ENGG1340 Programming Technologies / COMP2113 Computer Programming II

Assignment 3

Deadline: 21 April (Monday), 2025 23:59

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

> - [ ] Problem 1 (30 marks)
> - [ ] Problem 2 (35 marks)
> - [ ] Problem 3 (35 marks)

Total marks: 100 marks

General Instructions

Please read the instructions in this document carefully.

In this assignment, you will solve 3 tasks, and a tester will automatically test your submitted program. Therefore, 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.

One sample case is provided with every 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.

Coding environment

For all the problems, make sure the following compilation command is used to compile your program:

```
g++ -pedantic-errors -std=c++11 [your_program_name].cpp -o vpl_execution
```

Submission

Name your programs as the following table shows, and submit them to the corresponding VPL tester. **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

Late submission

If you submit within 3 days after the deadline, there will be a 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

Please write a C++ program that meets the following requirements:

1. Read a whole line of text from standard I/O.
2. Given that this line contains one or more words separated by whitespaces, keep the order of words, but reverse the letter order within each word.
3. Print the modified sequence.
4. Print the word with the most letters (if there are multiple, print the last one that appears).
5. Print the middle letter(s) of the reversed longest word. If the length of the longest word is even, print the two middle letters.

Assumptions:

- The line only contains letters from the ASCII character set.
- The line contains at least one word, and its length is less than 100.
- The number and positions of whitespaces are not restricted and can appear anywhere.
  - If there are no whitespaces in the line, the whole line should be analyzed.
- All whitespaces should be kept in the sequence.
- No punctuations will appear in the sequence.

Example Input:

```
The quick brown   fox jumped over the lazy dog
```

Example Output:

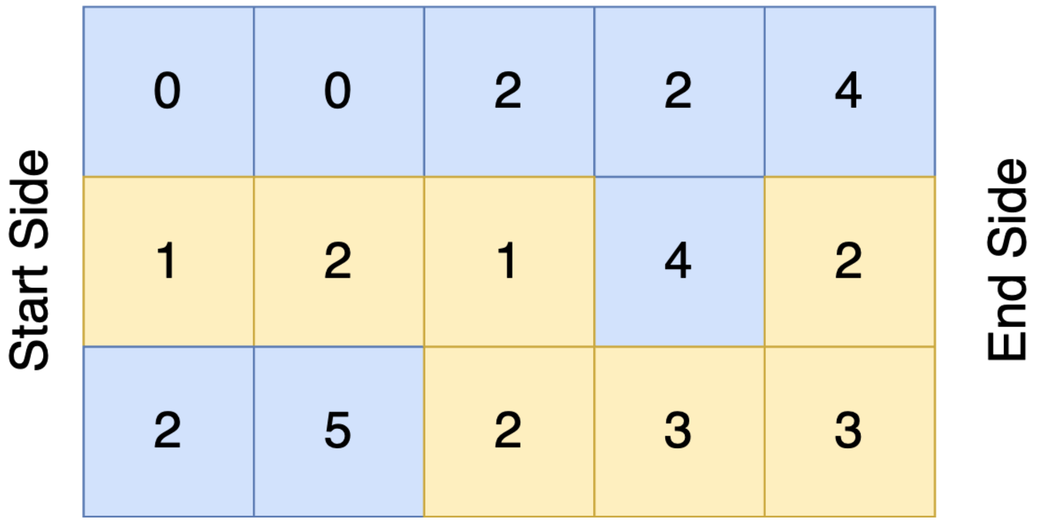
```
ehT kciuq nworb   xof depmuj revo eht yzal god
depmuj
p
m

```

Notes: There's a `\n` after the `m` is printed.

Problem 2

You are participating in a parkour competition where you need to overcome a series of obstacles to successfully complete the race. The scenario can be modeled with the following grids: you start on the left side and aim to reach the right side. The entire area consists of a 3x5 grid, where each cell represents an obstacle with a height (in meters). You can climb up to 1.5 meters and jump down up to 2.5 meters to overcome obstacles. You are only allowed to move up, down, left, or right, and cannot move diagonally.

1. Based on the given obstacle heights, determine if you can successfully complete the race.
2. If you can complete the race, assume that climbing 1 meter consumes 1 unit of energy, while jumping down does not consume any energy. Print the minimum energy consumed.



