("%s\n",pr[i].r);
}
for(i=0;i<z;i++)
{
for(j=i+1;j<z;j++)
{
q=strcmp(pr[i].r,pr[j].r);
if((pr[i].l==pr[j].l)&&!q)
{
pr[i].l='\0';
strcpy(pr[i].r,'\0');
}
}
}
printf("Optimized Code\n");
for(i=0;i<z;i++)
{
if(pr[i].l!='\0')
{
printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
}
}
getch();
}
```

Sample Input & Output:

Result:

The above C program was successfully executed and verified.

Experiment No: 8

AIM :Implementation of Code Generator

To write a C program to implement Simple Code Generator.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<graphics.h>
typedef struct
{
char
var[10]; int
alive;
}
regist;
regist preg[10];
void substring(char exp[],int st,int end)
{
int i,j=0;
char dup[10]="";
for(i=st;i<end;i++
)
dup[j++]=exp[i];
dup[j]='0';

strcpy(exp,dup);
}
int getregister(char var[])
{
int i;
for(i=0;i<10;i++
)
{
if(preg[i].alive==0)
```

```
{
strcpy(preg[i].var,var);
break;
}
}
return(i);
}
void getvar(char exp[],char v[])
{
int i,j=0;
char var[10]="";
for(i=0;exp[i]!=\
0';i++)
if(isalpha(exp[i]))
var[j++]=exp[i];
else
break;
strcpy(v,var)
;
}
void main()
{
char
basic[10][10],var[10][10],fstr[10],op; int
i,j,k,reg,vc,flag=0;
clrscr();
printf("\nEnter the Three Address Code\: n");
for(i=0;;i++)
{
gets(basic[i]);
if(strcmp(basic[i],"exit")==0)
break;
}
printf("\nThe Equivalent Assembly Code is\:
n"); for(j=0;j<i;j++)
{
getvar(basic[j],var[vc++]
); strcpy(fstr,var[vc-1]);
substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));
getvar(basic[j],var[vc++]);
reg=getregister(var[vc-1]);
if(preg[reg].alive==0)
{
printf("\nMov R%d,%s",reg,var[vc-1]);
preg[reg].alive=1;
}
op=basic[j][strlen(var[vc-1])];
substring(basic[j],strlen(var[vc-
1])+1,strlen(basic[j])); getvar(basic[j],var[vc++]);
switch(op)
{
case '+': printf("\nAdd"); break;
case '-': printf("\nSub"); break;
```

```
                    case '*': printf(" nMul"); break;
                    case '/': printf(" nDiv"); break;
                    }

Flag=1;
for(k=0;k<=reg;k++)
                    {
                    if(strcmp(preg[k].var,var[vc-1])==0)
                    {
                    prin
                    tf("
                    R%
                    d,
                    R%
                    d",k
                    ,reg
                    );
                    preg
                    [k].
                    aliv
                    e=0;
                    flag=0
                    ;
                    break;
                    }
                    }
                    if(flag)
                    {
                    printf("
                    %s,R%d",
                    var[vc-
                    1],reg);
                    printf("
                    nMov
                    %s,R%d",fstr,reg);
                    }
                    strcpy(pr
                    eg[reg].v
                    ar,var[vc
                    -3]);
                    getch();
                    }
                    }
```

Sample Input & Output:

```
          INTERMEDIATE CODE GENERATION

Enter the Expression :w:=a*b+c/d-e/f+g*h
The intermediate code:          Expression
        Z := c/d                w:=a*b+Z-e/f+g*h
        Y := e/f                w:=a*b+Z-Y+g*h
        X := a*b                w:=X+Z-Y+g*h
        W := g*h                w:=X+Z-Y+W
        V := X+Z                w:=V-Y+W
        U := Y+W                w:=V-U
        T := V-U                w:=T
        w := T
```

Result:

The above C program was successfully executed and verified.