University of Leeds **School of Computing**
Procedural Programming COMP1711
Semester 1, 2017-2018

Individual Mini-Project 2

(Covers Assignments 4 and 5)

40 Marks

Submission Deadlines:
Timely submission: 13/11/2017 at 10:00 am
Late submission: 20/11/2017 at 10:00 (penalty applies)

This project is comprised of two tiers. Tier 1 is compulsory and must be completed by all students. If you satisfy all the requirements of Tier 1, you can earn the full mark of the project.

Tier 2 is optional, and you can choose to do it if you would like to stand out from the crowd. Tier 2 is not marked, but if you accomplish it correctly, your name will be written in the Distinction List on the VLE page of the procedural programming module, and if your code is exemplary it will also be placed on the VLE for all your colleagues to see.

Tier 1 (Compulsory)

The Brief

You will write a program for a surgery (medical practice) to help them manage patients' queueing for treatment.

The Details

The Problem

A surgery provides medical care to patients every day from 8.00 to 17.00. Patients can walk into the surgery any time during opening hours without an appointment. Arriving patients are treated on a first-come first-served basis.

The surgery has 10 examination (consulting) rooms numbered 1 to 10. Each room is staffed by one doctor. To reduce expenses and cope with the shortage of qualified doctors, not all rooms are staffed at all times. The surgery also has a patient waiting area that can accommodate up to 60 patients. To identify patients unambiguously, every patient is given a unique **3** digit *id number* when they register with the surgery for the first time. Patients must use this number every time they visit the surgery.

When a patient arrives at the surgery, they are checked in by a receptionist. If a doctor is free, the patient is immediately directed to the appropriate examination room. If all the doctors are busy, the

patient is added to a waiting list (a queue) and asked to relax in the waiting area until one of the doctors is free. Currently, the receptionist uses pen and paper to maintain the waiting list, and calls on each patient in turn when a doctor is free. The receptionist must also keep constant vigil on all examination rooms to promptly notice when a patient has left and a room is free again. Unfortunately, this manual method is prone to errors, and has caused many problems with patients and doctors.

The Task

The surgery needs a computer program to provide self-service to patients. The program can be used from a console (e.g. touch screen) in the waiting area. Patients use this console to check in on arrival to the surgery. They can also use the console to enquire about their position in the waiting list, or to find out how many doctors are currently available at the surgery.

The program is also used by doctors to check out (discharge) patients when they have finished examining them. Doctors use the program as well to check in and out of their rooms.

The program must maintain a waiting list (queue) of all checked in patients, and as soon as one of the doctors is free, the program must call on the next patient in the queue by displaying a message similar to this:

```
Patient ###, please go to Room ##
```

Where ### is the patient's unique id number, and ## is the number of the free room.

When a doctor is finished treating a patient, the doctor must check out (discharge) this patient using the console. We are assuming an over-simplified setting where doctors use the same console as patients. In a real situation, however, each doctor will have their own console on their desk. However, this will require programming methods that we have not yet learned.

The program must provide a Command Line Interface (CLI) accepting a number of different commands. A CLI is similar in working principle to the Linux shell we are already familiar with. A program using a CLI waits for the user to type in a command and when the user hits the Enter key the command is immediately executed; the program then waits for another command. To indicate that it is ready to accept commands, the program displays a command prompt (e.g. >>>) at the beginning of a new line.

The program accepts two types of commands: patient commands and administrator commands. Only doctors and surgery staff can use the admin commands. For simplicity, each command is comprised of one letter only. Some commands take an argument (e.g. number) that must be provided after the command name.

Patient Commands

The following commands are mainly used by patients, but they can also be used by doctors and surgery staff.

- The **i** command (patient check **i**n). This command must be used by patients to check in on arrival to the surgery. The i command takes one integer argument that represents the patient's id number. For example, a patient whose id number is 764 must type in this command on arrival:

  ```
  >>> i 764
  ```

If one of the doctors is free, the program will immediately display a message similar to this:

```
Please proceed to Room ##
```

Where ## is the number of the free room.

However, if all doctors are busy, the program must append this patient's id number to the end of the waiting list (the queue) and display a message similar to this:

```
All doctors are busy right now, please relax in the waiting area. You are number ## in the queue.
```

where ## is the patient's position in the queue.

- The **p** command (patient queue **p**osition). This command can be used at any time if a patient wants to know his or her position in the waiting queue. This command takes one integer argument that represents the patient's id number. For example, a patient whose id number is 764 must type in this command to find out their position in the queue:

  >>> p 764

When this command is entered the program must display a message similar to this:

```
You are number ## in the queue
```

where ## is the position of this patient in the queue.

- The **q** command (patient **q**uit). This command can be used if a patient decides to leave the surgery after checking in, and before seeing a doctor. This command takes one integer argument that represents the patient's id number. Using this command will remove the patient from the waiting queue, and display this message:

```
You have been removed from the queue. Thank you for your visit.
```

- The **d** command (available **d**octors). This command displays the number of doctors currently available at the surgery, and a list of all examination rooms that are currently staffed. When this command is executed, the program must display a message similar to this one:

```
There are currently # doctors in the surgery. They are in rooms ##, ##, … , and ##
```

where # is the number of doctors available at the surgery right now, and ## is a room number.

