FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
B.E.(CSE) VI SEMESTER

CSCP 607 – COMPILER DESIGN LAB

Name : ……
Reg. No. : ……

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E.(CSE) VI SEMESTER

CSCP 607 – COMPILER DESIGN LAB

Certified that this Bonafide Record of work done by Mr. /Ms.....
Reg. No..... of B.E. (CSE) in the ***CSCP 607 – COMPILER DESIGN LAB*** *during the year 2020 – 2021.*

Staff-in- Charge

Internal Examiner

Annamalai nagar
Date:

External Examiner

Table of Contents

S. No	Date	Program	Page No	Signature
1		Construction of NFA from regular expression		
2		Construction of DFA from NFA		
3		Implementation of recursive descent parser		
4		Implementation of code optimization techniques		
5		Implementation of code generator		

Ex. No: 1

Date:

Construction of NFA from Regular Expression

Aim:

To write a C program to construct a Non Deterministic Finite Automata (NFA) from Regular Expression.

Algorithm:

1. Start the Program.
2. Enter the regular expression R over alphabet E.
3. Decompose the regular expression R into its primitive components
4. For each component construct finite automata.
5. To construct components for the basic regular expression way that corresponding to that way compound regular expression.
6. Stop the Program.

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);
```

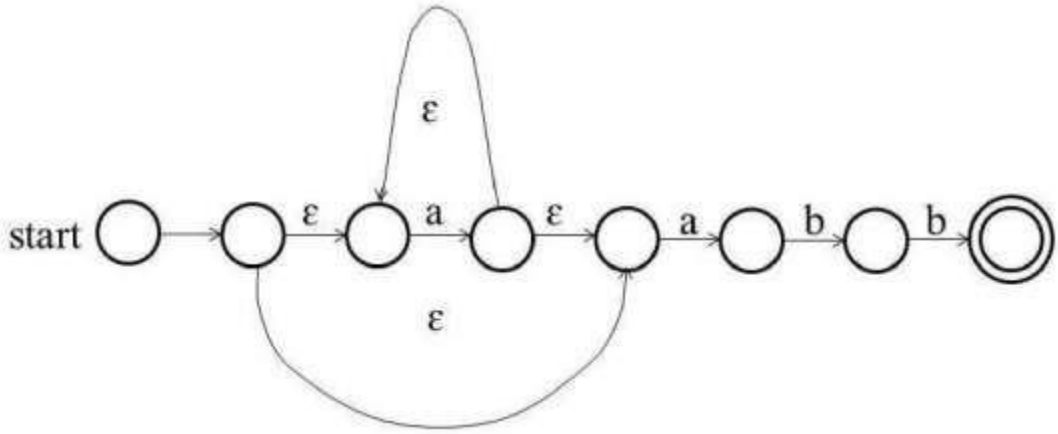
```
outtextxy((*x2-*x1)/2+*x1,*y1+30,pr);
ellipse(*x1+(*x2-*x1)/2,*y2+10,180,360,(*x2-*x1)/2+40,70);
outtextxy(*x2+37,*y2+14,"^");
if(*x1-40<minx)minx=*x1-40;
miny=*y1;
}
void star1(int *x1,int *y1,int *x2,int *y2)
{
char pr[10];
ellipse(*x1+(*x2-*x1)/2+15,*y2-10,0,180,(*x2-*x1)/2+15,70);
outtextxy(*x1-2,*y2-17,"v");

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);
```

```
outtextxy(*x1-15,*y1-3,">");
outtextxy(*x3-15,*y4-3,">");
circle(c1-40,(*y4-*y2)/2+*y2,10);
sprintf(pr,"%c",238);
outtextxy(c1-40,(*y4-*y2)/2+*y2+25,pr);
outtextxy(c1-40,(*y4-*y2)/2+*y2-25,pr);
line(*x2+10,*y2,c2+40,(*y4-*y2)/2+*y2-10);
line(*x3-10,*y3,c1-40,(*y3-*y1)/2+*y2+10);
circle(c2+40,(*y4-*y2)/2+*y2,10);
outtextxy(c2+40,(*y4-*y2)/2+*y2-25,pr);
outtextxy(c2-40,(*y4-*y2)/2+*y2+25,pr);
outtextxy(c2+35,(*y4-*y2)/2+*y2-15,"^");
outtextxy(c1+35,(*y4-*y2)/2+*y2+10,"^");
line(*x4+10,*y2,c2+40,(*y4-*y2)/2+*y2+10);
minx=c1-40;
miny=(*y4-*y2)/2+*y2;
}
void main()
{
int d=0,l,x1=200,y1=200,len,par=0,op[10];
int cx1=200,cy1=200,cx2,cy2,cx3,cy3,cx4,cy4;
char str[20];
int gd=DETECT,gm;
int stx[20],endx[20],sty[20],endy[20];
int pos=0,i=0;
clrscr();
initgraph(&gd,&gm,"c:\\dosapp\\tcplus\\bgi");
printf("\n enter the regular expression:");
scanf("%s",str);
len=(strlen(str));
while(i<len)
{
if(isalpha(str[i]))
{
if(str[i+1]=='*')x1=x1+40;
basis(&x1,&y1,str[i]);
stx[pos]=x1;
endx[pos]=x1+40;
sty[pos]=y1;
endy[pos]=y1;
x1=x1+40;
pos++;
}
if(str[i]=='*')
{
star(&stx[pos-1],&sty[pos-1],&endx[pos-1],&endy[pos-1]);
stx[pos-1]=stx[pos-1]-40;
```

```
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.

