Ain Shams University

Faculty of Engineering

Software Performance Evaluation CSE423

Assignment 1

Submitted By:

Daniella George Botros 16P1073

Kirolos Raouf Atteya 16P8045

Submitted to:

Dr. Islam El-Maddah

Fall 2020

1. Problem Statement:

The project presents the idea of doing instrumentation of a given C++ code as an input and producing event log file known as stack call along with the execution time to measure performance of each function which is used for creating the call context tree known as CTT in an automated way for user simplicity and flexibility.

2. Specifications:

A- Code instrumentation

B- Code compiled and executed after instrumentation

C- Creating Stack Call

D- Measure Time in Micro-seconds taken by each function using clocks for accuracy

E- Creating CTT

3- REQUIREMENTS:

To compile and execute the instrumentation code (CodeInst_Output.ccp) you need to have:

- gcc/g++, which can be found in the Mingw tool.
- Python IDE

4- Language Used:

Python version 3

5- Assumption:

In The function declaration, the close bracket")" should be followed by the open curly bracket "{"in the same line to match the given REGEX used for function declaration detection.

Example:

- Int f1(…){…

6- Steps for using the tool:

A- Load the main.py file in python environment

B- Place the code you need to instrument inside InputCode.txt found in Input folder

C- Run the python code

D- 4 outputs files are created in Output Folder discussed with screenshots in later section of this report

a. CodeInst_output.cpp

b. CodeInst_Compilation_Output.o

c. FunctionEventLog_Output.txt

d. CCT_Output.txt

7- Test Cases:

We have used 2 test cases of different C++ codes to ensure our project is running as expected in a correct way (validation and verification)

8- Test Case 1

a. Input Code

- The input file name: InputCode.txt
- Contains: The code that the user wants to input.
- We used the following C++ code sample:

```
#include <iostream>
#include<stdio.h>
int get_square(int x) {
  return  x * x;
}

int get_cube(int x) {
  return x * x * x;
}


int get_add(int a, int b) {
  return get_square(a) + get_cube(b);
}
int get_mult(int a, int b) {
  return get_square(a) * get_cube(b);
}



int get_diff(int a, int b) {
  return get_square(a) - get_cube(b);
}



int get_dev(int a, int b) {
  return  get_diff(a, b);

}
int main() {
```

```
for (int m = 0; m <= 4; m++) {
  get_mult(2, 2);
}
for (int c = 0; c <= 4; c++) {
  get_cube(2);
}

for (int a = 0; a <= 7; a++) {
  get_add(2, 2);
}



for (int j = 0; j <= 5; j++) {
  get_square(2);
}

int k = 2;
get_diff(3, k);
if(k!=0)
  get_dev(3, k);

return 0;
}
```

a. Output Codes

- **Output File 1:**
  - Name: CodeInst_Output.cpp
  - Contains: The output resulting from adding the instrumental code to the code that is inputted by the user in the InputCode.txt file.
  - Code:

```
#include <iostream>
```

```
#include<stdio.h>

#include <iostream>

#include<stdio.h>

#include <chrono>

#include <fstream>

using namespace std;

using namespace std::chrono;

ofstream Prof("FunctionEventLog_Output.txt");

ofstream Path("CCT_Output.txt");

auto baseLineMilliseconds = high_resolution_clock::now();


int get_square(int x) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start get_square Function @" << durationStart << " microS" << endl;


int INSERTEDVALINSTR =    x * x;

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End get_square Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}
```

