ENGG1340 / COMP2113, Assignment 1

Due Date: Oct 9, 2020 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 1.

Total marks: 100 marks

- 5 marks for proper code comments and indentation
- 95 marks for program correctness

A maximum of 5 marks will be deducted if you fail to follow the submission instructions strictly.

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 5 tasks and a tester would automatically test your submitted program. So if your submitted files and program outputs do not conform to our instructions given here, your programs cannot be evaluated and you will risk losing marks totally.

Sample test cases are provided with each task in this document. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you are able to design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your assignment submission.

Input and output format

Your C++ programs should read from the **standard input**. Also, your answer should be printed through the **standard output**. If you failed to follow this guide, the tester may not able to give a score for your program. Additionally, you should strictly follow the sample output format (including space, line breaker, etc.), otherwise, your answer might be considered as wrong.

How to use the sample test cases

Sample test cases in text file formats are made available for you to check against your work. Here's how you may use the sample test cases. Take Problem 2 test case 3 as an example. The sample input and the expected output are given in the files `input2_3.txt` and `output2_3.txt` , respectively. Suppose that your program is named "2", do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./2 < input2_3.txt > myoutput.txt
diff myoutput.txt output2_3.txt
```

Testing against the sample test cases is important to avoid making formatting mistakes. The additional test cases for grading your work will be of the same formats as the sample test cases.

Coding environment

You must make sure that your program can compile, execute and generate the required outputs on our standard environment, namely, the gcc C++11 environment we have on the CS Linux servers (academy*). Make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 [yourprogram].cpp
```

As a programmer/developer, you should always ensure that your code can work perfectly as expected on a target (e.g., your client's) environment, not only on yours.

While you may develop your work on your own environment, you should always try your program (compile & execute & check results) on our standard environment before submission.

Submission

Name your C++ programs as the following table shows and put them together into one directory. Make sure that the folder contains only these source files (`*.cpp`) and no other files. **Compress this directory as a `[uid].zip` file where [uid] is your university number** and check carefully that the correct file have been submitted. We suggest you to download your submitted file from Moodle, extract them, and check for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect files.** Resubmission after the deadline is not allowed.

Filename	Description
`1.cpp`	Problem 1
`2.cpp`	Problem 2
`3.cpp`	Problem 3
`4.cpp`	Problem 4
`5.cpp`	Problem 5

Late submission

If submit within 3 days after the deadline, 50% deduction. After that, no mark.

Evaluation

Your code will be auto-graded for technical correctness. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for logical redundancy. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Getting help

You are not alone! If you find yourself stuck on something, post your question to the course forum. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don’t know when or how to help unless you ask.

Discussion forum

Please be careful not to post spoilers. Please don’t post any code that is directly related to the assignments. However you are welcome and encouraged to discuss general ideas on the discussion forums. If you have any questions about this assignment you should post them in the discussion forums.

Problem 1: Polynomial Evaluation

Write a C++ program that evaluate a degree-n polynomial

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

with n multiplications and n additions only. Note that this is the optimal, meaning that you cannot do this evaluation with fewer arithmetic operations.

Implement the Horner's method to achieve this optimal computation.

Input:

- A single line containing $n + 3$ numbers, separated by spaces. The first number is the value of x, then the degree n of the polynomial to evaluate, followed by the $n + 1$ coefficients a_n, a_{n-1}, ..., a_0.
- The value x is a floating-point number and the numbers n, a_n, a_{n-1}, ..., a_0 are integers, and $n \geq 0$.

Output:

- A single line containing the result of the polynomial evaluation, printed as a fixed floating number with 6 decimal places.
- You may use `setprecision(n)` defined in the header `<iomanip>` to set the precision parameter of the output stream.

Requirement:

- Use the `float` data type for all floating point computations in this program.

- You can ONLY use the simple data types `char`, `bool`, `int`, `float`. In other words, you are not allowed the use other data types or data structures such as arrays, vectors, etc.

Sample Test Cases

User inputs are shown in blue.

1_1

```
-1.5 0 4
4.000000
```

1_2

```
2.43 1 4 -2
7.720000
```

1_3

```
3.289 3 -1 2 4 -6
-6.787785
```

1_4

```
-0.0001 10 11 -10 9 -8 7 -6 5 -4 3 -2 1
1.000200
```

Problem 2: A Divisor Matrix

Write a C++ program that implement the divisor matrix as follows:

Input:

- Your program should prompt the user for two positive integers `a`, `b`. (See sample test cases for the required prompt messages.) You may assume that the user will always input positive integers such that the following holds, `0 <= a < b < 1000`.

- The user will then be prompted for two divisors. You may assume that divisors entered by the user are valid, for which the following holds, `1 <= divisor < 1000`.

Output:
The program will then output a matrix with the following elements.

- The first row will start with an `M` followed by the two divisors in input order.
- The first column, beneath the `M`, will contain the integers between `a` and `b` (including `a` and excluding `b`), in increasing order.
- For the remaining two elements of each row, there will be a `1` if the integer in that row is divisible by the divisor at the head of that column, otherwise `0`.
- Matrix elements should be separated by a space. Note that there is no trailing space after the last element of a row.

