Experiment 02 : Code optimization Techniques

Learning Objective: Student should be able to Analyse and Apply code optimization techniques to increase efficiency of compiler.

Tools: Jdk1.8, Turbo C/C++, Python, Notepad++

Theory:

Code optimization aims at improving the execution efficiency. This is achieved in two ways.

1. Redundancies in a program are eliminated.
2. Computations in a program are rearranged or rewritten to make it execute efficiently.

The code optimization must not change the meaning of the program.

Constant Folding:

When all the operands in an operation are constants, operation can be performed at compilation time.

Elimination of common sub-expressions:

Common sub-expression are occurrences of expressions yielding the same value.

Implementation:

1. expressions with same value are identified
2. their equivalence is determined by considering whether their operands have the same values in all occurrences
3. Occurrences of sub-expression which satisfy the criterion mentioned earlier for expression can be eliminated.

Dead code elimination

Code which can be omitted from the program without affecting its result is called dead code. Dead code is detected by checking whether the value assigned in an assignment statement is used anywhere in the program

Frequency Reduction

Execution time of a program can be reduced by moving code from a part of program which is executed very frequently to another part of program, which is executed fewer times. For ex. Loop optimization moves, loop invariant code out of loop and places it prior to loop entry.

Strength reduction

The strength reduction optimization replaces the occurrence of a time consuming operations by an occurence of a faster operation. For ex. Replacement of Multiplication by Addition

<u>Example:</u>

A=B+C

B=A-D

C=B+C

D=A-D

After Optimization:

A=B+C

B=A-D

C=B+C

D=B

Code :

```
import pandas as pd
print("One thing is noticed before
starting of the program is that it
is compulsory to exist an 'input.cs
v' file\n")
print("The formate of 'input.csv' f
ile is that it should contains a 'l
eft' header and a 'right' header wi
th values indicate left and right r
espectively.\n")
print("Here we can't consider 'equa
l(=)' sign because it doesn't effec
t a bit of code with or without its
 presence.\n")
print("This program is case sensiti
ve this means that 'd*10' and '10*d
' is treated in different way\n The
y are not same. ")
print("The formate and input file f
or this program is as below..")
a=pd.read_csv("input.csv")
c=a.shape
print(a)
b=[]
for i in range(c[0]):
    for j in range(i+1,c[0]):
        if(a['right'][i]==a['right'
][j]):
            for d in range(c[0]):
                b=b+[(len(a['right'
][d]))]
                for z in range(b[d]
):
                    if(a['right'][d
][z]==a['left'][j]):
                        x=list(a['r
ight'][d])
                        x[z]=a['lef
t'][i]
                        l=''.join(x
)
                        a['right'][
d]=a['right'][d].replace(a['left'][
j],a['left'][i])
            a['left'][j]=a['left'][
i]
df=pd.DataFrame(a)
df.to_csv('output1.csv',index=False
)
p=pd.read_csv("output1.csv")
print("After checking and putting t
he value of common exepression ")
print(p)
i=0
j=i+1
while(j<c[0]):
    if(p['right'][i]==p['right'][j]
):
        if(p['left'][i]==p['left'][
j]):
            p.drop([j],axis=0,inpla
ce=True)
            i+=2
            j+=1
        else:
            i+=1
    j+=1
print("After elemenating the common
 expression")
df=pd.DataFrame(p)
df.to_csv('output1.csv',index=False
)
p=pd.read_csv("output1.csv")
print(p)
c=p.shape
#print(c)
count=0
i=0
j=0
h=1
while(j<c[0] and i<c[0]):
    b=[]
    b=b+[(len(p['right'][j]))]
    for z in range(b[0]):
        #print(z,j,i)
        if(p['right'][j][z]==p['lef
t'][i]):
            count=1
    j+=1
```

```
    #print(j)
    #print(c[0])
    if(j==c[0]):
        #print(count)
        if(count!=1):
            p.drop([i],axis=0,inpla
ce=True)
            df=pd.DataFrame(p)
            df.to_csv('output1.csv'
,index=False)
            p=pd.read_csv("output1.
csv")
            3#print(p)
            c=p.shape
            print(c)
        i+=1
        j=0
print("After dead code elimination"
)
print(p)
df=pd.DataFrame(p)
df.to_csv('output1.csv',index=False
)
p=pd.read_csv("output1.csv")
c=p.shape
print("The final optimized code is.
...")
for i in range(c[0]):
    print(str(p['left'][i])+"="+str
(p['right'][i]))
```

Output

```
  left right
0    a     9
1    b   c+d
2    e   c+d
3    f   b+e
4    r     f
After checking and putting the value of common exepression
left right
0    a     9
1    b   c+d
2    b   c+d
3    f   b+b
4    r     f
After eliminating the common expression
left right
```

```
0    a     9
1    b   c+d
2    f   b+b
3    r     f
(3, 2)
After dead code elimination
left right
0    b   c+d
1    f   b+b
2    r     f
The final optimized code is....
b=c+d
f=b+b
r=f
```

Application: To optimize code for improving space and time complexity.

Result and Discussion: Code optimization is the process of improving the efficiency and performance of a program by reducing its resource utilization, such as CPU time, memory usage, or disk I/O. There are various techniques used for code optimization, including: Loop unrolling: This technique involves duplicating the loop body several times to reduce the number of iterations, which can improve the program's execution speed. Dead code elimination: This technique involves removing code that is never executed, which can reduce the program's size and improve its performance.

Learning Outcomes: The student should have the ability to

LO1: **Define** the role of Code Optimizer in Compiler design.
LO2: List the different principle sources of Code Optimization.
LO3: ***Apply*** different code optimization techniques for increasing efficiency of compiler.
LO4: ***Demonstrate*** the working of Code Optimizer in Compiler design.

Course Outcomes: Upon completion of the course students will be able to Evaluate the synthesis phase to produce object code optimized in terms of high execution speed and less memory usage.

Conclusion:

To conclude we have successfully completed this experiment and learned about code optimization techniques

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				