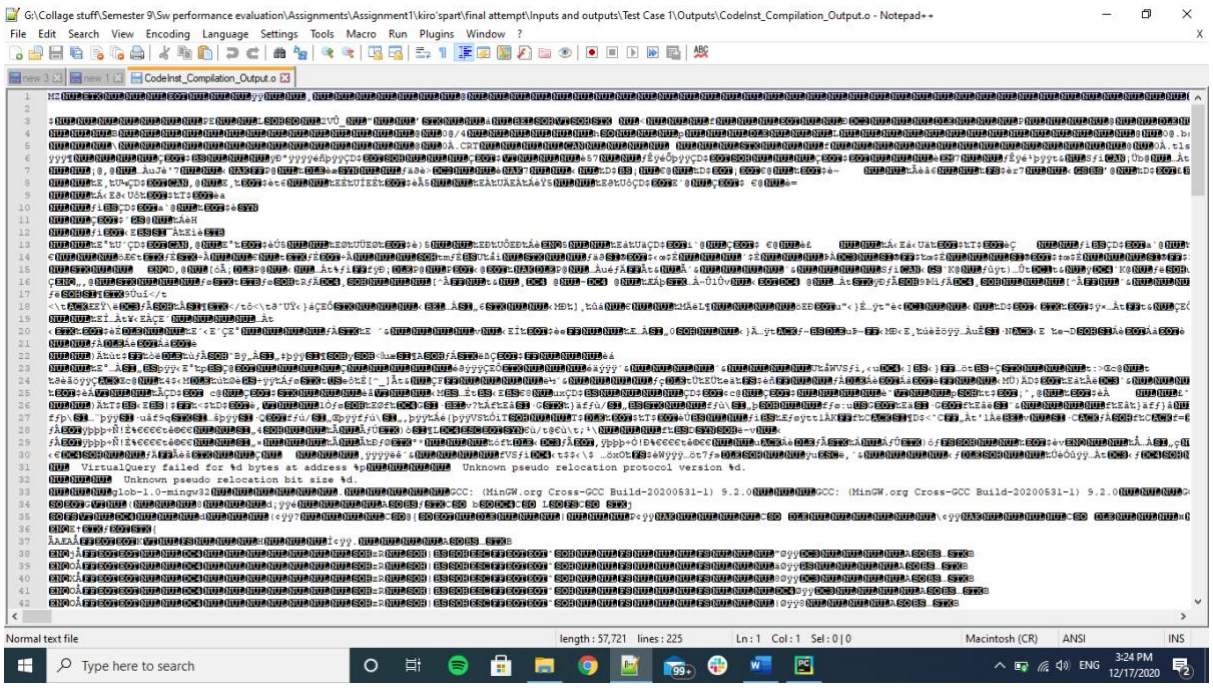
```
int get_cube(int x) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start get_cube Function @" << durationStart << " microS" << endl;


int INSERTEDVALINSTR =   x * x * x;

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End get_cube Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}


int get_add(int a, int b) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start get_add Function @" << durationStart << " microS" << endl;


int INSERTEDVALINSTR =   get_square(a) + get_cube(b);

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End get_add Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
```

```
}

int get_mult(int a, int b) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start get_mult Function @" << durationStart << " microS" << endl;


int INSERTEDVALINSTR =   get_square(a) * get_cube(b);

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End get_mult Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}

int get_diff(int a, int b) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start get_diff Function @" << durationStart << " microS" << endl;


int INSERTEDVALINSTR =   get_square(a) - get_cube(b);

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End get_diff Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
```

```
}

int get_dev(int a, int b) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start get_dev Function @" << durationStart << " microS" << endl;


int INSERTEDVALINSTR =    get_diff(a, b);

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End get_dev Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}

int main() {
baseLineMilliseconds = high_resolution_clock::now();

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime - baseLineMilliseconds).count();

Prof << "Start main Function @" << durationStart << " microS" << endl;

  for (int m = 0; m <= 4; m++) {

    get_mult(2, 2);

  }

  for (int c = 0; c <= 4; c++) {

    get_cube(2);
```

```
 }

 for (int a = 0; a <= 7; a++) {

   get_add(2, 2);

 }

 for (int j = 0; j <= 5; j++) {

   get_square(2);

 }

 int k = 2;

 get_diff(3, k);

 if(k!=0)

   get_dev(3, k);

int INSERTEDVALINSTR =   0;

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime - baseLineMilliseconds).count();

Prof << "End main Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}
```

- **Output File 2:**
  - Name: CodeInst_Compilation_Output.o
  - Contains: The result of compiling the modified code that is in the CodeInst_Output.cpp file.
  - Screen Shot:



