ENGG1340 Programming Technologies / COMP2113 Computer Programming II Assignment 2

Deadline: 16 March (Saturday), 2024 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 2.

Total marks: 100 marks

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 3 tasks and a tester would automatically test your submitted program. If your submitted files and program outputs do not conform to the instructions given here, your programs cannot be evaluated, and you will risk losing marks totally.

Sample test cases are provided with task 1 and 2 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 can 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

For tasks that require **reading input from user**, your answer should read from the **standard input**. Also, your program should **print** through the **standard output**. If you failed to follow this guide, the tester may not be able to give a score for your program. Additionally, you should strictly follow the sample output format, 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 1 test case 1 as an example. The sample input and the expected output are given in the files `input1_1.txt` and `output1_1.txt`, respectively. Suppose that you compiled your program to an executable named `1`, do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./1 < input1_1.txt > myoutput.txt
diff -Bw myoutput.txt output1_1.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. Note that for this assignment, the flag `-Bw` will be used when checking the output of your programs. This will ignore leading/trailing spaces and empty lines when comparing outputs.

Coding environment

You must ensure that your program can compile (for C++ programs only), execute and generate the required outputs on our standard environment, i.e., the CS Linux servers (`academy*`).

For C++ programs, they must be compilable with the gcc C++11 environment on our standard environment. The compilation flags `-pedantic-errors -std=c++11` must be used. For example for task 1, the following compilation command should be used to compile your programs:

```
g++ -pedantic-errors -std=c++11 -o [executable name] [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. If your submitted files cannot be compiled using the compilation method stated above on our standard environment, or if they cannot be executed on our standard environment, your files may not be graded.

Submission

Name your files as the following table shows and put them together into one directory. Make sure that the folder contains only these source files and no other files. **Compress this directory as a `[uid].zip` file where [uid] is your university number** and check carefully that the correct files have been submitted.

You are advised to prepare the zip file on the standard environment, then transfer the zip file to your own machine through FTP and submit it on Moodle. You should not transfer content of your files through copy-and-paste as the process is error prone. 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/incomplete files.** Resubmission after the deadline is not allowed.

Filename	Description
`1.cpp`	Problem 1
`2.cpp`	Problem 2
`Makefile`	Problem 3

Late submission

If submit within 3 days after the deadline, there will be 50% mark 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.

Use of generative AI tools, like ChatGPT, is not allowed for the assignment.

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: Affine Cipher

Write a C++ program which encrypts and decrypts some input characters using Affine Cipher.

Input:

- The program should read in a line of inputs `s a b c1 c2 c3 ...`, where
  - `s` is either the character `e` for encryption, or the character `d` for decryption.
  - `a` and `b` are the keys of the cipher.
  - `c1 c2 c3 ...` is a sequence of space separated characters, ended by `!`, to be encrypted or decrypted.

Output:

- The program should output the encrypted or decrypted message. Refer to the sample outputs for the exact format.

Algorithm:

- To encrypt a letter `c` (within the alphabet A-Z or a-z) with the input key `a` and `b`:

  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. Calculate $y = (ax + b) \bmod 26$.
  3. Let w be the letter corresponding to position y in the alphabet. If c is in uppercase, the encrypted letter is w in lowercase; otherwise, the encrypted letter is w in uppercase.

- To decrypt a letter `c`, we may reverse the encryption process as follow. However, it requires the understanding of modular multiplicative inverse.

  1. Let y be `c`'s position in the alphabet (0 based).
  2. Find the modular multiplicative inverse a^{-1} such that $aa^{-1} = 1 \bmod 26$, then calculate $x = (a^{-1}(y - b)) \bmod 26$.
  3. Let w be the letter corresponding to position x in the alphabet. If c is in uppercase, the decrypted letter is w in lowercase; otherwise, the decrypted letter is w in uppercase.

- Instead of decrypting the letter through calculation, we can apply encryption to all possible letter positions to find the letter that encrypts to `c`:

  1. Let y be `c`'s position in the alphabet (0 based).
  2. Find the value of x such that $y = (ax + b) \bmod 26$. Note that $0 \leq x < 26$.
  3. Let w be the letter corresponding to position x in the alphabet. If c is in uppercase, the decrypted letter is w in lowercase; otherwise, the decrypted letter is w in uppercase.

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

Assumptions:

- Input key `a` will always be a valid key such that decryption is possible.
- Input key `b` can be any integer.

Note:

- Note that the modulo operator `%` in C++ is different from the $\bmod$ notation, where `%` operator may produce negative results.

Sample Test Cases

1_1

Input:

```
e 1 1 !
```

Output:

```
!
```

1_2

Input:

```
e 1 1 a B c D e !
```

Output:

```
BcDeF!
```

1_3

Input:

```
d 1 1 D e F g H !
```

Output:

```
cDeFg!
```

1_4

Input:

```
e 5 8 A f f i n e C i p h e r !
```

Output:

```
iHHWVCsWFRCP!
```

1_5

Input:

```
e 3 -2 H e l l o E N G G 1 3 4 0 !
```

Output:

```
tKFFOklqq1340!
```

1_6

Input:

```
d 7 0 v 3 Z 3 D 3 V _ O U V 3 _ E W _ V 3 H K Q Q 3 V _ O U V 3 !
```

Output:

```
D3l3t3d_cod3_is_d3bugg3d_cod3!
```

Problem 2: Bounding Box

Write a C++ program that provides the function `mergeBoundingBoxes()` that merges two bounding boxes. For this question, the screen coordinates are used. I.e., x-coordinates go from left to right, and y-coordinates go from top to bottom.

Do not include the `main()` function in your program. Please read the following specifications carefully.

Input:

- Your program must include the provided header file `bounding.h`, and provides the definition of the function `mergeBoundingBoxes()`. The function takes 8 input parameters:
  - `x1`, `y1` represents the coordinates of the top-left corner of the first bounding box. The two parameters must be passed by reference and be used as the output parameters also.
  - `w1`, `h1` are the width and height of the first bounding box. The two parameters must be passed by reference and be used as the output parameters also.
  - `x2`, `y2` represents the coordinate of the top-left corner of the second bounding box.
  - `w2`, `h2` are the width and height of the second bounding box.

Output:

- The function returns `true` if the two bounding boxes overlap with each other, otherwise return `false`. Two bounding boxes are considered overlapping if the overlapping area is greater than 0. (i.e., two touching boxes are not considered overlapping).
- Output parameters `x1`, `y1`, `w1`, `h1` must be updated with the coordinates of the top-left corner, as well as the width and height of the merged bounding box.

Assumptions:

- The input widths and heights of the bounding boxes will always be positive integers.

Testing and compilation:

- Your program will be tested with tester programs. These tester programs will be compiled together with your `2.cpp` for testing. For example the tester `main2_1.cpp` has the following `main()` function (slightly modified for formatting).

```
int main() {
  int x1 = 1, y1 = 1, w1 = 10, h1 = 10;
  int x2 = 5, y2 = 5, w2 = 10, h2 = 10;
  mergeBoundingBoxes(x1, y1, w1, h1, x2, y2, w2, h2);
  cout << x1 << " " << y1 << " " << w1 << " " << h1 << endl;
}
```

The tester program will be compiled with the following command:

```
g++ -pedantic-errors -std=c++11 -o main2_1 main2_1.cpp 2.cpp
```

The corresponding sample output when running `./main2_1` could be found in `output2_1.txt`. You are expected to use the diff-test as explained in the section "How to use the sample test cases" above to check the output.

Do not include any `main()` function in `2.cpp`. We may not be able to grade your work if you do so. Instead, please implement your own tester programs for testing.

Sample Test Cases

2_1

Tester program: `main2_1.cpp` (See above)

Output:

```
1 1 14 14
```

2_2

Tester program: `main2_2.cpp`

```
int main() {
  int x1 = 1, y1 = 1, w1 = 10, h1 = 10;
  int x2 = 5, y2 = 5, w2 = 10, h2 = 10;
  bool result = mergeBoundingBoxes(x1, y1, w1, h1, x2, y2, w2, h2);
  if(result) {
    cout << "Overlaps" << endl;
  } else {
    cout << "Does not overlap" << endl;
  }
}
```

Output:

```
Overlaps
```

2_3

Tester program: `main2_3.cpp`

```
int main() {
  int x1 = 10, y1 = -10, w1 = 10, h1 = 20;
  int x2 = -10, y2 = 10, w2 = 50, h2 = 10;
  mergeBoundingBoxes(x1, y1, w1, h1, x2, y2, w2, h2);
  cout << x1 << " " << y1 << " " << w1 << " " << h1 << endl;
}
```

Output:

```
-10 -10 50 30
```

2_4

Tester program: `main2_4.cpp`

```
int main() {
  int x1 = 10, y1 = -10, w1 = 10, h1 = 20;
  int x2 = -10, y2 = 10, w2 = 50, h2 = 10;
  bool result = mergeBoundingBoxes(x1, y1, w1, h1, x2, y2, w2, h2);
  if(result) {
    cout << "Overlaps" << endl;
  } else {
    cout << "Does not overlap" << endl;
  }
}
```

Output:

```
Does not overlap
```

Problem 3: Monte Carlo Simulation

The provided files `random.h`, `random.cpp`, and `main3.cpp` implements the Monte Carlo simulation for finding π using 1,000,000 random points. These files are designed to be compiled using the separated compilation method. Prepare a **Makefile** for the compilation and testing of the program.

Requirements:

- Your Makefile must provide the following targets.
  - `random.o`: Creates the object file `random.o` from `random.cpp`. It depends on `random.h` and `random.cpp`.
  - `main3.o`: Creates the object file `main3.o` from `main3.cpp`, It depends on `random.h` and `main3.cpp`.
  - `main3`: Creates the executable `main3` from `random.o` and `main3.o`. It depends on `random.o` and `main3.o`.
  - `test`: Perform a test by executing the executable `main3`. It depends on `main3`.
  - `clean`: Perform cleaning by removing `random.o`, `main3.o`, and `main3` from the current folder.
- Targets `random.o`, `main3.o` and `main3` must only re-create the target file if necessary.
- Targets `test` and `clean` must always be executed even if the current folder has a file or folder named `test` or `clean`.

Note:

- There are no test cases available for this question. You must test your Makefile by setting up different scenarios. For example, if you run `make main3`, then `touch random.cpp`, running `make main3` again will re-create `random.o` and `main3` because of the dependencies.
- File format of a Makefile is essential. Your Makefile will not be graded if it cannot be used in the standard environment due to formatting problems (e.g., missing TAB separators etc.).