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 occurrence of a faster operation. For ex. Replacement of Multiplication by Addition

Example:

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

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

Design:

Result and Discussion:

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.

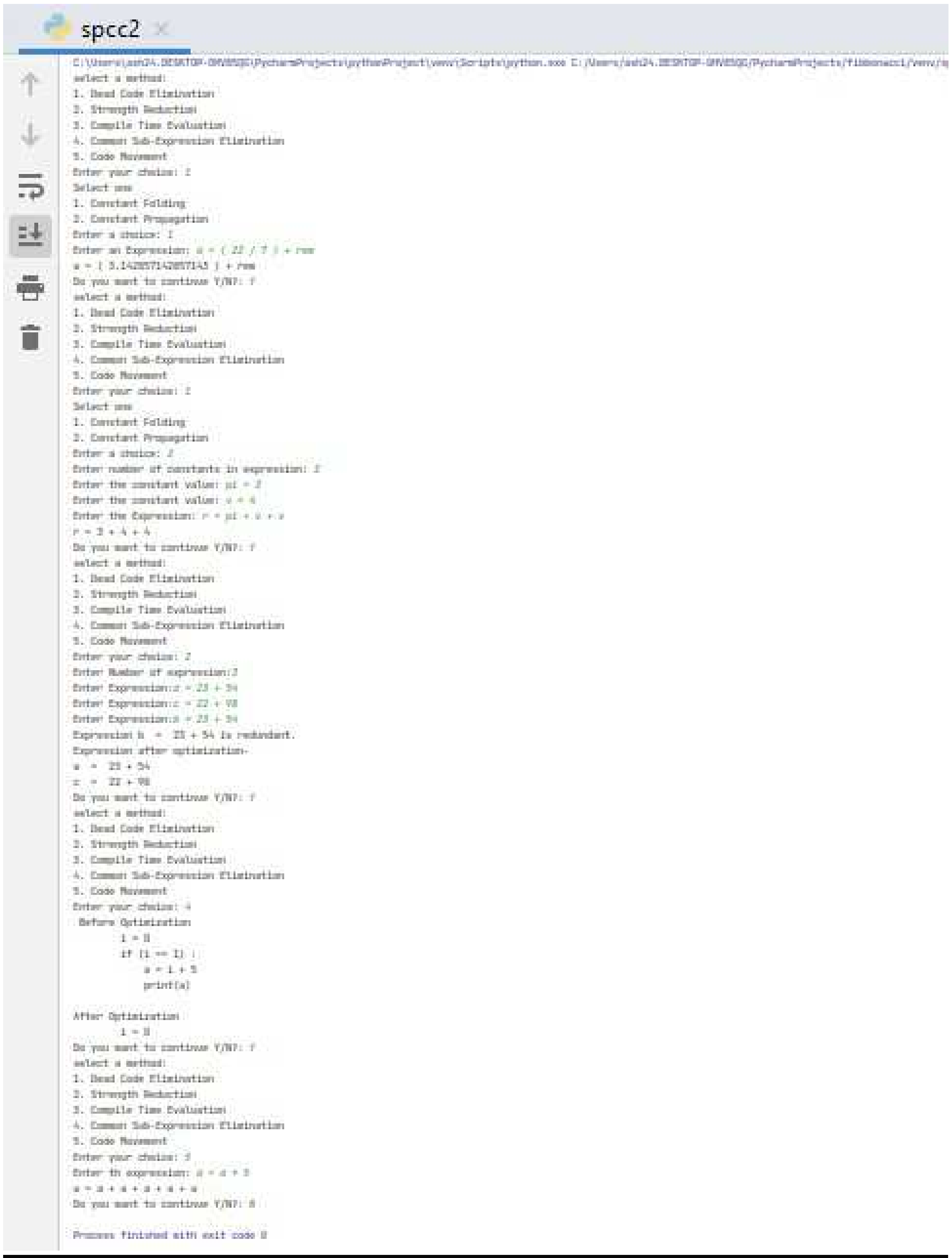
Conclusion:

For Faculty Use

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

```
from sqlalchemy import false, true

def strength_reduction():
    user_input = input("Enter th expression: ")
    l1 = list(user_input.split(" "))
    c = 0
    l2 = []
    for i in range(len(l1)):
        if l1[i] == "=":
            l2.append(l1[i])
            for j in range(i + 1, len(l1)):
                if l1[j] == "*":
                    for k in range(1, int(l1[j + 1])):
                        l2.append("+")
                        l2.append(l1[j - 1])
                    break
                elif l1[j] == "^":
                    for r in range(1, int(l1[j + 1])):
                        l2.append("*")
                        l2.append(l1[j - 1])
                    break
                else:
                    l2.append(l1[j])
            break
        else:
            l2.append(l1[i])
    print(" ".join(l2))


def compile_time_evaluation():
    print("Select one")
    print("1. Constant Folding")
    print("2. Constant Propagation")

    ch = int(input("Enter a choice: "))
    if ch == 1:
        user_input = input("Enter an Expression: ")
        l1 = list(user_input.split(" "))

        l2 = []
        i = 0
        while i < len(l1):
            if l1[i].isdigit():
                if l1[i + 1] == "+":
                    ans = str(int(l1[i]) + int(l1[i + 2]))
                elif l1[i + 1] == "*":
                    ans = str(int(l1[i]) * int(l1[i + 2]))
                elif l1[i + 1] == "-":
                    ans = str(int(l1[i]) - int(l1[i + 2]))
```

```
        inp = inp.split(" ")
        for i in range(len(inp)):
            if inp[i] == 'for':
                temp = inp[i + 2]
            if ('=' in inp[i]) and (temp not in inp[i]):
                print("Exp. {} is redundant.".format(inp[i][0:-1]))
            else:
                continue


def dead_code_elimination():
    print(""" Before Optimization
        i = 0
        if (i == 1) :
            a = i + 5
            print(a)
        """)

    print("""After Optimization
        i = 0 """)


user = true
while user == true:
    print("select a method: ")
    print("1. Dead Code Elimination")
    print("2. Strength Reduction")
    print("3. Compile Time Evaluation")
    print("4. Common Sub-Expression Elimination")
    print("5. Code Movement")

    choice = int(input("Enter your choice: "))
    if choice == 1:
        compile_time_evaluation()
    elif choice == 2:
        common_sub_exp_elimination()
    elif choice == 3:
        code_movement()
    elif choice == 4:
        dead_code_elimination()
    elif choice == 5:
        strength_reduction()
    else:
        print("Please select a valid choice.")

    con = input("Do you want to continue Y/N?: ")
    if con == ("Y" or "y"):
        user = true
    else:
        user = false
```



```
C:\Users\ash24.DESKTOP-0MV85QG\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/ash24.DESKTOP-0MV85QG/PycharmProjects/fibbonacci/venv/s
select a method:
1. Dead Code Elimination
2. Strength Reduction
3. Compile Time Evaluation
4. Common Sub-Expression Elimination
5. Code Movement
Enter your choice: 3
Select one
1. Constant Folding
2. Constant Propagation
Enter a choice: 1
Enter an Expression: a = ( 22 / 7 ) + rem
a = ( 3.142857142857143 ) + rem
Do you want to continue Y/N?: Y
select a method:
1. Dead Code Elimination
2. Strength Reduction
3. Compile Time Evaluation
4. Common Sub-Expression Elimination
5. Code Movement
Enter your choice: 3
Select one
1. Constant Folding
2. Constant Propagation
Enter a choice: 2
Enter number of constants in expression: 2
Enter the constant value: pi = 3
Enter the constant value: v = 4
Enter the Expression: r = pi + v + v
r = 3 + 4 + 4
Do you want to continue Y/N?: Y
select a method:
1. Dead Code Elimination
2. Strength Reduction
3. Compile Time Evaluation
4. Common Sub-Expression Elimination
5. Code Movement
Enter your choice: 2
Enter Number of expression:3
Enter Expression:a = 23 + 54
Enter Expression:c = 22 + 98
Enter Expression:b = 23 + 54
Expression b  =  23 + 54 is redundant.
Expression after optimization-
a  =  23 + 54
c  =  22 + 98
Do you want to continue Y/N?: Y
select a method:
1. Dead Code Elimination
2. Strength Reduction
3. Compile Time Evaluation
4. Common Sub-Expression Elimination
5. Code Movement
Enter your choice: 4
 Before Optimization
        i = 0
        if (i == 1) :
            a = i + 5
            print(a)

After Optimization
        i = 0
Do you want to continue Y/N?: Y
select a method:
1. Dead Code Elimination
2. Strength Reduction
3. Compile Time Evaluation
4. Common Sub-Expression Elimination
5. Code Movement
Enter your choice: 5
Enter th expression: a = a * 5
a = a + a + a + a + a
Do you want to continue Y/N?: N

Process finished with exit code 0
```