EXPERIMENT NO 5

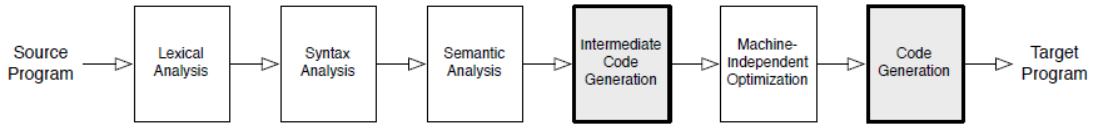
Date of performance :
Date of submission :

Aim: To perform code optimization, considering the target machine to be X86.

Software Used: c,c++,java language

Theory:

Code Optimization: An optimizing compiler is perhaps one of the most complicated, most powerful, and most interesting programs in existence. This chapter will talk about optimizations, although this chapter will not include a table of common optimizations.



Optimization within a compiler is concerned with improving in some way the generated object code while ensuring the result is identical. Technically, a better name for this chapter might be "Improvement", since compilers only attempt to improve the operations the programmer has requested. Optimizations fall into three categories:

- Speed; improving the runtime performance of the generated object code. This is the most common optimisation
- Space; reducing the size of the generated object code
- Safety; reducing the possibility of data structures becoming corrupted (for example, ensuring that an illegal array element is not written to)

Unfortunately, many "speed" optimizations make the code larger, and many "space" optimizations make the code slower -- this is known as the space-time tradeoff.
Common Optimization Algorithms
Common optimization algorithms deal with specific cases:

1. Dead Code Elimination
2. Common Sub-expression Elimination
3. Copy Propagation
4. Code Motion
5. Induction Variable Elimination
6. Reduction In Strength
7. Function Chunking

1.Dead code Elimination:-
A variable said to be live in program if the value continue into its used sequentially variable is dead if it is be performed by eliminating such a dead code.
Eg. i-j;
……
x=i+10;
……
The optimization can be performed by eliminating the assignment statement i=j. this assignment statement is called dead assignment.

8. Common Sub-expression Elimination:-

This is an expression appearing repeatedly in program which is completed previously. If the operand of such sub expression don't get changed at all then result of sub expression is used instead of recomputing it each time.

```
Eg. t1:=4x;                          t1:=4*i
t2:=a[t1];                           t2:=a[t1]
t3:=4*j;          t3:=4*j
t4:=4*I;          t5:=n
t5:=n                    t6:=b[t1]+t5
t6:=6[t4]+t5
```

9. Code Motion

It's a technique which moves the code outside the loop. Here is the name if there lies some expression in the loop whole result remain unchanged even after executing the loop for several time then such an expression should be placed just before the loop.

```
Eg. while(i<=max-1)                  n=max-1;
{                                    while(i<=n)
sum:=sum+a[i];           {
}                                    sum:=sum+a[i];
                                     }
```

10. Induction Variable Elimination

A variable x is called as induction variable of loop L if the value of variable get changed every time if the its either decremented / incremented by some constant

Eg. b1

```
i=t+1;
t1=4*j;
t2:=a[t1];
```

If t2<10 goto b1. in the above code the value of i& t1 are in block state.i.e. when value of I goto incremented by 1 the t1 get incremented by 4. hence i& t1 are induction variables.

11. Reduction In Strength

The strength of certain operands is higher than other eg. Strength of x is higher than t. in strength reduction technique the higher strength operands can be replaced by lower strength operand.

```
Eg for(i=1;i<=50;i++)                temp:=7
{                                    for(i=10;i<=50;i++)
count :=1*7                          {   count:=temp;
}                                            temp:=temp+7;
                                     }
```

Here the value of count as 7,14,21& so on upto 50.

PROGRAM:-

```
Import java.ip.*;
Import java.util.*;
Import java.long.String;
Public class optimization{
Public static void main(String args[]) throws IOException
{
DataInputStream in=new DataInputStream(System.in);
String s1,s2;
String code[]=new String[10];
System.out.println("Enter the string1:);
```

```
S1=in.readLine();
System.out.println("Enter the string2:);
S2=in.readLine();
If(s1.equals(s2)){
System.out.println("enter string is duplicate");
S2=null;}
Else
System.out.println("enter string is correct");
System.out.println("enter the line of code");
Int n=Integer.praseInt(in.readLine());
System.out.println("enter the code of program");
For(int i=0;i<=n;i++)
Code[i]=in.readLine();
For(int i=0;i<n;i++)
{
Char c[]=code[i].toCharArray();
Char d[]=code[i+1].toCharArray();
If ((c[0]=='I')&&(c[1]=='n')&&(c[2]=='t'))
If(d[3]==c[4])
System.out.println("the line"+code[i+1]+"will not be excuted since it's a dead code")'
Else
System.out.prinln("code is corrected");
}}}
```

Output:-

```
Enter the string1
Int i=0;
Enter the string2
Int j=3;
Enter string is corrected
Enter the line of code:
3
Enter the code of program
Int k=0;
If (k)
K=K+1;
The line if(k) will not be excuted since it's a dead code
```

Conclusion: Hence we have studied code optimization.

SIGN AND REMARK

DATE

R1	R2	R3	R4	R5	Total (15 Marks)	Signature