Requirement:

- You can ONLY use the simple data types `char`, `bool`, `int`, `double`. In other words, you are not allowed the use other data types or data structures such as strings, arrays, vectors, etc.

Sample Test Cases

User inputs are shown in blue.

2_1:

```
a: 1
b: 10
Divisor 1: 2
Divisor 2: 3
M 2 3
1 0 0
2 1 0
3 0 1
4 1 0
5 0 0
6 1 1
7 0 0
8 1 0
9 0 1
```

2_2:

```
a: 134
b: 141
Divisor 1: 4
Divisor 2: 5
M 4 5
134 0 0
135 0 1
136 1 0
137 0 0
138 0 0
139 0 0
140 1 1
```

2_3:

```
a: 2
b: 21
Divisor 1: 2
Divisor 2: 5
M 2 5
2 1 0
3 0 0
4 1 0
5 0 1
6 1 0
7 0 0
8 1 0
9 0 0
10 1 1
11 0 0
12 1 0
13 0 0
14 1 0
15 0 1
16 1 0
17 0 0
18 1 0
19 0 0
20 1 1
```

Problem 3: Sine Function Approximation

Write a C++ program which calculates an estimation of $\sin x$ using Taylor series approximation with the formula:

$$\sin x = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1},$$

where $i! = 1 \times 2 \times 3 \times \cdots \times i$ denote the factorial of i, and 0^0 is defined as 1. The integer n governs the order of approximation and you may assume that n ranges from 0 to 50 inclusively in this question.

Input:

- a line with two numbers: a real number x ($-3 \leq x \leq 3$) followed by an integer n ($0 \leq n \leq 50$).

Output:
Your program should output the estimations of $\sin x$ as n increases. Specifically, your program should

- On the first line, print the result of $\sin x$ as returned by the predefined function `sin()` in `<cmath>` **as a fixed floating number with 15 decimal places**.
- Output the values of n and $\sin x$ on the subsequent lines as n increases. The values n and $\sin x$ on each line are separated by a space. **The value $\sin x$ is printed as a fixed floating number with 15 decimal places.**
- You may use `setprecision(n)` defined in the header `<iomanip>` to set the precision parameter of the output stream.

Requirement:

- Use only the `double` and `int` data types in your calculations.
- You can ONLY use the simple data types `char` , `bool` , `int` , `double` . In other words, you are not allowed the use other data types or data structures such as strings, arrays, vectors, etc.

Note:

- You cannot evaluate the factorial operator in a brute-force manner using multiplications only, because **overflow** can easily occur which means that the resulting value is too large to be stored in an `int` . (Consider 30! ~= 2.6525e+32 which cannot be held by even the `unsigned long long` type).
- Divisions involving some large numbers can easily run into numerical inaccuracy issues (you will learn more about this in the machine organization course).

- Hence, **you should do divisions before multiplications (whenever possible) to keep the intermediate values at each iteration of the calculation small**.
- Since different orders of arithmetic operations may entail slight difference in the final evaluation, **your results will be check against the test outputs for correctness up to the first 5 decimal points only**.

Sample Test Cases

User inputs are shown in blue.

3-1

```
0 0
sin(x) by cmath: 0.000000000000000
Taylor series approximation:
0 0.000000000000000
```

3_2

```
1 10
sin(x) by cmath: 0.841470984807897
Taylor series approximation:
0 1.000000000000000
1 0.833333333333333
2 0.841666666666667
3 0.841468253968254
4 0.841471009700176
5 0.841470984648068
6 0.841470984808658
7 0.841470984807894
8 0.841470984807897
9 0.841470984807897
10 0.841470984807897
```

3_3

```
-2.9 20
sin(x) by cmath: -0.239249329213982
Taylor series approximation:
0 -2.900000000000000
1 1.164833333333333
2 -0.544429083333334
3 -0.202169632757937
4 -0.242147438026535
5 -0.239090953096454
6 -0.239255728982749
7 -0.239249130100826
8 -0.239249334132433
9 -0.239249329115164
10 -0.239249329215629
11 -0.239249329213960
12 -0.239249329213983
13 -0.239249329213983
14 -0.239249329213983
15 -0.239249329213983
16 -0.239249329213983
17 -0.239249329213983
18 -0.239249329213983
19 -0.239249329213983
20 -0.239249329213983
```

Problem 4: Caesar Shifting

Write a C++ program which encrypts and decrypts some input characters using the Casesar Shifting algorithm detailed below.

Input:

- a line of input $s\ k\ c_1\ c_2\ c_3\ \ldots$, where
    - s is either the character `e` for encryption, or the character `d` for decryption
    - k is an integer for the number of shifts used in the Caesar shift algorithm
    - $c_1\ c_2\ c_3\ \ldots$ is a sequence of space separated characters, ended by `!`, to be encrypted or decrypted

Output:

- the encrypted/decrypted message ended by `!`. No space between two consecutive characters.

