CSC 840

Project 7

Matthew Wishoff

5/26/16

Contents

1.0 Abstract

A profiler is useful for when you want to see how often a block of code is executed. How it works is the profiler will run on a C++ source program, and leave annotated comment blocks on how many times each block has run. Let's say you have a program, and you are unsure of the complexity of the loops you created, you could easily run the profiler on your program and then get the results annotated into your program. You could then check to see how well your program is performing and if the logic you are using is correct, instead of debugging with print statements.

2.0 Software used

For this project I used PyCharm the professional edition to develop my Python profiler. The reason I ended up writing the profiler in python was due to Python's easy file I/O. It allowed me to read in an entire C++ file into a list allowing each line to have its own index. I used Dev-C++ to develop some test code to initially test the profiler to see how well it works. I also used notepad for the outputted text file from the profiler.

3.0 Process of how the profiler works

The profiler works by taking in a C++ source file, and then reading that in line by line. My program first checks for left curly brackets counting all of them, and storing that value in a variable. After that it goes through the list again injecting a global array above the main function. This global array is given the size of how many curly brackets you counted on the first pass through the program. As well as injecting the global array with an index value for each left curly bracket, and then attaching a ++ at the end of it to increment the global array value. At the end of this cycle through the program output code is injected right before the return 0 of the program. This allows the C++ code when executed to output the results of the global array to a text file. We then output the modified lines of code to a new C++ file. Once all the needed counters have been injected, we use a Python sub process call to first compile the C++ source code we just created generating an .exe file. Then we execute that same .exe file we just generated by the C++ code we just modified. This will generate a text file with the outputs of the global variable array. We will read this text file back into the Python script, and store the values in an array. We will then cycle through the original source program once more, this time on each left curly bracket we will annotate with block comments the number of times each block has been ran. After all this we should be left with a source program that has an annotated number of times each block of code has been ran as you will see in the appendix of this paper.