- The **h** command (help). This command displays a list of all available commands and a brief description of each one of them.

Administrator Commands

The following commands can only be used by doctors and other qualified surgery staff. Therefore, before attempting to execute any of them, the program must ask for, and verify, a **4 digit** password that is only known to doctors.

- The **o** command (patient check **o**ut). This command is used by doctors to check out (or formally discharge) a patient from the surgery. The o command takes one integer argument that represents

the patient's id number, for example, to check out patient 764, the doctor must type the following command:

>>> o 764

After this command is successfully executed, the program must automatically call upon the next patient in the queue and instruct them to proceed to this doctor's room with a message similar to this:

```
Patient ###, please go to Room ##
```

- The **r** command (doctor **r**eady). This command must be used by a doctor when they arrive at their room and they are ready to see patients. The command takes one argument representing the number of the examination room of this doctor. For example, if a doctor arrives at room 6, and is ready to see patients, they should type the following command:

  >>> r 6

  When this command is executed, the program must automatically calls on the next patient in the queue, and instruct them to proceed to room 6.

- The **a** command (doctor **a**way). This command is used when a doctor leaves his room. The command takes one argument representing the number of the examination room of this doctor. For example, before leaving room 9, the doctor must type the following command:

  >>> a 9

  When this command is successfully executed, no more patients can be forwarded to this room.

- The **w** command (**w**ho is waiting). This command displays the entire waiting queue. For example, if there are 4 patients waiting with id numbers of 345, 786, 222, and 801 respectively, the program displays a message similar to this:

```
The following patients are still waiting to be seen by a doctor:
345, 786, 222, 801
```

In the above example, patient 345 is in the first position in the queue, and 801 is in the last position.

- The **x** (e**x**it program) command. This command is used to terminate the program.

Implementation Instructions

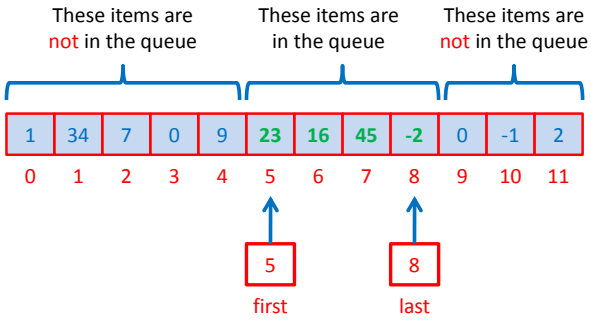
The queue data structure

The main task in this project is to implement a simple queue *data structure* for maintaining the list of waiting patients. A queue data structure works in a similar manner to a queue in everyday life, i.e. first come first served. The two main operations that a queue data structure must provide are:

- The ***enqueue*** operation, for appending a new data item to the end of the queue.
- The ***dequeue*** operation, for removing the first data item from the front of the queue.

The simplest way of implementing a queue in C is to use an array to store the queue items. In addition to the array itself, we also need two integer variables that are used as indices to the front and tail of the queue. These indices are usually called **first** and **last**. The following figure shows a queue of

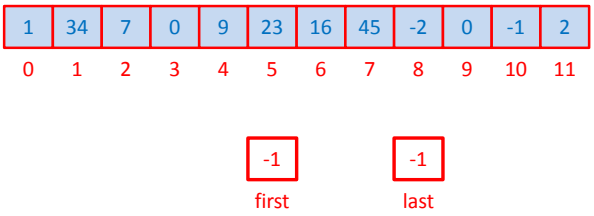
integers implemented with an array. In this example, there are exactly 4 items in the queue. The first item is in array location 5 and the last is in location 8.



Note that the first item in the queue is not always in location 0 of the array. The location of the first item keeps moving to the right every time we dequeue an item. If we want to keep the first item in location 0, we have to shift all items one place to the left after every dequeue operation.

Initializing the queue

Before using a queue, it must be initialised (i.e. cleared). To clear a queue, we assign an illegal index (e.g. a negative value) to both first and last, as shown in the following figure:



The above queue is considered empty, because neither first nor last point to a legal location in the array.

Enqueueing a new item to a queue

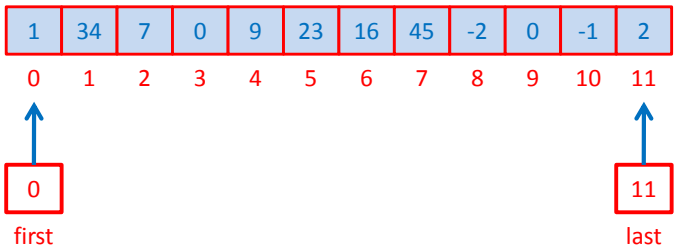
When enqueueing a new item, there are four cases to consider:

1. The queue is full. In this case we cannot add any items to the queue. This is an overflow case that should be handled as a run-time error.
2. The queue is empty (i.e. first == last == -1). In this case, the new item will be the first and last one in the queue. To append the item, we set first = last = 0, and copy the new item to a[0], where a is the array name.
3. The queue is neither full nor empty **and** the value of 'last' is less than the index of the last array location. To add the item, we increment 'last' and copy the new item to a[last].
4. The queue is neither full nor empty but the value of 'last' is equal to the index of the last array location. In this case, we set last to 0, and copy the new item to a[last].