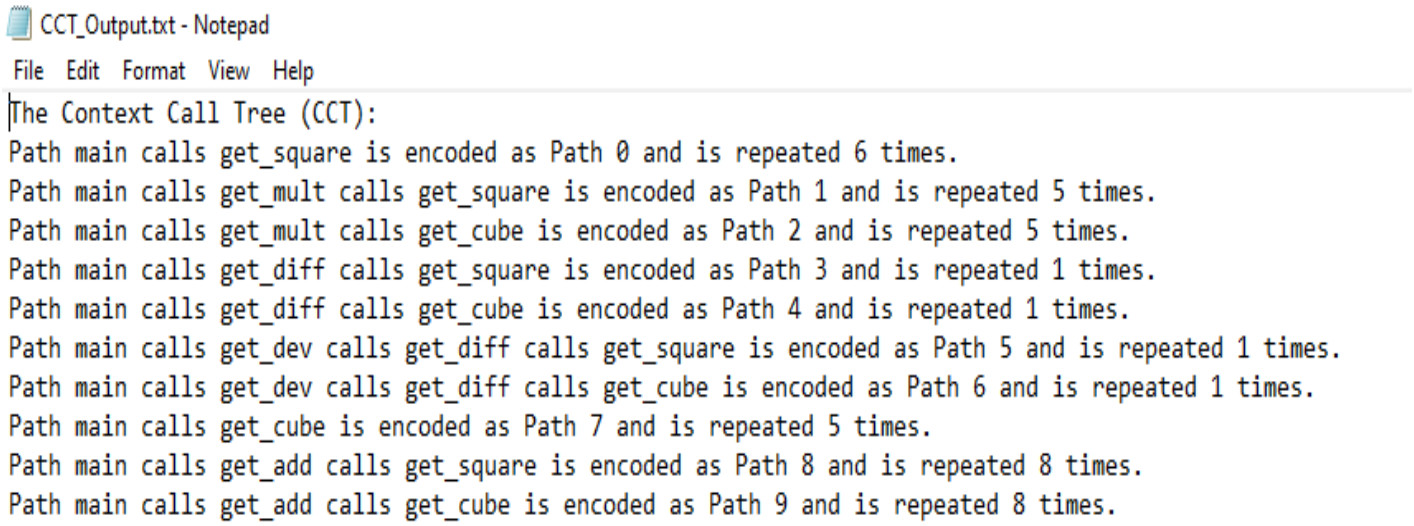
- **Output File 3:**
  - Name: FunctionEventLog_Output.txt
  - Contains: The output resulting from running the instrumental code, which shows simulates the running of the code with the start and end time of each function.
  - Screen Shot:

FunctionEventLog_Output.txt - Notepad

File Edit Format View Help

```
Start main Function @0 microS
Start get_mult Function @447414 microS
Start get_square Function @447569 microS
End get_square Function @447675 microS
Start get_cube Function @447789 microS
End get_cube Function @447881 microS
End get_mult Function @447973 microS
Start get_mult Function @448064 microS
Start get_square Function @448155 microS
End get_square Function @448247 microS
Start get_cube Function @448338 microS
End get_cube Function @448429 microS
End get_mult Function @448520 microS
Start get_mult Function @448612 microS
Start get_square Function @448734 microS
End get_square Function @448833 microS
Start get_cube Function @448925 microS
End get_cube Function @449523 microS
End get_mult Function @449621 microS
Start get_mult Function @449697 microS
Start get_square Function @449753 microS
End get_square Function @449803 microS
Start get_cube Function @449851 microS
End get_cube Function @449898 microS
End get_mult Function @449945 microS
Start get_mult Function @449992 microS
Start get_square Function @450039 microS
End get_square Function @450085 microS
Start get_cube Function @450132 microS
End get_cube Function @450179 microS
End get_mult Function @450228 microS
Start get_cube Function @450296 microS
End get_cube Function @450368 microS
Start get_cube Function @450442 microS
End get_cube Function @450518 microS
Start get_cube Function @450612 microS
End get_cube Function @450715 microS
Start get_cube Function @450808 microS
End get_cube Function @450885 microS
Start get_cube Function @450973 microS
End get_cube Function @451082 microS
Start get_add Function @451147 microS
Start get_square Function @451211 microS
End get_square Function @451261 microS
Start get_cube Function @451308 microS
End get_cube Function @451354 microS
End get_add Function @451400 microS
Start get_add Function @451447 microS
Start get_square Function @451494 microS
End get_square Function @451540 microS
Start get_cube Function @451587 microS
End get_cube Function @451632 microS
End get_add Function @451686 microS
Start get_add Function @451735 microS
Start get_square Function @451782 microS
End get_square Function @451828 microS
Start get_cube Function @451875 microS
End get_cube Function @451921 microS
End get_add Function @451967 microS
Start get_add Function @452013 microS
Start get_square Function @452059 microS
End get_square Function @452106 microS
Start get_cube Function @452152 microS
End get_cube Function @452198 microS
End get_add Function @452244 microS
Start get_add Function @452291 microS
Start get_square Function @452337 microS
End get_square Function @452383 microS
Start get_cube Function @452430 microS
End get_cube Function @452475 microS
End get_add Function @452521 microS
Start get_add Function @452567 microS
Start get_square Function @452613 microS
End get_square Function @452659 microS
Start get_cube Function @452909 microS
End get_cube Function @453022 microS
End get_add Function @453129 microS
Start get_add Function @453227 microS
Start get_square Function @453315 microS
End get_square Function @453429 microS
Start get_cube Function @453515 microS
End get_cube Function @453624 microS
End get_add Function @453729 microS
Start get_add Function @453801 microS
Start get_square Function @453873 microS
End get_square Function @453942 microS
Start get_cube Function @454012 microS
End get_cube Function @454077 microS
End get_add Function @454395 microS
Start get_square Function @454825 microS
End get_square Function @454913 microS
Start get_square Function @454972 microS
End get_square Function @455044 microS
Start get_square Function @455117 microS
End get_square Function @455190 microS
Start get_square Function @455266 microS
End get_square Function @455331 microS
Start get_square Function @455381 microS
End get_square Function @455427 microS
Start get_square Function @455473 microS
End get_square Function @455519 microS
Start get_diff Function @455565 microS
Start get_square Function @455611 microS
End get_square Function @455657 microS
Start get_cube Function @455863 microS
End get_cube Function @455939 microS
End get_diff Function @455989 microS
Start get_dev Function @456037 microS
Start get_diff Function @456085 microS
Start get_square Function @456131 microS
End get_square Function @456178 microS
Start get_cube Function @456225 microS
End get_cube Function @456271 microS
End get_diff Function @456317 microS
End get_dev Function @456363 microS
End main Function @456410 microS
```

