TIC1002—Introduction to Coputing and Programming II
National University of Singapore
School of Continuing and Lifelong Education

# Practical Examination 2

21 April 2018

**Time allowed:** 2 hours

**Instructions (please read carefully):**

1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **two** questions. The time allowed for solving this test is **2 hours**.
3. The maximum score of this test is **20 marks**. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
5. While you are also provided with the templates `q1-template.cpp` and `q2-template.cpp` to work with, your answers should be submitted on Coursemology. Note that you can **only run the test cases on Coursemology for a limited number of tries** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness locally on VS Code and not depend only on the provided test cases. Do ensure that you pasted your answers correctly by running the test cases at least once.
6. In case there are problems with Coursemology.org and we are not able to upload the answers to Coursemology.org, you will be required to name your file `q1-<student no>.cpp` and `q2-<student no>.cpp` where `<student no>` is your student number and leave the file in a `tic1002-pe2` folder on the Desktop. If your file is not named correctly, we will choose any .c file found at random.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology and correctly left in the desktop folder at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam, or the parts that are not uploaded correctly.
8. Please note that while sample executions are given, it is **not sufficient to write programs that simply satisfy the given examples**. Your programs will be tested on other inputs and they should exhibit the required behaviours as specified by the problems to get full credit. There is no need for you to submit test cases.

**ALL THE BEST!**

## Question 1: Chains [10 marks]

A `node` is a structure that contains an integer value, and a pointer to another node, as defined in the following:

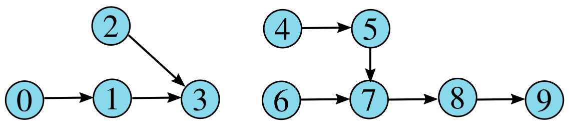
```
typedef struct node {
    int value;
    struct node *next;
} node;
```

A `graph` is a collection of `nodes`, defined as follows:

```
typedef struct graph {
    int size;
    node** nodes;
} graph;
```

Assume that the integer value of the nodes in the graph are all unique. Also assume that all the `nodes` in the graph either point to another `node` in the graph, or is `NULL`.

In other words, our graph consists of nodes chained together. One such example of a graph with nodes 0 to 9 could be:



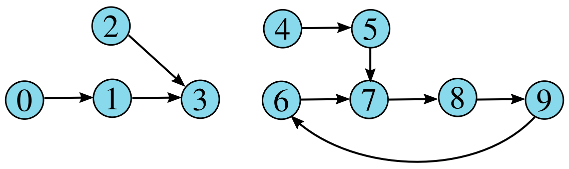
**A.** Implement the function `int longest_chain(graph g)` which takes in a `graph`, and returns the number of nodes in the longest chain in the graph. With the example graph above, the longest chain consists of 5 nodes: $4 \to 5 \to 7 \to 8 \to 9$.

You may assume that there are no cycles or loops in the graph. [4 marks]

**B.** On top of returning the number of nodes in the longest chain, your function should also print to standard out, the value of the nodes of the longest chain in order. The nodes should be separated with `->`. With the example graph above, the output will be: `4->5->7->8->9`.

You may assume that the longest chain is unique in the graph, i.e., no other chain has the same length as the longest. [3 marks]

**C.** We will remove the assumption that there are no cycles in the graph. Your function should handle such cycles and not double-count the revisited nodes. Consider the following graph:



The longest chain should now consist of 6 nodes: $4 \to 5 \to 7 \to 8 \to 9 \to 6 \to 7 \to \ldots$

Your function should return the value 6 and output `4->5->7->8->9->6->...` with the repeated nodes replaced with `...`.

[3 marks]

## Question 2: Fun with Queues [10 marks]

Consider the following `Queue` class:

```
class Queue {
protected:
    vector<int> items;

public:
    virtual void enqueue(int i) {
        items.push_back(i);
    }

    virtual int dequeue() {
        int i = items[0];
        items.erase(items.begin());
        return i;
    }

    virtual int peek() {
        return items[0];
    }

    int size() {
        return items.size();
    }

    bool empty() {
        return items.size() == 0;
    }
};
```

**A.** A `UniqueQueue` is a queue that does not allow duplicate entries. If the value being enqueued already exists in the queue, then nothing happens. Otherwise, the value is enqueued as in a regular queue.

For example, enqueuing the following elements

```
1, 4, 15, 3, 12, 11, 25, 3, 2, 4, 5, 1
```

in this order will return

```
1, 4, 15, 3, 12, 11, 25, 2, 5
```

when the queue is dequeued.

Without modifying the `Queue` class, provide an implementation for `UniqueQueue` which subclass `Queue`. Your implementation should employ OOP concepts as much as possible, e.g., you should not be duplicating code unnecessarily and make use of inheritance. [4 marks]

**B.** A `PriorityQueue` is a queue that is sorted. In other words, elements with a higher priority will be dequeued earlier even if they were enqueued later.

