COMP2113 Programming Technologies / ENGG1340 Computer Programming II
Assignment 3
Deadline: 21 November (Thursday) 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 3.

Total marks: 100 marks

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. 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.

Coding environment

You must make sure that your C++ 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 [program_name].cpp -o [object_code_name]
```

The name after the "-o" option is the name of the object code. In the above example, you can run the program using the following command:

```
./[object_code_name]
```

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

Submit your C++ program files on the VPL. **You need to use VPL to test your code.** Resubmission after the deadline is not allowed.

Late submission

If you submit within 2 days after the deadline, there will be 30% mark deduction. If you submit within 3 to 5 days after the deadline, there will be 50% mark deduction. After that, no mark will be given.

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 questions to the course forum, send us emails or come to our support sessions. 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 [30%]:

Write a C++ program that reads a line of text containing only alphabetical characters (both uppercase and lowercase) and performs both a character frequency analysis and a unique character sequence extraction based on the following instructions.

Your program should:
1. Count the occurrences of each character (case-sensitive) in the input and output each character's frequency in alphabetical order by ASCII values.
2. Generate a new sequence that contains only the first occurrence of each character in the original text, preserving the order of their initial appearance.

Input Format
The program reads a single line of text from standard input. The line will contain only letters (a-z, A-Z) with a maximum length of 200 characters.

Output Format
1. Character Frequency Table: Display each unique character and its frequency on a separate line. The characters should be listed in alphabetical order according to ASCII values.
2. Unique Character Sequence: After the frequency table, output a string that contains each character only once, in the order of its first appearance in the input text.

Requirements
We have declared some functions for you. You may modify their inputs or outputs as needed to complete your program. You can add some other functions if necessary.

```
void countFrequencies(const string &input, map<char, int> &freqMap);
string getUniqueSequence(const string &input);
void printFrequencies(const map<char, int> &freqMap);
```

Sample Test Cases
User inputs are shown in blue.

1_1:
```
aAbBcCA
A: 2
B: 1
C: 1
a: 1
b: 1
c: 1
aAbBcC
```

1_2:
```
HelloWorld
H: 1
W: 1
d: 1
e: 1
l: 3
```

```
o: 2
r: 1
HeloWrd
```

Problem 2 [35%]:

Write a C++ program that determines whether a path exists from the starting point to the destination in a given maze. The maze is represented by a 2D grid where each cell can either be passable or an obstacle. The program should find the unique shortest path, if it exists, and output the sequence of coordinates along this path.
Your program should:
1. Read the maze dimensions, layout, and start/end coordinates from standard input.
2. Check if there exists a path from the starting point to the destination.
3. Output the sequence of coordinates along the shortest path if one is found; otherwise, output “No path found.”

Input:
The program reads from standard input with the following structure:
1. The first line contains two integers rows and cols, representing the dimensions of the maze.
2. The next rows lines contain cols integers (either 0 or 1), representing the maze layout:
   - 1 indicates a passable cell.
   - 0 indicates an obstacle.
3. The final line contains four integers: startRow, startCol, endRow, and endCol, specifying the starting and destination coordinates.

Output:
1. If a path is found, output each coordinate along the shortest path, starting from the starting point and ending at the destination, in the format (row, col).
2. If no path exists, output “No path found.”

Assumptions:
1. Valid Input: Assume that the starting and destination coordinates are valid cells within the maze bounds.
2. Movement: The program can only move in four directions: up, down, left, and right.

Requirements
We have declared some functions for you. You may modify their inputs or outputs as needed to complete your program. You can add some other functions if necessary.

```
// Struct Point represents a point in the maze (row and column coordinates)
struct Point {
    int row, col;
};

// Checks if the given position is within the maze bounds and passable
// Parameters: current row, current column, total maze rows, total maze
// columns, maze layout
// Returns: boolean indicating whether the position is valid and passable
bool isValid(int currentRow, int currentCol, int mazeRows, int mazeCols,
const vector<vector<int>>& maze);

// Outputs the path from the starting point to the destination by tracing
// back using the parent map
// Parameters: parent map records the previous point for each point in the
// path, destination point
void printPath(const vector<vector<Point>>& parent, Point destination);

// Initializes the visited markers and parent tracking arrays for path
// tracing in the maze
// Parameters: total maze rows, total maze columns, visited array, parent
// array
// Note: `parent` is used to record the previous point of each point, enabling
path reconstruction
void initializeMazeData(int mazeRows, int mazeCols, vector<vector<bool>>&
visited, vector<vector<Point>>& parent);