4.0 Experimental results

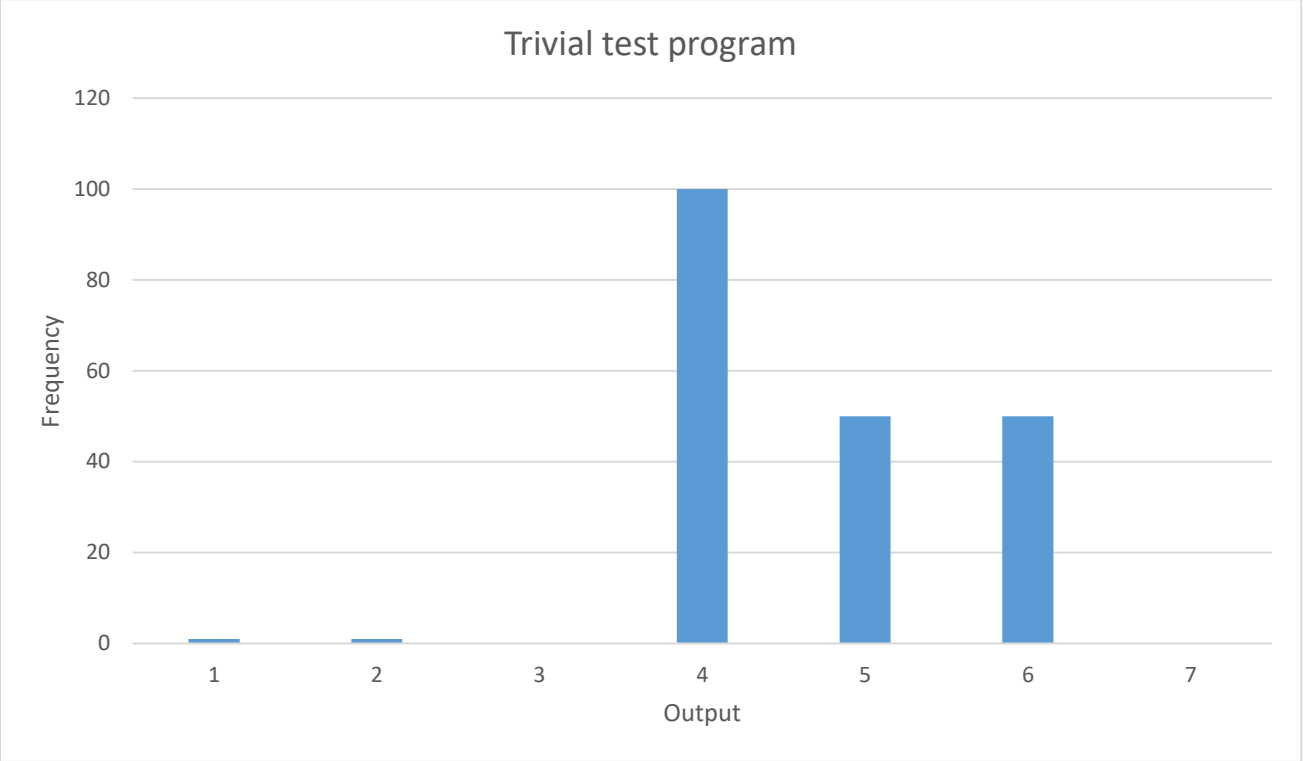

Figure 1

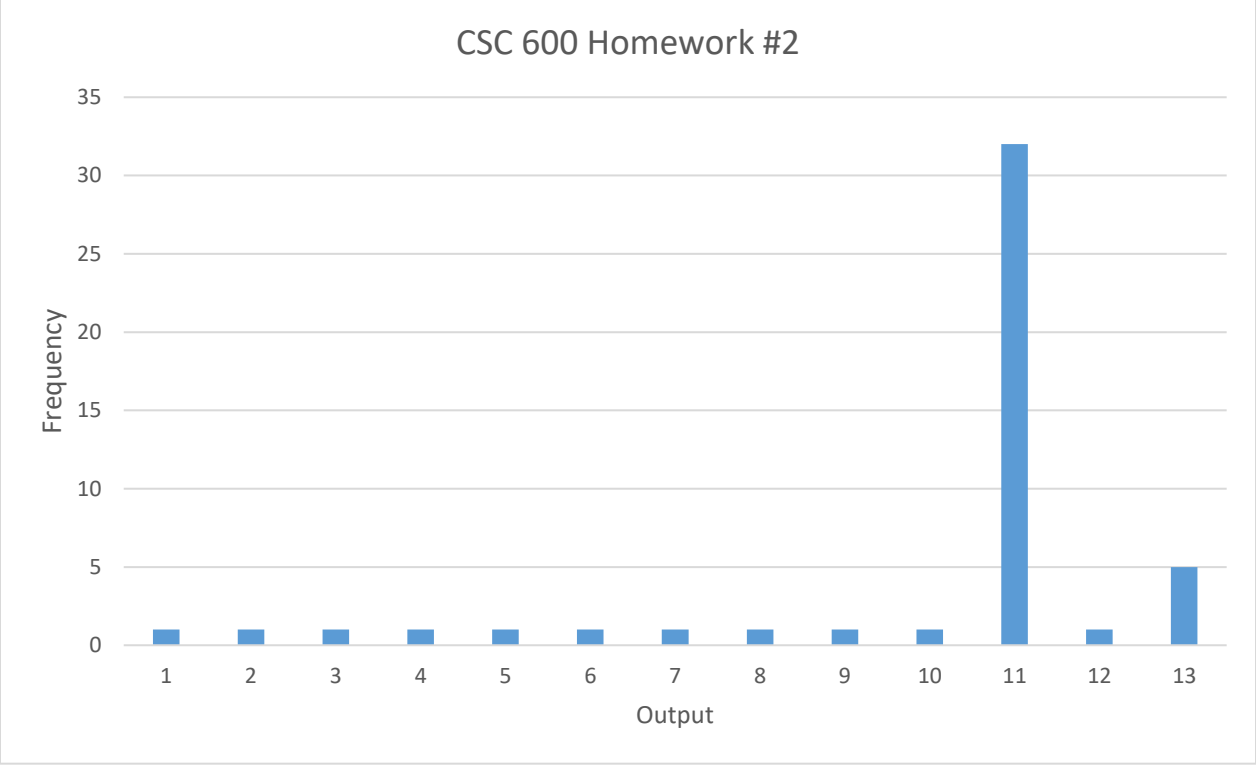

Figure 2

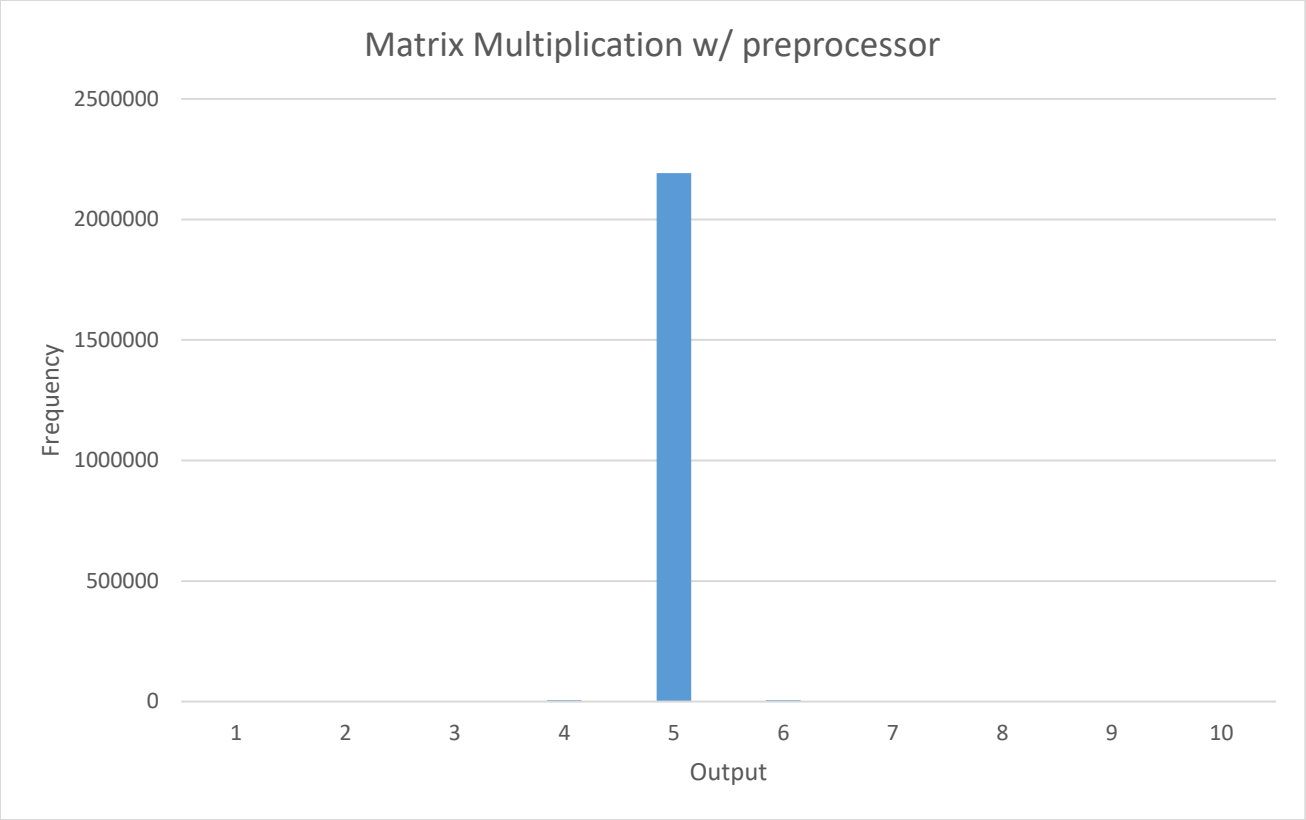

Figure 3

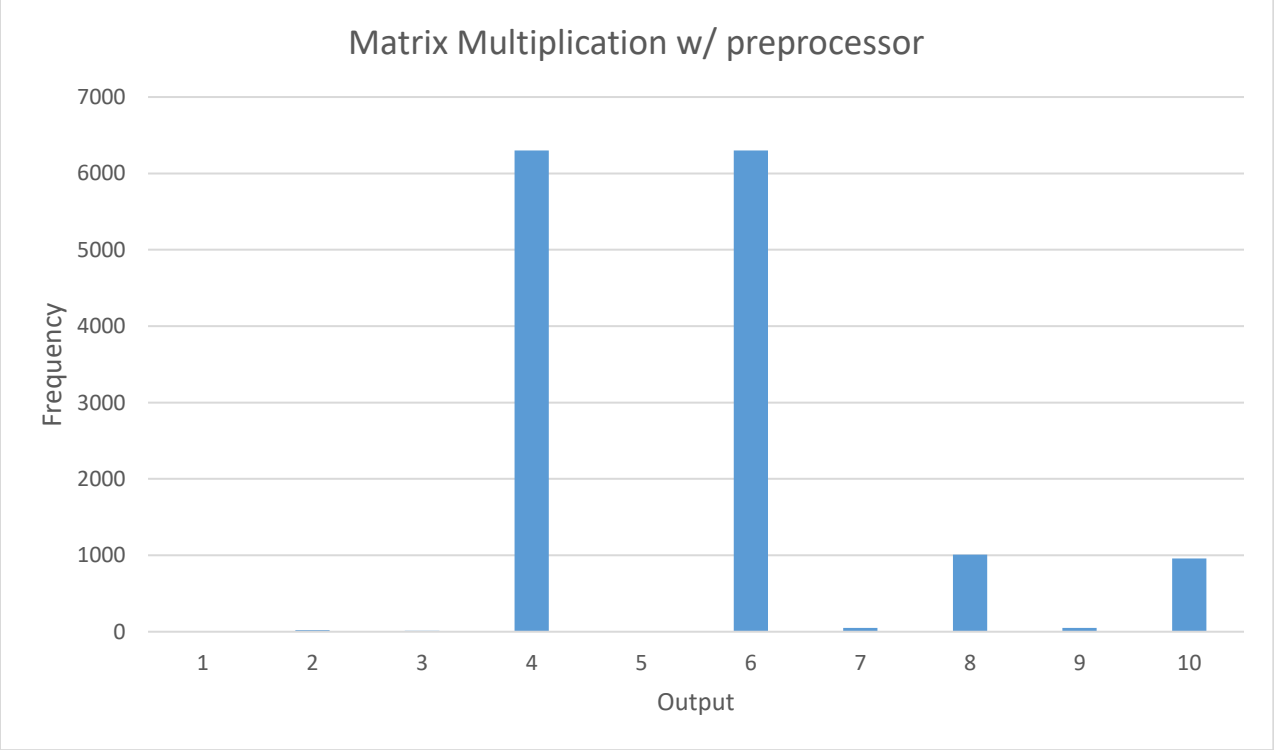

Figure 4

5.0 Analysis

Starting with figure one I like how you can tell how many iterations per block of code very clearly. You can see in figure one it spends most of its time in block 4 which is the for loop which loops for 100 times. And then an equal amount of time in the next two blocks 5 and 6 which are inside block 4 that print out the index of the for loop each loop through.

Looking at figure two I thought it was very interesting. You could almost run this profiler to check to see if the student did the homework correctly. You can see across the board that a lot of ones are there which is correct. There should be a lot of functions being called by the main function. This is with the exception of a for loop, and a while loop that calculates the square root of two. The other thing that I found interesting was that the while loop that calculates the square root of two only ran 5 times. This means that you can get an accurate approximation of the square root of two in only 5 iterations!

Looking at figure 3 you can see one bar shooting straight up to the sky like a rocket ship. This is the loop in which the matrices are created. It's so large because at each iteration I delete the matrices, and recreate them with the appropriate size to try and