Ex.No: 2

Date:

Construction of DFA from NFA

Aim:

To write a C program to construct a DFA from the given NFA.

Algorithm:

1. Start the program.
2. Accept the number of state A and B.
3. Find the E-closure for node and name if as A.
4. Find v(a,a) and (a,b) and find a state.
5. Check whether a number new state is obtained.
6. Display all the state corresponding A and B.
7. Stop the program.

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:

Enter the NFA state table entries: 11

(Note: *Instead of* '-' *symbol use blank spaces in the output window*)

```
0 1 2 3 4 5 6 7 8 9 10 11
------------------------------------------------------------------------
1   - e - - - - - e - -  -
2   - - e - e - - - - -  -
3   - -  - a - - - - - -  -
4   - -  - - - - e - - -  -
5   - -  - - - b - - - -  -
6   - -  - - -    - e - - -  -
7   - e - - - - - e - -  -
8   - -  - - -    - - - e -  -
9   - -  - - -    - - - - e  -
10 ----------------------e
11 - -  - - -    - - - - - -
```

The Epsilon Closures Are:

```
E(1)={12358}
E(2)={235}
E(3)={3}
E(4)={234578}
E(5)={5}
E(6)={235678}
E(7)={23578}
E(8)={8}
E(9)={9}
E(10)={10}
E(11)={11}
```

DFA Table is:

	a	b
{12358}	{2345789}	{235678}
{2345789}	{2345789}	{23567810}
{235678}	{2345789}	{235678}
{23567810}	{2345789}	{23567811}
{23567811}	{2345789}	{235678}

Result:

The above C program was successfully executed and verified.

Ex.No: 3

Date:

Implementation of Recursive Descent Parser

Aim:

To write a C program to implement Recursive Descent Parser.

Algorithm:

Input: Context Free Grammar without last recursion and an input string from the grammar.

Output: Sequence of productions rules used to derive the sentence.

Method:

Consider the grammar

E->TE
E'->+TE'/e
T->FT
T->*FT/e
F->(E)/Id

To recursive decent parser for the above grammar is given below

Procedure:

```
Begin
T()
E_prime();
print E-> TE'
end

procedureeprime():
ifip_sym+='+'    then
begin
advance();
T();
eprime();
prime E'->TE'
end else
print E'->e

procedure
T(); begin
e();
```

```
Tprime();
print T->FT';
end;

procedureTprime();
ifip_sym='*' then
begin
advance();
F();
Tprime()
print T'->T*FT'
end else print
T'->e

procedure F()
ifip_ sym =id
then begin
advance();
print->id
end
else
Error();
end;
else
Error();
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
char ip_sym[15],ip_ptr=0;
void e_prime();
void t();
void e();
void t_prime();
void f();
void advance();
void e()
{
printf("\n\t\tE' ------ >TE'"); t(); e_prime();
}
void e_prime()
{
```

```
if(ip_sym[ip_ptr]=='+')
{
printf("\n\t\tE' ------- >+TE'");
advance();
t();
e_prime();
}
else printf("\n\t\tE'---- >e'");
}
void t()
{
printf("\n\t\tT' ------>FT'");
f();
 t_prime();
}
void t_prime()
{
if(ip_sym[ip_ptr]=='*')
{
printf("\n\t\tT----- >*FT'");
advance();
f();
t_prime();
}
else
{
printf("\n\t\tT' ---->e");
}
}
void f()
{
if((ip_sym[ip_ptr]=='i')||(ip_sym[ip_ptr]=='j'))
{
printf("\n\t\tF----- >i");
advance();
}
else
{
if(ip_sym[ip_ptr]=='(')
{
advance();
e();
if(ip_sym[ip_ptr]==')')
{
advance(); printf("\n\t\tF---->(E)");
}
```

```
else
{
printf("\n\t\tSyntax Error");
getch(); exit(1);
}
}
}
}
void advance()
{
ip_ptr++;
}
void main()
{
int i;
clrscr();
printf("\n\t\tGRAMMER WITHOUT RECURSION");
printf("\n\t\tE------>TE'\n\t\tE'/e\r\t\tT ---->FT");
printf("\n\t\tT------>*FT/e\n\t\tF ----->(E)/id");
printf("\n\t\tEnter the Input Symbol: "); gets(ip_sym);
printf("\n\t\tSequence of Production Rules");
e();
getch();
}
```

Sample Input & Output:

```
GRAMMER WITHOUT RECURSION
E------>TE'
T---- >FT
T----- >*FT/e
F----- >(E)/id

Enter the Input Symbol: T

Sequence of Production Rules
E' ------ >TE'
T' ------ >FT'
T' ---- >e
E' ---- >e'
```

Result:

The above C program was successfully executed and verified.