Algorithm:
To encrypt (decrypt) a letter c (within the alphabet A-Z or a-z) with a shift of k positions:

1. Let x be c's position in the alphabet (0 based), e.g., position of `B` is 1 and position of `g` is 6.
2. For encryption, calculate $y = x + k$ modulo 26;
   for decryption, calculate $y = x - k$ modulo 26.
3. Let w be the letter corresponding to position y in the alphabet. If c is in uppercase, the encrypted (decrypted) letter is w in lowercase; otherwise, the encrypted (decrypted) letter is w in uppercase.

A character which is not within the alphabet A-Z or a-z will remain unchanged under encryption or decryption.

Example. Given letter `B` and k = 3, we have x = 1, y = 1 + 3 mod 26 = 4, and w = `E`. As `B` is in uppercase, the encrypted letter is `e`.

Requirement:

- Implement a function `CasesarShift()` which takes in a `char c` and an `int k`, where `c` is the character to undergo Caesar Shifting and `k` is the number of positions (can be negative) to shift, and return the processed `char` after Caesar Shifting. The function prototype is given by: `char CaesarShift(char c, int k)`
- You can ONLY use the simple data types `char`, `bool`, `int`, `double`. In other words, you are not allowed the use other data types or data structures such as strings, arrays, vectors, etc.

Sample Test Cases

User inputs are shown in blue.

4_1

```
e 1 !
!
```

4_2

```
e 3 a B c D e !
DeFgH!
```

4_3

```
d 3 D e F g H !
aBcDe!
```

4_4

```
e -1 H e l l o   E N G G 1 3 4 0 / C O M P 2 1 1 3 !
gDKKNdmff1340/bnlo2113!
```

4_5

```
d 10 n 3 V 3 D 3 N _ M Y N 3 _ S C _ N 3 L E Q Q 3 N _ M Y N 3 !
D3l3t3d_cod3_is_d3bugg3d_cod3!
```

Problem 5: Bounding Boxes

Write a C++ program to compute a **minimum-sized** axis-aligned bounding box (AABB) for a set of input 2D geometries. An AABB is a rectangle with sides parallel to the x-, y-axes which encloses some given geometries. The input and output of your program are as follows:

Input: Each line of the user input begins with a character indicating the type of geometry, followed by some parameters of the geometric object. The input line can be one of the followings:

- `R` $x\ y\ width\ height$
  where `R` represents an input rectangle, x, y are floating-point numbers for the x-, y-coordinates of the rectangle center, $width$ and $height$ are floating-point numbers for the rectangle size along the x- and y-axes, respectively.
- `C` $x\ y\ radius$
  where `C` represents an input circle, x, y are floating-point numbers for the x-, y-coordinates of the circle center, and $radius$ is a floating-point number for the radius of the center
- `P` $n\ x_1\ y_1\ x_2\ y_2\ \ldots\ x_n\ y_n$
  where `P` represents an input point set, n is an integer indicating the number of points in the set, and $x_i, y_i, i = 1, \ldots, n$ are floating point numbers for the x-, y-coordinates of the n points
- `#`
  indicates end of input

Output:

- A single line
  $x\ y\ width\ height$
  where x and y are floating-point numbers for the x-, y-coordinates of the center of the minimum-sized AABB, $width$ and $height$ are floating-point numbers giving the sizes of the AABB along the x-, y-axes, respectively.

Requirement:

- Use the `double` data type for floating point calculations.
- Implement the functions `RectangleBB()` , `CircleBB()` and `PointSetBB()` which take user inputs and calculate the corresponding bounding boxes of the different geometries (rectangles, circles & point sets). The function prototypes are:
  `void RectangleBB(double &xmin, double &xmax, double &ymin, double &ymax)`
  `void CircleBB(double &xmin, double &xmax, double &ymin, double &ymax)`
  `void PointSetBB(double &xmin, double &xmax, double &ymin, double &ymax)`
  where `xmin` , `xmax` , `ymin` , `ymax` are the calculated extends of the bounding boxes for the geometry. Note that these parameters are pass-by-reference, since they are the computation results with we would like the calling function to be able to access after function termination. By using pass-by-reference, the functions would be able to modify the actual arguments passed in from the calling function.
- You can ONLY use the simple data types `char` , `bool` , `int` , `double` . In other words, you are not allowed the use other data types or data structures such as strings, arrays, vectors, etc.

Note:

- **The questions assume no bound for the floating-point parameters (x, y, $width$, $height$, $radius$) of the input geometries and therefore they can be any values that a double data type can hold.** You may use `std::numeric_limits<double>::lowest()` and `std::numeric_limits<double>::max()` defined in the header `<limits>` in your program to obtain the smallest and the largest possible values, respectively, for the double data type.

Sample Test Cases

User inputs are shown in blue.

5_1

```
R 0 0 3 2
#
0 0 3 2
```

5_2

```
C -0.5 3.2 1.6
#
-0.5 3.2 3.2 3.2
```

5_3

```
P 2 3 -2 -1 4
#
1 1 4 6
```

5_4

```
P 3 -1.5 3 3 3 5 3
#
1.75 3 6.5 0
```

5_5

```
P 2 3 -2 -1 4
C -0.5 3.2 1.6
P 3 -1.5 3 3 3 5 3
R 0 5.75 3 2
#
1.45 2.375 7.1 8.75
```