- **Output File 4:**
  - Name: CCT_Output.txt
  - Contains: The Context Call Tree which, is obtained from the EventLog_Output.txt file.
  - Screen Shot:



9- Test Case 2:

a. Input Code

- The input file name: InputCode.txt
- Contains: The code that the user wants to input.
- We used the following C++ code sample:

```
int f2(int i) { return i + 2; }
int f1(int i) { return f2(2) + i + 1; }
int f0(int i) { return f1(1) + f2(2); }
int pointed(int i) { return i; }
int not_called(int i) { return 0; }
int main(int argc, char **argv) {
    int (*f)(int); // pointer to function
    f0(1);
    f1(1);
    f = pointed;
    if (argc == 1)
        f(1);
    if (argc == 2)
        not_called(1);
    return 0;
}
```

b. Output Codes

- **Output File 1:**
  - Name: CodeInst_Output.cpp
  - Contains: The output resulting from adding the instrumental code to the code that is inputted by the user in the InputCode.txt file.
  - Code:

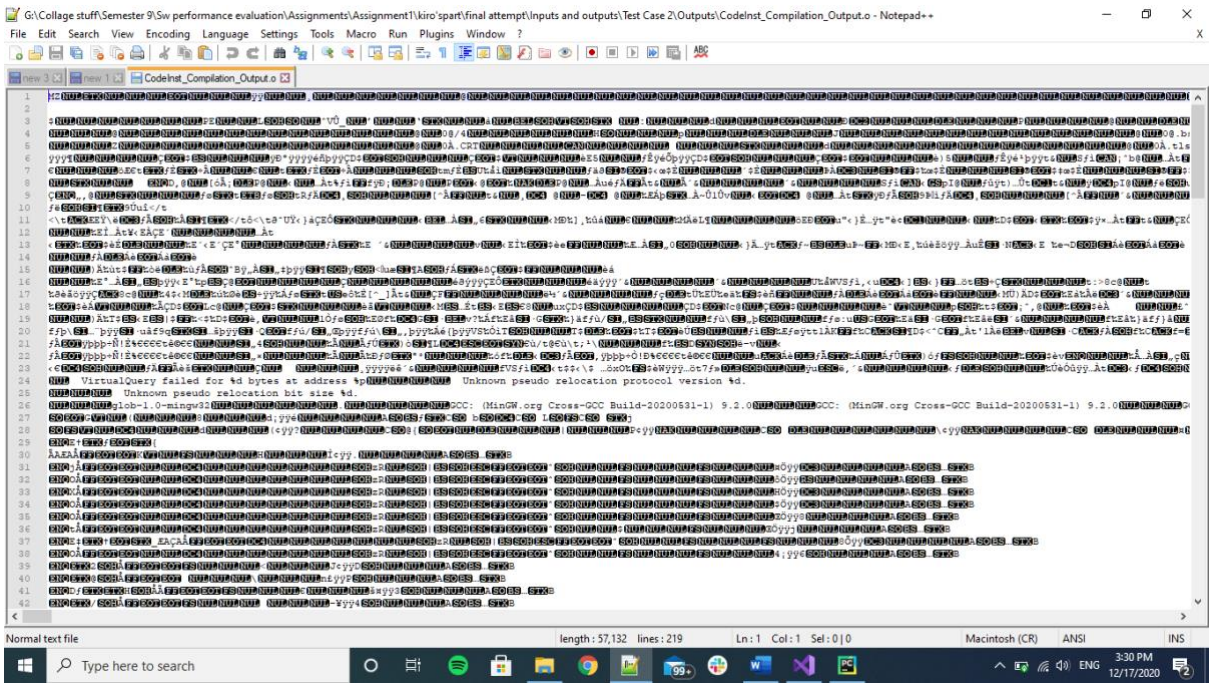
```
#include <iostream>

#include<stdio.h>

#include <chrono>

#include <fstream>

using namespace std;

using namespace std::chrono;

ofstream Prof("FunctionEventLog_Output.txt");

ofstream Path("CCT_Output.txt");

auto baseLineMilliseconds = high_resolution_clock::now();


int f2(int i) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();

Prof << "Start f2 Function @" << durationStart << " microS" << endl;



int INSERTEDVALINSTR =  i + 2;
auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();

Prof << "End f2 Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}
int f1(int i) {
```

```
auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();

Prof << "Start f1 Function @" << durationStart << " microS" << endl;



int INSERTEDVALINSTR =  f2(2) + i + 1;
auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();