keep performance strong in my program. So instead of just one graph for this I also added figure 4 which is the same graph minus the out liar which made the graph look poor. Here you can see the most costly thing I am doing is creating and deleting the matrices if you look at appendix 3 you will notice this. You might be asking yourself well this is matrix multiplication what happened to the multiplication? Well my profiler actually didn't account for the preprocessor voodoo I did. Also I don't have brackets on my matrix multiplication for loops, so no global counters were added. This is a little disappointing as I wanted to see how these two programs would act together.

6.0 Future additions

For the future I think it would be cool to have multiple linked files, and use the profiler on all of them simultaneously. For example a C++ project that has multiple linked files that execute together. Something else that would be nice to work in the future is my preprocessor C++ program with the profiler. It has issues with adding the / after the global counter that may easily be fixable in the future. Also it has an issue that I didn't use brackets with my for loops in the matrix multiplication so I would either need my program to detect that I need to add them, or make sure I add them myself. I would also like to expand my profiler to work for multiple languages. For example Java, Ruby, C, and Javascript. I think this might be possible as you can inject code into any language with a python script (I've seen Sam do it himself for his project 2). Overall this was an enjoyable project that I wouldn't mind adding more cool features to in the future.

7.0 Conclusions

Some conclusions that I can draw from this project are that you can inject code using a Python script into any language. It is a little painful the first time using sub process calls, but is total manageable. Something else that I found interesting was it was reasonably painless annotating the original source code with the commented lines of code of how many times it ran. I feel a tool like this could also be adapted to do some sort of automated grading. However, there should probably always be a human element at the end checking to see if it was done correctly, as student input is not always the greatest. The profiler could be adapted to check for output of the program possibly, and the number of time each block is ran so that you can check if methods are being called to the main function.

8.0 Appendix

1a.) Test program

```
//By: Matthew Wishoff
//Date: 5/22/16
//Purpose: to test my profiler program.
#include <iostream>
#include <stdio.h>
#include <chrono>

using namespace std;

int main(int argc, char**argv)
{
	if(true)
	{
		cout << "AGH test program" << endl;
	}
	else
	{
		cout << "not true";
	}
	for(int i = 1; i <= 100; i++)
	{
		if(i % 2 == 0)
		{
			cout << i << endl;
```

```
                }
                else if(i %2 != 0)
                {
                        cout << i << endl;
