PART B
EXPERIMENT NUMBER 5

Aim: Write a program to implement any code optimization techniques.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 30/03/2021	**Date of Submission:** 30/03/2021
Grade:	

B.1 Software Code written by a student:
(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-5.C**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct op
{
char l;
char r[20];
}
op[10],pr[10];
int main()
{
int a,i,k,j,n,z=0,m,q;
char *p,*l;
char temp,t;
char *tem;
printf("ENTER THE NUMBER OF VALUES: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("LEFT: ");
scanf(" %c",&op[i].l);
```

```
printf("RIGHT: ");
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\n",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';
}
}
}
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);
}
}
}
```

B.2 Input and Output:

```
C:\Users\ameyt\Desktop>gcc SPCC-5.C

C:\Users\ameyt\Desktop>a.exe
ENTER THE NUMBER OF VALUES: 5
LEFT: a
RIGHT: 9
LEFT: b
RIGHT: c+d
LEFT: e
RIGHT: c+d
LEFT: f
RIGHT: b+e
LEFT: r
RIGHT: f
INTERMEDIATE CODE
a=9
b=c+d
e=c+d
f=b+e
r=f

AFTER DEAD CODE ELIMINATION
b       =c+d
e       =c+d
f       =b+e
r       =f
pos: 2
ELIMINATE COMMON EXPRESSION
b       =c+d
b       =c+d
f       =b+b
r       =f
OPTIMIZED CODE
b=c+d
f=b+b
r=f
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

Hence we learnt and observed the various code optimization techniques.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

Hence we can implement various code optimization techniques like dead code and common subexpression. The implementation of common sub-expression elimination is done and the output of the same is given above.

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. What are the types of block transformation?

Ans:

There are two types of basic block optimization. These are as follows:

1. **Structure preserving transformations:**

   The primary Structure-Preserving Transformation on basic blocks is as follows:

   a. **Common subexpression elimination:**

      In the common sub-expression, you don't need to be computed over and over again. Instead of this you can compute it once and keep it in-store from where it's referenced when encountered again.

      a : = b + c

      b : = a - d

      c : = b + c

      d : = a - d

      In the above expression, the second and forth expression computed the same expression. So the block can be transformed as follows:

      a : = b + c

      b : = a - d

c : = b + c

d : = b

b. Dead-code elimination:

A program may contain a large amount of dead code.

This can be caused when once declared and defined and forget to remove them in this case they serve no purpose.

Suppose the statement x:= y + z appears in a block and x is a dead symbol that means it will never subsequently be used. Then without changing the value of the basic block you can safely remove this statement.

c. Renaming temporary variables:

A statement t:= b + c can be changed to u:= b + c where t is a temporary variable and u is a new temporary variable. All the instances of t can be replaced with u without changing the basic block value.

d. Interchange of the statement:

Suppose a block has the following two adjacent statements:

t1 : = b + c

t2 : = x + y

These two statements can be interchanged without affecting the value of the block when the value of t1 does not affect the value of t2.

2. Algebraic transformations:

In the algebraic transformation, we can change the set of expressions into an algebraically equivalent set. Thus the expression x:= x + 0 or x:= x *1 can be eliminated from a basic block without changing the set of expressions.

Constant folding is a class of related optimization. Here at compile-time, we evaluate constant expressions and replace the constant expression with their values. Thus the expression 5*2.7 would be replaced by13.5.

Sometimes the unexpected common subexpression is generated by the relational operators like <=, >=, <, >, +, = etc.

Sometimes the associative expression is applied to expose common subexpression without changing the basic block value. if the source code has the assignments

a:= b + c

e:= c +d +b

The following intermediate code may be generated:

a:= b + c

t:= c +d

e:= t + b

2. State the Normal form of Block.

Ans:

- ➔ In compiler construction, a basic block is a straight-line code sequence with no branches in except to the entry and no branches out except at the exit. This restricted form makes a basic block highly amenable to analysis. Compilers usually decompose programs into their basic blocks as a first step in the analysis process. Basic blocks form the vertices or nodes in a control flow graph.
- ➔ The code in a basic block has:
    - One entry point, meaning no code within it is the destination of a jump instruction anywhere in the program.
    - One exit point, meaning only the last instruction can cause the program to begin executing code in a different basic block.