Prof << "End f1 Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}
int f0(int i) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();

Prof << "Start f0 Function @" << durationStart << " microS" << endl;



int INSERTEDVALINSTR =  f1(1) + f2(2);
auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();

Prof << "End f0 Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}
int pointed(int i) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();

Prof << "Start pointed Function @" << durationStart << " microS" << endl;



int INSERTEDVALINSTR =  i;
auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();

Prof << "End pointed Function @" << durationEnd << " microS" << endl;
```

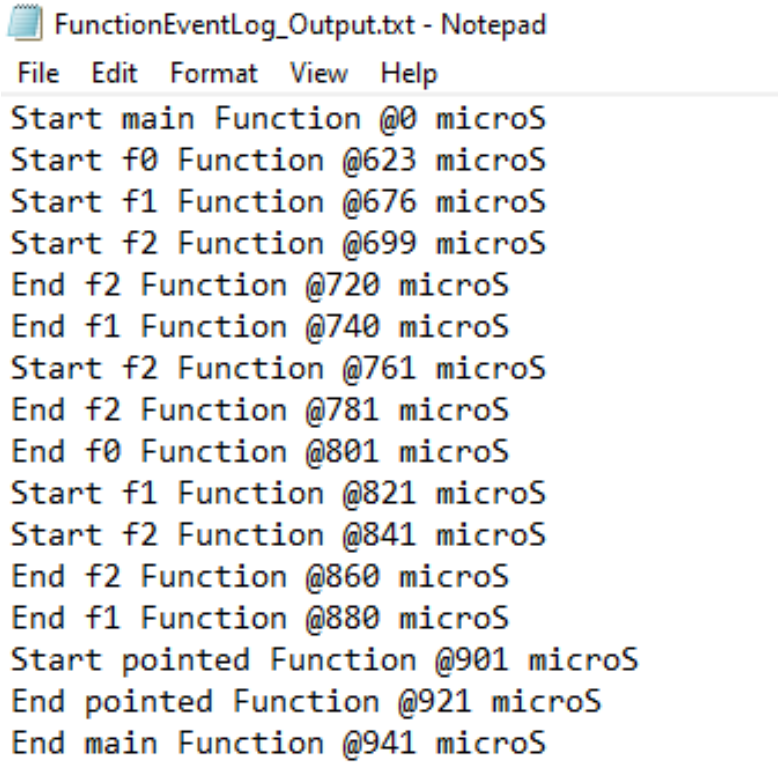
```
return INSERTEDVALINSTR ;
}
int not_called(int i) {

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();

Prof << "Start not_called Function @" << durationStart << " microS" <<
endl;



int INSERTEDVALINSTR =  0;
auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();

Prof << "End not_called Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;
}
int main(int argc, char **argv) {
baseLineMilliseconds = high_resolution_clock::now();

auto  startTime = high_resolution_clock::now();

auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();

Prof << "Start main Function @" << durationStart << " microS" << endl;



    int (*f)(int); // pointer to function

    f0(1);

    f1(1);

    f = pointed;

    if (argc == 1)

        f(1);

    if (argc == 2)

        not_called(1);

int INSERTEDVALINSTR =     0;

auto  endTime = high_resolution_clock::now();

auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();
```

```
Prof << "End main Function @" << durationEnd << " microS" << endl;

return INSERTEDVALINSTR ;


}
```