                }
                else
                {
                        cout << "Probably broken" << endl;
                }
        }
        cout << "This is a parser thing mother fucker" << endl;

        return 0;
}
```

1b.) Test program injected Counters

```
//By: Matthew Wishoff
//Date: 5/22/16
//Purpose: to test my profiler program.
#include <iostream>
#include <stdio.h>
#include <chrono>

using namespace std;

int GLOBAL_BODY_COUNTER[7];

int main(int argc, char**argv)
{GLOBAL_BODY_COUNTER[0]++;
        if(true)
        {GLOBAL_BODY_COUNTER[1]++;
                cout << "AGH test program" << endl;
        }
        else
        {GLOBAL_BODY_COUNTER[2]++;
                cout << "not true";
        }
        for(int i = 1; i <= 100; i++)
        {GLOBAL_BODY_COUNTER[3]++;
                if(i % 2 == 0)
                {GLOBAL_BODY_COUNTER[4]++;
                        cout << i << endl;
                }
                else if(i %2 != 0)
                {GLOBAL_BODY_COUNTER[5]++;
                        cout << i << endl;
                }
                else
                {GLOBAL_BODY_COUNTER[6]++;
                        cout << "Probably broken" << endl;
                }
        }
        cout << "This is a parser thing mother fucker" << endl;


        freopen ("DataForTestProgram.txt","w",stdout);

        cout << "Times Ran " << endl;
```

```
	for(int i = 0; i < 7; i++)
		cout << GLOBAL_BODY_COUNTER[i] << endl;

	return 0;
}
```

