ENGG1340 Programming Technologies / COMP2113 Computer Programming II Assignment 2

Deadline: 22 March (Saturday), 2025 23:59

General Instructions

Submit your assignment via VPL on Moodle. Ensure that your program can execute, and generate the required outputs in VPL. Work incompatible with the VPL may not be marked.

For C++ programs, ensure compilation with the gcc C++11 standard on our standard environment. This is already enforced in VPL. If you are testing in your own environment, use the following command to compile your programs:

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

As a developer, ensure that your code works flawlessly in the intended environment, not just your own. While you may develop your work in your own environment, always test your program in our standard environment before submission.

Evaluation

For tasks requiring **user input**, utilize the **standard input**. Likewise, your program should **output/print** through the **standard output**. Strict adherence to the sample output format is required, or your answer may be marked incorrect.

Your code will be automatically graded for technical correctness. Essentially, we use test cases to evaluate your solution, failure to pass any of the test cases may result in zero marks. Partial credits are generally not given for incomplete solutions as it may be challenging to objectively assess incomplete program logic. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Additional test case will be used during grading. Scoring full marks on VPL does not ensure full marks in the assignment. Sample test cases may or may not encompass all boundary cases. Designing proper test cases to verify your program's accuracy is part of the assessment.

Academic dishonesty

Your code will be cross-checked with other submissions and online sources for logical duplication. Note that providing your work to others, aiding others in copying, or copying from others will be considered plagiarism, and will be dealt with as per departmental policy. Please refer to the course information notes for more details.

Use of generative AI tools, like ChatGPT, is not permitted for all assignment.

Getting help

You are not alone! If you are stuck, post your query on the course forum. This assignment should be educational and rewarding, not frustrating. We are here to help, but we can only do so if you reach out.

Please avoid spoilers on the discussion forum. Do not post any code directly related to the assignments. You are, however, encouraged to discuss general concepts on the forums.

Submission

Deadlines are strictly enforced. Resubmission beyond the submission period will not be accepted.

Late Policy:

- If you submit within 2 days after the deadline, 30% deduction.
- If you submit within 3-5 days after the deadline, 50% deduction.
- After that, no mark.

Problem 1: Squarefree semiprimes

A semiprime is a natural number that is the product of two prime numbers. A squarefree semiprime is a semiprime that is not a perfect square. Write a C++ program, `1.cpp`, that finds the smallest squarefree semiprime that is greater than or equals to a given integer.

Input:

- An integer n where $1 \leq n \leq 10000000$. The input will always be an integer.

Output:

- The program should output three numbers: a, b, and s, such that $a < b$ and $s = a \times b$. Here, s is the smallest squarefree semiprime that is greater than or equals to n.

Sample Test Cases

1_1

Input: `1`

Output: `2 3 6`

1_2

Input: `7`

Output: `2 5 10`

1_3

Input: `1000`

Output: `17 59 1003`

Problem 2: Find home

Write a C++ program, `2.cpp`, that implements four functions, `returnToHome(...)`, `readInput(...)`, `move(...)`, and `findHome(...)`, declared in the provided header file `turtle.h`.

Function `r2d(...)` and `d2r(...)` are utility functions provided in `utils.cpp`, and the main routine is provided in `main.cpp`. Your `2.cpp` must be compatible with these provided files. Read the following function specifications carefully.

Your program will be compiled using the following command. This is automatically done in VPL.

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

The main function in `main.cpp` tests the four functions and handles the outputs. Examine `main.cpp` to understand how it works.

Do not include any `main()` function in `2.cpp`. We may not be able to grade your work if you do so. If you are working in your own environment, you may implement your own tester programs (i.e., create an alternative `main.cpp`) for testing.

Function Details

For all four functions, parameters passed by reference are output parameters. The function should output a suitable value to these parameter according to the specification.