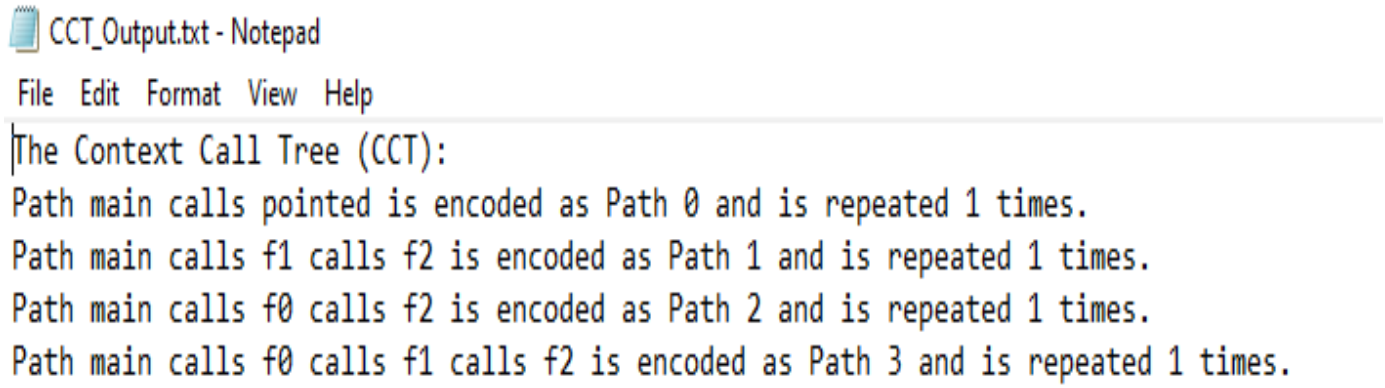
- **Output File 2:**
  - Name: CodeInst_Compilation_Output.o
  - Contains: The result of compiling the modified code that is in the CodeInst_Output.cpp file.
  - Screen Shot:



- **Output File 3:**
  - Name: FunctionEventLog_Output.txt
  - Contains: The output resulting from running the instrumental code, which shows simulates the running of the code with the start and end time of each function.
  - Screen Shot:



- **Output File 4:**
  - Name: CCT_Output.txt
  - Contains: The Context Call Tree which, is obtained from the EventLog_Output.txt file.
  - Screen Shot:



10- CODE:

- **Language:** Python.