1c.) Test program Text file

```
Times Ran
1
1
0
100
50
50
0
```

1d.) Test program annotated

```
//By: Matthew Wishoff
//Date: 5/22/16
//Purpose: to test my profiler program.
#include <iostream>
#include <stdio.h>
#include <chrono>

using namespace std;

int GLOBAL_BODY_COUNTER[7];

int main(int argc, char**argv)
{/* Ran 1 times */
	if(true)
	{/* Ran 1 times */
		cout << "AGH test program" << endl;
	}
	else
	{/* Ran 0 times */
		cout << "not true";
	}
	for(int i = 1; i <= 100; i++)
	{/* Ran 100 times */
		if(i % 2 == 0)
		{/* Ran 50 times */
			cout << i << endl;
		}
		else if(i %2 != 0)
		{/* Ran 50 times */
			cout << i << endl;
		}
		else
		{/* Ran 0 times */
			cout << "Probably broken" << endl;
		}
	}
	cout << "This is a parser thing mother fucker" << endl;


	freopen ("DataForTestProgram.txt","w",stdout);

	cout << "Times Ran " << endl;
```

```
	for(int i = 0; i < 7; i++)
		cout << GLOBAL_BODY_COUNTER[i] << endl;

	return 0;
}
```

2.) CSC600 Assignment 2 Annotated

```
//By Matthew Wishoff
// 2/20/16

#include <iostream>
#include <time.h>
#include <math.h>
#include <string>
#define N 10000

// CSC 600 Assignment Outline
//1. Show that for unsigned int a,b and a>0, b>0, we can get a+b < a
//2. Show that for int a,b and a>0, b>0, we can get a+b < 0
//3. Show that for int a,b and a<0, b<0, we can get a+b > 0
//4. Show that for double x and x>0 we can get 1. + x = = 1.
//5. Show that for double a,b,c in some cases (a+b)+c != (c+b)+a
//6. Show the results of the following power function:
//pow(-2., 3), pow(-2., 3.0) , pow(-2., 3.00000000001)
//7. Show the memory size of the following constants 1. , 1.F, 1 , '1' , and "1"
//8. Display 1./3. using 20 digits and show the correct and incorrect digits
//9. Display all printable characters of the ASCII table in 3 columns:
//first column: 32-63, second column: 64-95, third column: 96-127. Each column
//must include the numeric value and the corresponding character. Following is
//an example of one of 32 rows in the ASCII table:
//33 ! 65 A 97 a
//10. Compute sqrt(2.) using your own program for square root function.

void numberOne();
void numberTwo();
void numberThree();
void numberFour();
void numberFive();
void numberSix();
void numberSeven();
void numberEight();
void numberNine();
void numberTen();

using namespace std;

int main()
{/* Ran 1 times */
        numberOne();
        cout << "*******************************************************" << endl;
        numberTwo();
        cout << "*******************************************************" << endl;
        numberThree();
        cout << "*******************************************************" << endl;
        numberFour();
        cout << "*******************************************************" << endl;
        numberFive();
        cout << "*******************************************************" << endl;
        numberSix();
        cout << "*******************************************************" << endl;
        numberSeven();
        cout << "*******************************************************" << endl;
        numberEight();
        cout << "*******************************************************" << endl;
        numberNine();
        cout << "*******************************************************" << endl;
        numberTen();

        return 0;
}
```

```
void numberOne()
{/* Ran 1 times */
	unsigned int a = 4294967095;
	unsigned int b = 1000;
	unsigned int Sum = 0;

	cout << "a = 4294967095" << endl;
	cout << "b = 1000" << endl;
	cout << "Sum before: " << Sum << endl;
	cout << "Sum = 4294967095 + 1000" << endl;
	Sum = a + b;
	cout << "Sum after: " << Sum << endl;
}

void numberTwo()
{/* Ran 1 times */
	int a = 2147483647;
	int b = 1000;
	int Sum = 0;

	cout << "a = 2147483647" << endl;
	cout << "b = 1000" << endl;
	cout << "Sum before: " << Sum << endl;
	cout << "Sum = " << a << " + " << b << endl;
	Sum = a + b;
	cout << "Sum after: " << Sum << endl;
}

void numberThree()
{/* Ran 1 times */
	int a = -2147483647;
	int b = -1000;
	int Sum = 0;

	cout << "a = -2147483647" << endl;
	cout << "b = -1000" << endl;
	cout << "Sum before: " << Sum << endl;
	cout << "Sum = " << a << " + " << b << endl;
	Sum = a + b;
	cout << "Sum after: " << Sum << endl;
}

void numberFour()
{/* Ran 1 times */
	double x = 0.000001;
	double y = 1.0;
	double Sum = 0.0;

	cout << "x = 0.000001" << endl;
	cout << "y = 1.0" << endl;
	cout << "Sum before: " << Sum << endl;
	cout << "Sum = " << x << " + " << y << endl;
	Sum = x + y;
	cout << "Sum after: " << Sum << endl;
}

void numberFive()
{/* Ran 1 times */
	double a = 0.003;
	double b = 0.05;
	double c = 0.07;

	cout << (a + b) + c << endl;
	cout << a + (b + c) << endl;
}
```

```
void numberSix()
{/* Ran 1 times */
		cout << pow(-2.0, 3) << endl;
		cout << pow(-2.0, 3.0) << endl;
		cout << pow(-2.0, 3.00000000001) << endl;
}

void numberSeven()
{/* Ran 1 times */
		double a = 1.0;
		float b = 1.0;
		int c = 1;
		char d = '1';
		string e = "1";

		cout << "Number of bytes in 1.0(double): " << sizeof(a) << endl;
		cout <<"Number of bytes in 1.0(float): " << sizeof(b) << endl;
		cout <<"Number of bytes in 1(int): " << sizeof(c) << endl;
		cout <<"Number of bytes in '1'(char): " << sizeof(d) << endl;
		cout <<"Number of bytes in '1'(string): " << sizeof(e) << endl;
}

void numberEight()
{/* Ran 1 times */
		double a = 1.0;
		double b = 3.0;

		cout << "Six digits of precesion: " << a/b << endl;

		cout.precision(20);

		cout << "20 digits of precesion: " << fixed << a/b << endl;
}

void numberNine()
{/* Ran 1 times */
		for(int i = 32; i <= 63; i++)
		{/* Ran 32 times */
			cout << (char)i << "		" << (char)(i + 32) << "		" << (char)(i + 64) << endl;
		}
}

void numberTen()
{/* Ran 1 times */
		double x = 2;

   double y = (1+x)/2;		//approximate the value of y
   double z = 0;
   while (y != z) {/* Ran 5 times */
      z = y;				//Compare y old to y new
      y = (y + x/y) / 2.0; //get closer to actuall square root value
   }
   cout << y << endl;		//Print y to console.

}
```