// Searches for a path from start to destination and updates the parent map
// to trace the path
// Parameters: maze layout, starting point, destination point
// Returns: boolean indicating whether a path was found
bool PathSearch(const vector<vector<int>>& maze, Point start, Point
destination);

// Reads maze dimensions, layout, and start/end points from input
// Parameters: total maze rows, total maze columns, maze layout, starting
// point, destination point
void readInput(int& mazeRows, int& mazeCols, vector<vector<int>>& maze,
Point& start, Point& destination);
```

Sample Test Cases

User inputs are shown in blue.

2_1:

```
3 4
1 1 0 1
0 1 1 1
1 0 1 1
0 0 2 3
(0, 0)
(0, 1)
(1, 1)
(1, 2)
(2, 2)
(2, 3)
```

2_2:

```
4 4
1 0 0 1
1 1 0 0
0 1 1 1
1 1 1 1
0 0 3 3
(0, 0)
(1, 0)
(1, 1)
(2, 1)
(3, 1)
```

```
(3, 2)
(3, 3)
```

2_3:

```
3 3
1 0 0
0 0 1
1 1 1
0 0 2 2
No path found
```

Problem 3 [35%]:

Write a C++ program to manage a list of orders using a linked list. Each order has a unique ID and a quantity. Your program should support the following operations:
1. Insert: Insert an order into the linked list in ascending order by ID.
2. Delete: Delete an order by its ID.
3. Print: Print all orders in the linked list, showing each order's ID and quantity in ascending order by ID.

Functionality:
The program should:
1. Accept commands to manage the order list interactively from standard input.
2. Process each command to modify or display the linked list.
3. Maintain the list in ascending order by ID at all times.

Input:
The program reads a series of commands from standard input. Each command represents one operation on the order list and follows one of these formats:

1. Insert: insert <ID> <quantity>
   - <ID> is a positive integer representing the order ID
   - <quantity> is a positive integer representing the quantity of the order.
   - If an order with the same ID already exists, ignore the insert command.
2. Delete: delete <ID>
   - <ID> is the ID of the order to delete. If the ID does not exist in the list, ignore the command.
3. Print: print
   - Prints the entire list of orders in ascending order by ID. Each order should be printed on a new line in the format <ID>: <quantity>. If the list is empty, output "Empty list."

Output:
1. For the print command, output each order in the linked list, with each order on a new line in the format:
   `<ID>: <quantity>`
2. If the list is empty when print is called, output:
   `Empty list`

Assumptions:
1. Unique IDs: Each order has a unique ID. If an insert command is issued with an ID that already exists, the program should ignore it.
2. Order Management: Orders should always be stored in ascending order by ID within the linked list.
3. Memory Management: Use dynamic memory allocation for the linked list nodes, and ensure proper memory management, particularly for insertion and deletion operations.

Requirements
We have declared some functions for you. You may modify their inputs or outputs as needed to complete your program. You can add some other functions if necessary.

```
struct Order {
    int id;
    int quantity;
    Order* next;
    Order(int id, int quantity) : id(id), quantity(quantity),
          next(nullptr) {}
};
```

```
class OrderList {
public:
    OrderList();                                  // Constructor
    void insert(int id, int quantity); // Insert order in ascending order
                                       // by ID
    void deleteOrder(int id);          // Delete order by ID
    void print() const;                // Print all orders in ascending
                                       // order by ID
    ~OrderList();                      // Destructor

private:
    Order* head;
    bool exists(int id) const;         // Check if an order with the given
                                       // ID exists
};
```

Sample Test Cases
User inputs are shown in blue.

3_1:

```
insert 101 5
insert 202 10
insert 150 3
print
delete 150
print
101: 5
150: 3
202: 10
101: 5
202: 10
```

3_2:

```
insert 300 15
insert 200 8
insert 400 10
print
delete 200
delete 500
print
200: 8
300: 15
400: 10
300: 15
400: 10
```

2_3:

```
print
```

```
insert 500 20
insert 100 5
insert 300 10
print
delete 100
delete 700
insert 200 8
print
Empty list
100: 5
300: 10
500: 20
200: 8
300: 10
500: 20
```

If you face any problems, **please feel free to contact us or join our STA support session! We are very happy to help you!**
We wish you enjoy this assignment! ☺