- **Actual Code:**

```
import os
import re
def writeToFile(text):
    open('CodeInst_Output.cpp', 'w').close()
    f = open("CodeInst_Output.cpp", "a")
    for item in text:
        f.write("%s\n" % item)

    f.close()

def instrmentationAtCodeStart(codeReadedLines,insertIndex):
    s1='#include <iostream>\n'
    s2='#include<stdio.h>\n'
    s3='#include <chrono>\n'
    s4 ='#include <fstream>\n'
    s5 ='using namespace std;\n'
    s6 ='using namespace std::chrono;\n'
    s7 ='ofstream Prof("FunctionEventLog_Output.txt");\n'
    s8='ofstream Path("CCT_Output.txt");\n'
    s9='auto baseLineMilliseconds = high_resolution_clock::now();\n'
    s10=''
    codeInserted = [s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]
    newIndex=insertIndex+len(codeInserted)
    for i in range(0, len(codeInserted)):
        codeReadedLines.insert(insertIndex+i+1,codeInserted[i])
    return codeReadedLines, newIndex

def codeNameModifier(codeName):
    functionName=''
    state = False
    if ('void' in codeName):
        codeName=codeName.replace('void ', ' ')
    if ('int' in codeName):
        codeName=codeName.replace('int ',' ')
    if ('double' in codeName):
```

```
            codeName=codeName.replace('double ',' ')
        if ('float' in codeName):
            codeName= codeName.replace('float ',' ')
        if ('(' in codeName):
            codeName=codeName.replace('(', ' ')
        if (')' in codeName):
            codeName=codeName.replace(')', ' ')
        if ('{' in codeName):
            codeName=codeName.replace('{', ' ')
        for i in range(0,len(codeName)) :
            if codeName[i]==' ':
                state= True
            elif state:
                functionName=functionName+codeName[i]
                if codeName[i+1]==' ':
                    break
        return functionName

    def codeBracesStyler(codeLine,codeLines,index):
        i = 0
        countOpen=0
        countClose=0
        last=0
        codeLines[index] = ''
        while i < len(codeLine):
            codeLines[index] = codeLines[index]+codeLine[i]
            if (codeLine[i] == '{'):
                break
            i = i + 1
        codeLines.insert(index + 1, codeLine[i+1:len(codeLine)])

        if '}' in codeLines[index+1]:

            countClose= codeLines[index+1].count('}')
            countOpen =codeLines[index+1].count('{')
            if not(countClose-countOpen)==0:
                last=codeLines[index+1].rfind('}')
                codeLines[index+1]= codeLine[i+1:last+i+1]
                codeLines.insert(index + 2, '}')

