**CSC 180 ASSIGNMENT 3**

Matt Dunaway

Exercise 4.2. This exercise reviews the ow of execution through a program with multiple

methods. Read the following code and answer the questions below.

public class Buzz {

public static void Baffle(string blimp) {

Console.WriteLine(blimp); **5TH**

Zippo("ping", -5); **6TH, 9TH**

}

public static void Zippo(string quince, int flag) {

if ( flag < 0 ) { **2ND, 7TH**

Console.WriteLine(quince + " zoop"); **8TH**

} else {

Console.WriteLine("ik"); **3RD**

Baffle(quince); **4TH, 10TH**

Console.WriteLine("boo-wa-ha-ha"); **11TH**

}

}

public static void Main(string[] args) {

Zippo("rattle", 13); **1ST, 12TH**

}

}

1. Write the number 1 next to the first statement of this program that will be executed.

Be careful to distinguish things that are statements from things that are not.

2. Write the number 2 next to the second statement, and so on until the end of the

program. If a statement is executed more than once, it might end up with more than

one number next to it.

3. What is the value of the parameter blimp when Baffle gets invoked? **“rattle”**

4. What is the output of this program? **ik, rattle, ping zoop, boo-wa-ha-ha**

Exercise 4.5. Fermat's Last Theorem says that there are no integers a, b, and c such that

an + bn = cn

except in the case when n = 2.

Write a method named CheckFermat that takes four integers as parameters--a, b, c and

n--and that checks to see if Fermat's theorem holds. If n is greater than 2 and it turns out

to be true that an +bn = cn, the program should print \Holy smokes, Fermat was wrong!"

Otherwise the program should print “No, that doesn't work."

You should use the Math.Pow library function that takes two integers as arguments and

raises the first argument to the power of the second. For example:

int x = Math.Pow(2, 3);

would assign the value 8 to x, because 23 = 8.

using System;

class Program

{ static void CheckFermat(int a, int b, int c, int n)

{

int asq = (int) Math.Pow(a, n);

int bsq = (int) Math.Pow(b, n);

int absum = asq + bsq;

int csq = (int)Math.Pow(c, n);

{

if ((n > 2) && (csq == absum))

{Console.WriteLine("Holy smokes, Fermat was wrong!");

return;

}

}

Console.WriteLine("No that doesn't work.");

}

public static void Main()

{CheckFermat(2, 2, 4, 2);

}

}

Exercise 5.2. Many computations can be expressed concisely using the \multadd" opera-

tion, which takes three operands and computes a \* b + c. Some processors even provide

a hardware implementation of this operation for floating-point numbers.

1. Create a new program called Multadd.cs.

2. Write a method called Multadd that takes three doubles as parameters and that

returns their multadditionization.

3. Write a Main method that tests Multadd by invoking it with a few simple parameters,

like 1.0, 2.0, 3.0.

4. Also in Main, use multadd to compute the following values:

sin \_

4 + cos \_

4

2

log 10 + log 20

5. Write a method called Yikes that takes a double as a parameter and that uses

multadd to calculate

xe􀀀x +

p

1 􀀀 e􀀀x

Hint: The Math.Exp method can be used for raising e to a power.

In the last part, you get a chance to write a method that invokes a method you wrote.

Whenever you do that, it is a good idea to test the first method carefully before you start working on the second. Otherwise, you might find yourself debugging two methods at the same time, which can be difficult. One of the purposes of this exercise is to practice pattern-matching: the ability to recognize a specific problem as an instance of a general category of problems.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Multadd

{

public class Multadd

{

public static void multadd(double a, double b, double c)

{

double result = a \* b + c;

Console.WriteLine(result);

}

public static void Yikes(double x)

{

double a = x;

double b = Math.Exp(-x);

double c = Math.Sqrt(1.0 - b);

multadd(a, b, c);

}

public static void Main(String[] args)

{

Console.WriteLine("The multadditionization of 1.0, 2.0, and 3.0 is: ");

multadd(1.0, 2.0, 3.0);

Console.WriteLine();

double a1 = Math.Cos(Math.PI / 4.0);

double b1 = 1.0 / 2.0;

double c1 = Math.Sin(Math.PI / 4.0);

Console.WriteLine("sin(pi/4) + cos(pi/4)/2 = ");

multadd(a1, b1, c1);

Console.WriteLine();

double a2 = 1.0;

double b2 = Math.Log(10);

double c2 = Math.Log(20);

Console.WriteLine("log(10) + log(20) = ");

multadd(a2, b2, c2);

}

}

}