- `void returnToHome(double & x, double & y)`: The function sets the position of an object to home, which is at the coordinates $(0, 0)$. `x` and `y` represent the x and y coordinates of the object. The x-axis goes from West to East and the y-axis goes from South to North. The function sets `x` and `y` to 0.
- `bool readInput(double & distance, double & direction)`: The function first reads the `distance` value from user input. If the `distance` value is zero or less, the function should return `false`. If `distance` is greater than zero, the function then reads a `direction` from user input and adjust it to a value within the range of $[0, 360)$ degrees (i.e., $0 <= direction < 360$). For example, if the input `direction` is -30, the function should output `direction` as 330; Similarly, if the input `direction` is 361, the function should output `direction` as 1. The function outputs the `distance` and `direction` and returns `true` in such case.
- `void move(double & x, double & y, double distance, int direction)`: This function moves an object located at (x, y) for a specific `distance` in a specific `direction`, and outputs the new position, `x` and `y`, of the object.
- `void findHome(double x, double y, double & distance, int & direction)`: This function calculates the `distance` and `direction` needed for an object at (x, y) to return to home, and outputs these values. `distance` must be non-negative, and `direction` must be within the range of $[0, 360)$.

Tips

- For an object moving from (x, y) for distance d at angle $theta$ from North reaching (x', y'):
  $x' = x + d \cdot sin(\theta)$
  $y' = y + d \cdot cos(\theta)$
  $\theta = arctan2(x' - x, y' - y)$
  $d = \sqrt{(x' - x)^2 + (y' - y)^2}$
  Note: $arctan2$ is an alternative form of $arctan$ that avoids division by zero.
  Reference: Converting between polar and Cartesian coordinates
- You need to include the `#include <cmath>` header to use math functions in C++.
- Angles in trigonometry functions in C++ uses radians. Use the functions `d2r(...)` and `r2d(...)` provided in `utils.cpp` for conversion.

Inputs and Outputs

The main function in `main.cpp` manages all inputs via the `readInput(...)` functions, as well as all outputs. It continuously reads the `distance` and `direction` input from user until a negative `distance` value is entered. For each pair of `distance` and `direction` values, it outputs the values entered, followed by the location after movement. Eventually, it outputs the `distance` and `direction` needed for the object to return to home.

Sample Test Cases

2_1

Input: `100 0 -1`

Output:

```
100.0 @ 0.0 -> 0.0, 100.0
100.0 @ 180.0 -> Home
```

Explanation

Input:

- Move the object for 100 units North (`100 0`);
- Return home (`-1`)

Output:

- Move the object for 100 units at 0 degree North (`100.0 @ 0.0`), the object is moved to $(0, 100)$ (` -> 0.0, 100.0`).
- The object needs to move 100 units at 180 degree North to return to home (`100.0 @ 180.0 -> Home`).

2_2

Input: `50 90 100 180 -1`

Output:

```
50.0 @ 90.0 -> 50.0, 0.0
100.0 @ 180.0 -> 50.0, -100.0
111.8 @ 333.4 -> Home
```

2_3

Input: `30 120 30 450 30 15 -1`

Output:

```
30.0 @ 120.0 -> 26.0, -15.0
30.0 @ 90.0 -> 56.0, -15.0
30.0 @ 15.0 -> 63.7, 14.0
65.3 @ 257.6 -> Home
```

Problem 3: Monte Carlo Simulation

The provided files `random.h`, `random.cpp`, and `main.cpp` implements the Monte Carlo simulation for calculating the value of π using n random points, where n is provided by the user. 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 should provide the following targets:
    - `random.o`: This target generates the object file `random.o` from `random.cpp`. It depends on `random.h` and `random.cpp`.
    - `main.o`: This target generates the object file `main.o` from `main.cpp`. It depends on `random.h` and `main.cpp`.
    - `main`: This target generates the executable `main` from `random.o` and `main.o`. It depends on `random.o` and `main.o`.
    - `run`: This target runs a test by executing the `main` executable. It generates the executable `main` if necessary.
    - `autotest`: This target runs a routine that test the `main` executable with inputs 100000, 200000, and 500000 automatically.
    - `clean`: This target cleans up the current folder by removing `random.o`, `main.o`, and `main`.
- The targets `random.o`, `main.o` and `main` should only recreate the target file when necessary.
- The targets `run`, `autotest`, and `clean` should always be executed, even if there is a file or folder named `run`, `autotest`, or `clean` in the current folder.

Note:

- The file format of a Makefile is crucial. However, VPL coverts tabs to spaces while editing. To address this, the VPL for this problem is set up to convert all leading spaces to tabs, allowing you to edit the Makefile directly in VPL.
- The VPL for this problem tests your Makefile based on different scenario. For example, it checks if `main.o` will be rebuilt when `main.cpp` is updated. More scenario will be set up during grading. As the "Run" feature in VPL cannot test scenarios like these, you should test your Makefile in your own environment to ensure that it meets all requirements.