```

```
    return codeLines

def instrmentationAtFunctionStart(codeReadedLines,insertIndex,codeName):
    s1 = ''
    if 'main' in codeName:
        s1='baseLineMilliseconds = high_resolution_clock::now();\n'
    s2 ='auto  startTime = high_resolution_clock::now();\n'
    s3 ='auto durationStart = duration_cast<microseconds>(startTime -
baseLineMilliseconds).count();\n'
    s4 ='Prof << "Start '+codeName +' Function @" << durationStart << " microS" << endl;'
    s5=''
    codeInserted = [s1,s2,s3,s4,s5]
    newIndex=insertIndex+len(codeInserted)-1
    for i in range(0, len(codeInserted)-1):
        codeReadedLines.insert(insertIndex+i+1,codeInserted[i])
    return codeReadedLines, newIndex

def
instrmentationAtFunctionEnd(codeReadedLines,insertIndex,codeName,functionType,argument
):
    s1 = ''
    s5 = ''
    if (functionType == 'double'):
        s1 = 'double INSERTEDVALINSTR =' + argument
    elif (functionType == 'int'):
        s1 = 'int INSERTEDVALINSTR =' + argument
    elif (functionType == 'float'):
        s1 = 'float INSERTEDVALINSTR =' + argument
    s2 ='auto  endTime = high_resolution_clock::now();\n'
    s3 ='auto durationEnd = duration_cast<microseconds>(endTime -
baseLineMilliseconds).count();\n'
    s4 ='Prof << "End '+codeName +' Function @" << durationEnd << " microS" << endl;\n'
    if not (functionType=='void'):
        s5 ='return INSERTEDVALINSTR ;'
    codeInserted = [s1,s2,s3,s4,s5]
    newIndex=insertIndex+len(codeInserted)
    for i in range(0, len(codeInserted)):
        codeReadedLines.insert(insertIndex+i+1,codeInserted[i])
    return codeReadedLines, newIndex
```

```
def functionStartEndIndexChecker (codeline):
    functionType=''
    foundval=False
    argument=''
    changeName=True
    if ('{' in codeline or '}' in codeline):
        foundval = True
    elif ('return' in codeline):
        argument = codeline.replace('return', '')
        foundval = True
    for index in range(0,len(codeline)-1):
        if not (codeline[index]==' ') and changeName:
            functionType=functionType+codeline[index]
            if codeline[index+1] == ' ':
                changeName= False
        if codeline[index]==';':
            functionType = ''
            break

    if not(functionType=='void'or functionType=='int'or functionType=='double'or
functionType=='float'):
        functionType=''
    return foundval , functionType,argument

def regexFunctionChecker(codeLine):
    patternToMatch = '((int |float |double |void )((?!=).)*\((.)*\) *){'
    result = re.findall(patternToMatch, codeLine)
    if result:
        return True
    else:
        return False

def readFileLineByLine():
    codeFile = open("InputCode.txt", "r")
    readedCode =codeFile.readlines()
    codeFile.close()
    return readedCode

def automatedCodeInstrumentation():
    endFunctionCandidate =0
```

```
    startStateFlag = True
    codeReadedLines = readFileLineByLine()
    functionName=''
    isRegex = True
    index =0
    firstFunction= True
    foundVal = False
    functionType = ''
    argumentVal=''
    while index < (len(codeReadedLines) - 1):
        if regexFunctionChecker(codeReadedLines[index]):

codeReadedLines=codeBracesStyler(codeReadedLines[index],codeReadedLines,index)
            isRegex = False
            if firstFunction:
                firstFunction=False
                codeReadedLines, index = instrmentationAtCodeStart(codeReadedLines, index-1)

        if isRegex == False :

foundVal,functionTypeDummy,argument=functionStartEndIndexChecker(codeReadedLines[ind
ex])
            if not (argument== ''):
                argumentVal=argument
                codeReadedLines[index]='\n'

            if foundVal:
                if startStateFlag:
                    functionName = codeReadedLines[index]
                    codeReadedLines, index = instrmentationAtFunctionStart(codeReadedLines,
index, codeNameModifier(functionName))
                    startStateFlag = False
                elif regexFunctionChecker(codeReadedLines[index]):
                    startStateFlag=True
                    codeReadedLines ,
index=instrmentationAtFunctionEnd(codeReadedLines,endFunctionCandidate-
1,codeNameModifier(functionName),functionType,argumentVal)
                    endFunctionCandidate = -1
                    isRegex = True
                    index-=1
```

```
                else:
                    endFunctionCandidate = index
            if not (functionTypeDummy == ''):
                functionType = functionTypeDummy
        index = index + 1

    codeReadedLines, index = instrmentationAtFunctionEnd(codeReadedLines,
endFunctionCandidate - 1, codeNameModifier(functionName), "int", argumentVal)
    writeToFile(codeReadedLines)

def sim_cpp():
    os.system('g++ CodeInst_Output.cpp -o CodeInst_Compilation_Output.o')
    os.system('CodeInst_Compilation_Output.o')

def get_paths():
    codeFile = open("FunctionEventLog_Output.txt", "r")
    arr = codeFile.readlines()
    codeFile.close()
    pointerParent = None
    pointerChild = 0
    path = []
    m = 0
    queue = []
    testPath = ''
    queue.append("main")
    for i in range(1, len(arr)):
        a = arr[i].split()
        if (a[0] == "Start"):
            if (pointerParent == pointerChild):
                path.insert(m, testPath)
                m = m + 1
                testPath = ''
            testPath = queue[len(queue) - 1] + " calls " + a[1]
            queue.append(testPath)
            pointerParent = pointerChild
            pointerChild = i
        elif (a[0] == "End"):
            if (pointerChild == pointerParent):
                pointerParent = pointerParent - 1
            pointerChild = pointerParent
```

```
            if (len(queue) >= 1):
                queue.pop()
            if (len(queue) == 0):
                path.insert(m + 1, testPath )
    return path

def get_profiling(path):
    path.sort()
    count = [1]
    key_id = 0
    non_rep_path = []
    non_rep_path.append(path.pop())
    for k in range(len(path) - 1, -1, -1):
        if (non_rep_path[key_id] == path[k]):
            count[key_id] = count[key_id] + 1
            path.pop()
        else:
            key_id = key_id + 1
            count.append(1)
            non_rep_path.append(path.pop())
    return non_rep_path, count

def get_CCT(non_rep_path, count):
    file2 = open("CCT_Output.txt", "w")
    l = []
    l.extend("The Context Call Tree (CCT): \n")
    for j in range(len(non_rep_path)):
        l.append("Path " + str(non_rep_path[j]) + " is encoded as Path " + str(j) + " and is repeated
" + str(
            count[j]) + " times.\n")
    file2.writelines(l)
    file2.close()
if __name__ == '__main__':
    automatedCodeInstrumentation()
    sim_cpp()
    path=get_paths()
    non_rep_path, count= get_profiling(path)
    get_CCT(non_rep_path, count)
```