Please write a C++ program to determine if you can complete the race.

- The program will read a line (i.e. the path of a txt file) from the standard I/O.
- The program will read the content of this file, containing the shape of grid array and the status of each cell.
- If you can complete the race, the program will return 1 and the minimum energy consumed. Otherwise, it will return 0.

Assumptions:

- The first line consists of two integers (i.e., the number of rows and columns respectively) to represent the array shape of the grid. Both integers are less than 10.
- The following lines consist of integer sequences to denote the height of each cell.
- All the numbers in the same line are separated by a whitespace.

Sample:

- Take the above figure as an example. The input file should be

  `3 5`

```
0 0 2 2 4
1 2 1 4 2
2 5 2 3 3
```

- You can pass the race along the green path, and the minimum energy consumed is 3. Therefore, your program should return

```
1
4

```

Problem 3

Write a C++ program to manage course schedules. Each course has a unique ID and a time slot (start time and end time). The program should support the following operations:

1. **Insert**: Insert a course into the linked list in ascending order by start time. Each time a course is added, automatically check for time conflicts. If there is a conflict, output `false` and refuse to add the course.
2. **Delete**: Delete a course by its ID.
3. **Update**: Update the time slot of a course based on its ID. If the new time slot causes a conflict with existing courses, return `false` and refuse to update.
4. **Print**: Print all courses in the linked list, showing each course's ID and time slot in ascending order by start time.

Requirements:

1. Accept commands interactively from standard input to manage the course list.
2. Process each command to modify or display the linked list.
3. Maintain the linked list in ascending order by start time at all times.
4. Use dynamic memory allocation for the linked list nodes and ensure proper memory management, especially for insertion, deletion, and update operations.

Functions:

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

1. **Insert**: `insert <ID> <start_time> <end_time>`

  - `<ID>` is a positive integer representing the unique ID of the course.
  - `<start_time>` and `<end_time>` are integers representing the time slot, where `<start_time>` must be less than `<end_time>`.
  - If a course with the same ID already exists or if there is a time conflict, output `false` and ignore the insert command.
  - If the start time of one course is equal to the end time of another course, they do not count as a time conflict.
2. **Delete**: `delete <ID>`
  - `<ID>` is the ID of the course to delete. If the ID does not exist in the list, ignore the command.
3. **Update**: `update <ID> <new_start_time> <new_end_time>`
  - `<ID>` is the ID of the course to update.
  - `<new_start_time>` and `<new_end_time>` are integers for the new time slot. If this new time slot causes a conflict with existing courses, return `false` and refuse to update.
  -
4. **Print**: `print`
  - Print the entire list of courses in ascending order by start time. Each course should be printed on a new line in the format: `<ID>: <start_time> - <end_time>`. If the list is empty, output "Empty".
5. **Quit**: `quit`
  - Quit the program.

Note: Even if this problem can be solved with other approaches, only linked lists are suggested for your solution to enhance your understanding of linked lists.

Example:

Input:

`insert 1 9 11`

`insert 2 10 12`

`insert 3 13 15`

`print`

`update 2 11 13`

```
delete 1
```

```
print
```

```
update 3 14 16
```

```
print
```

```
quit
```

Output:

```
false
```

```
1: 9 - 11
3: 13 - 15
```

```
false
```

```
3: 13 - 15
```

```
3: 14 - 16
```

```

```

Output analysis per step:

1. **Command:** `insert 1 9 11`
   Output: (no output)
2. **Command:** `insert 2 10 12`
   Output:

   ```
   false
   ```

   ```

   ```

   Reason: Time conflict with Course 1 (9-11 overlaps with 10-12).
3. **Command:** `insert 3 13 15`
   Output: (no output)
4. **Command:** `print`
   Output:

   ```
   1: 9 - 11
   3: 13 - 15
   ```

   ```

   ```
5. **Command:** `update 2 11 13`
   Output:

```
false
```

```
```

Reason: Course 2 does not exist.

6. **Command:** `delete 1`
   Output: (no output)
7. **Command:** `print`
   Output:

   ```
   3: 13 - 15
   ```

   ```
   ```
8. **Command:** `update 3 14 16`
   Output: (no output)
9. **Command:** `print`
   Output:

   ```
   3: 14 - 16
   ```

   ```
   ```
10. **Command:** `quit`
    Output: (no output)