3.) Project 2 Matrix Multiplication

```
/*
* Class: CSC 840
* By: Matthew Wishoff
* Date: 3/4/2016
* Description: This program multiplies two matricies 48 different ways.
*				It also steps the matrix size by 20 on each iteration through the 48 ways.
```

```
*/
#include <iostream>
#include <stdio.h>
#include <chrono>

using namespace std;

#define DO_MULTIPLY_1 c[i][j] += a[i][k]*b[k][j]
#define DO_MULTIPLY_2 c[i][j] += a[i][k]*b[j][k]
#define DO_MULTIPLY_3 c[i][j] += a[k][i]*b[k][j]
#define DO_MULTIPLY_4 c[i][j] += a[k][i]*b[j][k]
#define DO_MULTIPLY_5 c[j][i] += a[i][k]*b[k][j]
#define DO_MULTIPLY_6 c[j][i] += a[i][k]*b[j][k]
#define DO_MULTIPLY_7 c[j][i] += a[k][i]*b[k][j]
#define DO_MULTIPLY_8 c[j][i] += a[k][i]*b[j][k]

#define DO_MULTIPLY_IJK(method)                                         \
        for(int m=0; m<M; m++)                                          \
                for(int i=0; i<n; i++)                                  \
                        for(int j=0; j<n; j++)                          \
                                for(int k=0; k<n; k++)                  \
                                        DO_MULTIPLY_##method;



#define DO_MULTIPLY_IKJ(method)                                         \
        for(int m=0; m<M; m++)                                                  \
                for(int i=0; i<n; i++)                                  \
                        for(int k=0; k<n; k++)                          \
                                for(int j=0; j<n; j++)                  \
                                        DO_MULTIPLY_##method;


#define DO_MULTIPLY_KJI(method)                                         \
        for(int m=0; m<M; m++)                                                  \
                for(int k=0; k<n; k++)                                  \
                        for(int j=0; j<n; j++)                          \
                                for(int i=0; i<n; i++)                  \
                                        DO_MULTIPLY_##method;


#define DO_MULTIPLY_KIJ(method)                                         \
        for(int m=0; m<M; m++)                                                  \
                for(int k=0; k<n; k++)                                  \
                        for(int i=0; i<n; i++)                          \
                                for(int j=0; j<n; j++)                  \
                                        DO_MULTIPLY_##method;


#define DO_MULTIPLY_JKI(method)                                         \
        for(int m=0; m<M; m++)                                                  \
                for(int j=0; j<n; j++)                                  \
                        for(int k=0; k<n; k++)                          \
                                for(int i=0; i<n; i++)                  \
                                        DO_MULTIPLY_##method;


#define DO_MULTIPLY_JIK(method)                                         \
        for(int m=0; m<M; m++)                                                  \
                for(int j=0; j<n; j++)                                  \
                        for(int i=0; i<n; i++)                          \
                                for(int k=0; k<n; k++)                  \
                                        DO_MULTIPLY_##method;
```

```
#define MULTIPLY(method)
                                        \
                    startTime = chrono::steady_clock::now();
          \
                              DO_MULTIPLY_IJK(method);
                                \
                    stopTime = chrono::steady_clock::now();
          \
                    RUN_TIME[TEST_NUM++ % NUM_OF_TESTS][col] = chrono::duration_cast<chrono::duration<double>>(stopTime -
startTime).count() / M;          \

                                                                                \
                    startTime = chrono::steady_clock::now();
\
                              DO_MULTIPLY_IKJ(method);
                                        \
                    stopTime = chrono::steady_clock::now();
                    \
                    RUN_TIME[TEST_NUM++ % NUM_OF_TESTS][col] = chrono::duration_cast<chrono::duration<double>>(stopTime -
startTime).count() / M;          \

                                                                                \
                    startTime = chrono::steady_clock::now();
          \
                              DO_MULTIPLY_KJI(method);
                                        \
                    stopTime = chrono::steady_clock::now();
          \
                    RUN_TIME[TEST_NUM++ % NUM_OF_TESTS][col] = chrono::duration_cast<chrono::duration<double>>(stopTime -
startTime).count() / M;          \

                                                                                \
                    startTime = chrono::steady_clock::now();
          \
                              DO_MULTIPLY_KIJ(method);
                                        \
                    stopTime = chrono::steady_clock::now();
          \
                    RUN_TIME[TEST_NUM++ % NUM_OF_TESTS][col] = chrono::duration_cast<chrono::duration<double>>(stopTime -
startTime).count() / M;          \

                                                                                \
                    startTime = chrono::steady_clock::now();
\
                              DO_MULTIPLY_JKI(method);
                                        \
```

```
                stopTime = chrono::steady_clock::now();

                \
                RUN_TIME[TEST_NUM++ % NUM_OF_TESTS][col] = chrono::duration_cast<chrono::duration<double>>(stopTime -
startTime).count() / M;          \


                                                                                \
                startTime = chrono::steady_clock::now();

        \
                        DO_MULTIPLY_JIK(method);

                                \
                stopTime = chrono::steady_clock::now();

                \
                RUN_TIME[TEST_NUM++ % NUM_OF_TESTS][col] = chrono::duration_cast<chrono::duration<double>>(stopTime -
startTime).count() / M;

#define NUM_OF_TESTS (8*6)
#define NMAX 500
#define NMIN 100
#define STEP 20

int TEST_NUM = 0;
double RUN_TIME[NUM_OF_TESTS][((NMAX - NMIN) / STEP) + 1];

int main(int argc, char**argv)
{ /* Times ran 1 */
        chrono::steady_clock::time_point startTime, stopTime;
        int col = 0;
        int M ;

        // Initialize
        for(int n=NMIN; n <= NMAX; n+=STEP, col++) //increase size of matrix
        { /* Times ran 21 */
                M =(NMAX*NMAX*NMAX)/(n*n*n);
                if(M < 4)
                { /* Times ran 10 */
                        M = 4;
                }
                double** a = new double*[n]; //Initialize to the min, the next iteration will step by 20
                double** b = new double*[n]; //Initialize to the min, the next iteration will step by 20
                double** c = new double*[n]; //Initialize to the min, the next iteration will step by 20

                cout << "before init" << endl;
                for(int i = 0; i < n; i++)
                { /* Times ran 6300 */
                        a[i] = new double[n];
                        b[i] = new double[n];
                        c[i] = new double[n];
                }

                // Matrix a[ ][ ], b[ ][ ], and c[ ][ ] initialization
                for(int i=0; i<n; i++)
                        for(int j=0; j<n; j++)
                        { /* Times ran 2198000 */
                                //Initialize diagonal to 2.0002, and everywhere else to 1.0001
                                if(i == j)
                                { /* Times ran 6300 */
                                        a[i][j] = 2.0002;
                                        b[i][j] = 2.0002;
                                }
                                else
                                { /* 2191700 */
```

```
                                        a[i][j] = 1.0001;
                                        b[i][j] = 1.0001;
                                }
                                c[i][j]=0.0;
                        }

                MULTIPLY(1); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(2); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(3); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(4); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(5); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(6); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(7); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array
                MULTIPLY(8); //multiply 6 different matricies and save their run times to the RUN_TIME 2d array

                //Delete dynamic array of pointers.
                for(int i = 0; i < n; i++)
                { /* Times ran 6300 */
                                delete[] a[i];
                                delete[] b[i];
                                delete[] c[i];
                }
                delete[] a;
                delete[] b;
                delete[] c;
        }

        //Pipe the RUN_TIME 2D array out into a text file.
        freopen ("MatrixMultiplicationWithBatteryPower5Iterations.txt","w",stdout);

        for(int i = 0; i < NUM_OF_TESTS; i++) // this is correct
        { /* Times ran 48 */
                for(int j = 0; j <= (NMAX - NMIN) / STEP; j++)
                { /* Times ran 1008 */
                        if(j == (NMAX - NMIN) / STEP)
                        { /* Times ran 48 */
                                cout << RUN_TIME[i][j];
                        }
                        else
                        { /* Times ran 960 */
                                cout << RUN_TIME[i][j] << ", "; //Print a row of data
                        }
                }
                cout << endl;
        }
        cout << endl;
        fclose (stdout);

        return 0;
}
```