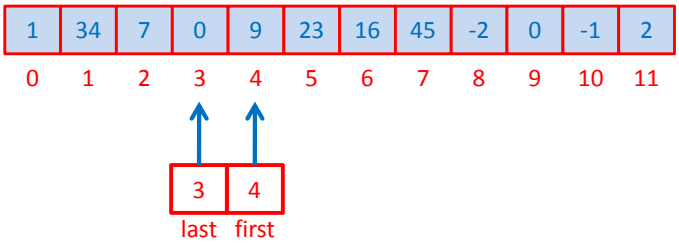
But, how can we decide if a queue is full?

There are two cases of a full queue, as shown below:

The first item is in location zero of the array and the last item is in the last location.



The first item is somewhere in the middle of the array and the last item has wrapped around and is in the location just before it.



Dequeueing an item from the queue

When dequeueing an item, there are four cases:

5. The queue is empty (i.e. first == last == -1). In this case there is nothing to dequeue. This is an exceptional case that should be handled as a run-time error.

1. The queue contains one item only (i.e. first == last). In this case, when this only item is dequeued, the queue becomes empty again.

2. The queue contains more than one item, and 'first 'is the index of the last array location. We get the value at a[first], then set first to 0.

3. The queue contains more than one item, and 'first' is **not** the index of the last array location. We get the value at a[first], then increment first.

General Guidelines

- In this project you will be using the following C statements and constructs only:
  - Variable declaration statements.
  - The printf and scanf functions and other IO functions.
  - Assignment statements.
  - Conditional 'if else' statements and other conditional statements such as the switch statement.
  - Loops.
  - Arrays.
  - Functions.
- The main objective of this project is to help you learn to use functions in programs, hence it is advisable to define and use functions whenever you feel that this will make your program concise and well structured. In particular, you **must implement** the following integer queue functions:
  - void enqueue (int n) // append n to the queue
  - int dequeue () // remove the first item in the queue, and return its value
  - int first() // get the first item without removing it

- int last () // get the last item without removing it
- void clear () // clear (initialize) the queue
- bool IsEmpty () // returns true if the queue is empty
- bool IsFull () // returns true if the queue is full
- void print () // print the entire queue
- int position (int n) // returns the position of n in the queue, or -1 if n is not in the queue
- void remove (int n) // remove n from the queue

You may also decide to define other useful queue functions.

- Please do **NOT** use structs or pointers in this project, even if you are already familiar with them.
- Write the program in standard C. If you write your code in any other language, it will NOT be assessed and you will get a zero mark.
- This is an individual project, and you are not supposed to work in groups or pairs with other students.
- Be aware that plagiarism in your code will earn you a zero mark and will have very serious consequences. It is much better to submit your own partially finished work, than to fall into the trap of plagiarism.

To simplify program development, you can divide your work into individual Work Packages (WP) as follows:

WP1: implement the queue data structure and its functions, and test them using simple code in 'main'.

WP2: implement and test the CLI. At this stage of program development, the CLI can read and distinguish various commands but cannot execute any of them.

WP3: Implement the various instructions, one at a time. Test each instruction as you proceed.

WP4: Test and debug your entire program.

WP5 (optional): Implement the simulator (see below).

<u>You will be provided with a set of test data and the corresponding program behavior so that you can test your program with confidence.</u>

Marking Scheme

The queue data structure, and all its functions are defined and used correctly	(8 marks)
The command line interface is implemented correctly	(5 marks)
All 10 commands work correctly	(20 marks, 2 marks each)
The program is well structured, clear, and efficient	(6 marks)
The program uses comments properly	(1 marks)
Penalty for late submission	(-5 marks)

Tier 2 (Optional)

There has been many cases when patients arrived at a much higher rate than doctors available at the surgery could handle them. When this happened, the number of patients in the waiting area exceeded its capacity, patients complained of unreasonable waiting times, and some doctors had to work beyond the surgery closing time to clear the patients queue. The surgery wants to know how many doctors must be available in the surgery at any one time to keep patients waiting times at a reasonable threshold, accommodate patients in the waiting area, or alleviate the need for doctors to work overtime

We need to add a simple simulator to our program that simulates patient queueing at the surgery for various values of the following parameters:

- Number of doctors staffing the surgery at the same time (<=10).
- Patients' arrival rate (patients/minute).
- The average patient consultation/treatment time (minutes).
- What time in the morning patients start to walk into the surgery. This need not be the same as the opening time.
- What time in the evening patients no longer walk into the surgery. This need not be the same as the closing time.

The simulator should determine the following:

- The number of unserved patients (if any) at closing time.
- How much overtime is needed by one doctor to clear the queue (if needed)?
- The average, minimum, and maximum waiting times for patients.

Write the above mentioned simulator in C, then add an **S** (simulate) command to launch the simulator from the command line.

.