For our `PriorityQueue`, elements with the smaller rightmost digit have higher priority over ones with a larger rightmost digit. If two elements have the same priority, they will remain in the same order in which they are enqueued (it is stable).

For example, enqueuing the following elements

```
1, 4, 15, 3, 12, 11, 25, 3, 2, 4, 5, 1
```

in this order will return

```
1, 11, 1, 12, 2, 3, 3, 4, 4, 15, 25, 5
```

when the queue is dequeued.

Without modifying the `Queue` class, provide an implementation for `PriorityQueue` which is a subclass `Queue`. [3 marks]

**C.** A `UniquePriorityQueue` is, as its name suggest, a queue that does not allow duplicates like a `UniqueQueue`, while also prioritising the elements according to the same scheme as `PriorityQueue`.

For example, enqueuing the following elements

```
1, 4, 15, 3, 12, 11, 25, 3, 2, 4, 5, 1
```

in this order will return

```
1, 11, 12, 2, 3, 4, 15, 25, 5
```

when the queue is dequeued.

Without modifying the `Queue` class, provide an implementation for `PriorityQueue`. You may subclass either `UniqueQueue` or `PriorityQueue`, whichever you deem more suitable. These classes will be provided for in Coursemology so you do not need to include your implementation in your answer.

[3 marks]

## Appendix: ASCII Table

### ASCII Control Characters

| Num | Sym | Description |
|---|---|---|
| 0 | NUL | Null char |
| 1 | SOH | Start of Heading |
| 2 | STX | Start of Text |
| 3 | ETX | End of Text |
| 4 | EOT | End of Transmission |
| 5 | ENQ | Enquiry |
| 6 | ACK | Acknowledgment |
| 7 | BEL | Bell |
| 8 | BS | Backspace |
| 9 | HT | Horizontal Tab |
| 10 | LF | Line Feed |
| 11 | VT | Vertical Tab |
| 12 | FF | Form Feed |
| 13 | CR | Carriage Return |
| 14 | SO | Shift Out / X-On |
| 15 | SI | Shift In / X-Off |
| 16 | DLE | Data Line Escape |
| 17 | DC1 | Device Control 1 |
| 18 | DC2 | Device Control 2 |
| 19 | DC3 | Device Control 3 |
| 20 | DC4 | Device Control 4 |
| 21 | NAK | Negative Acknowl |
| 22 | SYN | Synchronous Idle |
| 23 | ETB | End of Trans Block |
| 24 | CAN | Cancel |
| 25 | EM | End of Medium |
| 26 | SUB | Substitute |
| 27 | ESC | Escape |
| 28 | FS | File Separator |
| 29 | GS | Group Separator |
| 30 | RS | Record Separator |
| 31 | US | Unit Separator |

### ASCII Printable Characters

| Num | Sym | Num | Sym | Num | Sym |
|---|---|---|---|---|---|
| 32 |  | 64 | @ | 96 | ` |
| 33 | ! | 65 | A | 97 | a |
| 34 | " | 66 | B | 98 | b |
| 35 | # | 67 | C | 99 | c |
| 36 | $ | 68 | D | 100 | d |
| 37 | % | 69 | E | 101 | e |
| 38 | & | 70 | F | 102 | f |
| 39 | ' | 71 | G | 103 | g |
| 40 | ( | 72 | H | 104 | h |
| 41 | ) | 73 | I | 105 | i |
| 42 | * | 74 | J | 106 | j |
| 43 | + | 75 | K | 107 | k |
| 44 | , | 76 | L | 108 | l |
| 45 | - | 77 | M | 109 | m |
| 46 | . | 78 | N | 110 | n |
| 47 | / | 79 | O | 111 | o |
| 48 | 0 | 80 | P | 112 | p |
| 49 | 1 | 81 | Q | 113 | q |
| 50 | 2 | 82 | R | 114 | r |
| 51 | 3 | 83 | S | 115 | s |
| 52 | 4 | 84 | T | 116 | t |
| 53 | 5 | 85 | U | 117 | u |
| 54 | 6 | 86 | V | 118 | v |
| 55 | 7 | 87 | W | 119 | w |
| 56 | 8 | 88 | X | 120 | x |
| 57 | 9 | 89 | Y | 121 | y |
| 58 | : | 90 | Z | 122 | z |
| 59 | ; | 91 | [ | 123 | { |
| 60 | < | 92 | \ | 124 | \| |
| 61 | = | 93 | ] | 125 | } |
| 62 | > | 94 | ^ | 126 | ~ |
| 63 | ? | 95 | _ | 127 | <DEL> |

**— E N D  O F  P A P E R —**