Ex.No: 4

Date:

IMPLEMENTATION OF THREE ADDRESS CODE FOR ASSIGNMENT STATEMENT

Aim:

To write a C program to implement three address code for Assignment statement.

Algorithm:

Input : Context free Grammar supported strings involving + & *.

Output : Three address code for given ststements.

Method : The three address code generator can be described below.

```
Procedure E();
begin
var i=t();
Eprime(var);
end;
procedure Eprime(var):
If input symbol='+'then
begin
advance();
tvar1=t();
print var="var"+"var1";
Eprime(var);
end;
procedure T() return var;
begin
vari=F();
tprime(var);
end;
procedure Tprime(int var);
begin
advance();
var1=t();
print var+"="+var"+"var1;
Tprime(var);
end;
procedure F()return var;
begin
if(ip[ip-ptr]>='A'&&(ip[ip-ptr]^='z'))
advance();
var++;
print var=ip-sym[ip-ptr];
end;
begin
```

```
    if((ip-sym[ip-ptr]=='c')
    advance();
    e();
    end;
    else return var1;
    procedure advance();
    ip-ptr ++;
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
char ip_sym[15],ip_ptr=0;
void e();
void e_prime(int);
void t_prime(int);
int t();
int f();
void advance();

void e()
{
int var;
var=t();
e_prime(var);
}

void e_prime(int var)
{
int var1;
if(ip_sym[ip_ptr]=='+')
{
advance();
var1=t();
printf("\n t%d=t%d + t%d",var,var,var1);
e_prime(var);
}    }

int t()
{
int var;
var=f();
t_prime(var);
return(var);
}
void t_prime(int var)
{
int var1;
if(ip_sym[ip_ptr]=='*')
```

```
{
advance();
var1=f();
printf("\n t%d = t%d * t%d",var,var,var1);
t_prime(var);
}
}
int f()
{
int static var=0;
if(isalpha(ip_sym[ip_ptr])!=0)
{
advance();
var++;
printf("\n t%d = %c",var,ip_sym[ip_ptr-1]);
return(var);
}
else
{
getch();
exit(1);
return 0;
}
}

void advance()
{
ip_ptr++;
}
void main()
{
int i;
clrscr();
printf("\n\t Enter an expression(involving + and * a-z):");
scanf("%s",ip_sym);
printf("\n\t Three adress code are:\n");
printf("\n\t .....\n");
e();
for(i=0;i<strlen(ip_sym);i++)
{
if(ip_sym[i]!='+' && ip_sym[i]!='*' && ip_sym[i]!='(' && ip_sym[i]!=')' && isdigit(ip_sym[ip_ptr])!=0)
{
printf("\n Syntax error");
break;
}
}
getch();
}
```

Sample output:

Enter an expression(involving + and * a-z):a+b*c+d

Three adress code are:

```
t1 = a
t2 = b
t3 = c
t2 = t2 * t3
t1=t1 + t2
t4 = d
t1=t1 + t4
```

Result:

The above Algorithm is implemented in C and the output is verified.

Ex.No: 5

Date:

Implementation of Code Generator

Aim:

To write a C program to implement Simple Code Generator.

Algorithm:

Input: Set of three address code sequence.

Output: Assembly code sequence for three address codes (opd1=opd2, op, opd3).

Method:

1- Start
2- Get address code sequence.
3- Determine current location of 3 using address (for 1st operand).
4- If current location not already exist generate move (B,O).
5- Update address of A(for 2nd operand).
6- If current value of B and () is null,exist.
7- If they generate operator () A,3 ADPR.
8- Store the move instruction in
memory 9-Stop.

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)
{
printf("R%d, R%d",k,reg);
preg[k].alive=0;
flag=0;
break;
}
}
if(flag)
{
printf(" %s,R%d",var[vc-1],reg);
printf("\nMov %s,R%d",fstr,reg);
}
strcpy(preg[reg].var,var[vc-3]);
getch();
}
}
```

Sample Input & Output:

Enter the Three Address
Code: a=b+c
c=a*c
exit

The Equivalent Assembly Code is:

```
Mov R0,b
Add c,R0
Mov a,R0
Mov R1,a
Mul c,R1
Mov c,R1
```

Result:

The above C program was successfully executed and verified.