4.) Profiler

```
import sys
from subprocess import call

newSourceName = "MatrixMultFixed.cpp"
source = "MatrixMultiplicationProject2.cpp"
curlyCounter = 0
count = 0
outputArray = []
iterator = 0


f = open(source, "r+")
newSource = open(newSourceName, 'w')

lines = f.readlines()


for i in lines:
    if "{" in i:
        curlyCounter += 1

print curlyCounter

# i is index, range(len(lines)) is the length of lines.
for i in range(len(lines)):
    # adds the Global variable that tracks times it loops through blocks
    if "int main" in lines[i]:
        lines[i - 1] = lines[i - 1].rstrip()
        lines[i - 1] = "\nlong int GLOBAL_BODY_COUNTER[" + str(curlyCounter) +
"];\n\n"

    if "{" in lines[i]:
        lines[i] = lines[i].rstrip()
        lines[i] += "GLOBAL_BODY_COUNTER[" + str(count) + "]++;\n"
#        lines[i] += "\n"
        count += 1

    if "return 0" in lines[i]:
        # inject the code needed to output to a file.
        #               freopen (       "     DataFor    source    .txt    " ,"w"
,stdout);
        lines[i] = "\n\tfreopen (" + "\"" + "DataFor" + source[:-4] +
".txt\",\"w\",stdout);\n\n"
        lines[i] += "\tcout << \"Times Ran \" << endl;\n"
        lines[i] += "\tfor(int i = 0; i < " + str(curlyCounter) + "; i++)\n"
        # cout << "Curly: " << i << ", Times Ran: " << GLOBAL_BODY_COUNTER[i] << endl;
        lines[i] += "\t\tcout << GLOBAL_BODY_COUNTER[i] << endl;\n\n"
        lines[i] += "\treturn 0;\n"
    sys.stdout.write(lines[i])  # prints the source program

for i in lines:
    newSource.write(i)  # Writes the new program to another .cpp file.

newSource.close()

# Execute C++ code from python.
#   set up the required environment for compiling and linking, then compile
call(["vcvars32.bat", "&&", "cl.exe", "/EHsc", newSourceName])
```

```
#   run resulting executable
sourceEXE = newSourceName[:-4] + ".exe"
print "This is my source program: " + sourceEXE
call([sourceEXE])

#  this code will be inputing the block comments into the original C++ program.
outputFile = open("DataFor" + source[:-4] + ".txt")
outputLines = outputFile.readlines()

for i in range(len(outputLines)):
    if i > 0:
        outputArray.append(outputLines[i].rstrip())

print outputArray

f.seek(0)
fLines = f.readlines()

for i in range(len(fLines)):
    if "{" in fLines[i]:
        fLines[i] = fLines[i].rstrip()
        fLines[i] += "/* Ran " + outputArray[iterator] + " times */\n"
        iterator += 1
    print fLines[i]

f.seek(0)
for i in range(len(fLines)):
    f.write(fLines[i])


outputFile.close()
f.close()
# already done
# Currently reads in a file, and searches through to find the '{' braces.
# Can count curlies (Works so far with macros)
# inject globalBodyCounter into source code of new C++ file.
# output numeric results of the GlobalBodyCounter into a text file.
#
# inject comments of the results into the old C++ file.
#


# need to do
# Compile new C++ file (Technically extra, could do this manually)
# Run new C++ file (Technically extra, could do this manually)
#
#
```