Due:

February 25 2019, **before 11:59 pm.**

Notes

- Please follow both the "Programming Standards" and "Assignment Guidelines" for all work you submit. Programming standards 1-25 are in effect.
- Hand-in will be via the D2L Dropbox facility. Make sure you leave enough time before the deadline to ensure your hand-in works properly.
- Official test data will be provided several days before the assignment due date. Until then, work with your own test data.
- All assignments in this course carry an equal weight.

Assignment 2: Discrete Event Simulation

Description

To begin working in C++, you will write a program that will use a discrete-event simulation to simulate the operation of a hospital Emergency Department (ED). **Background material on event-driven simulations is posted to the website.** The ED will manage patients who enter the ED, and require treatment. Treatment consists of initial assessment (which happens for every patient), followed by possible diagnostics (e.g. blood work or X-ray), then by treatment. Lastly, the patient is discharged, and leaves the ED.

We will make one simplification from the way a real ED operates; we will not have any "Code Blue" patients arriving (patients arriving in an ambulance with a critical illness requiring immediate attention).

You should be making use of all the facilities for doing OO in C++ that we have covered, including appropriate use of abstract classes and methods, and safe dynamic casting. While we are not expecting you to completely follow OCCF standards, **you must write destructors to properly free any dynamically allocated memory** that you use. You don't need to worry about memory allocated in linked structures that will exist until the end of the program, but anything that is unlinked or goes out of scope during the course of the program's run must be properly disposed of. **You do not need to separately compile this program, but you can if you want to.** You may also use the *make* utility if you want to, but there is nothing forcing you to.

Details

A data file is used to drive the simulation. Each non-comment line in the data file describes a Patient arriving at the ED, with some information related to that Patient's treatment. The data file will be ordered by time, so you must have only **one arrival event in the event list at any time**. Each arrival must cause the next arrival event to be read. Each Patient will get an ID number (start at 1) on arrival at the ED, and each new Patient will have an ID one higher than that before it. You must use a C++ static variable to keep track of the next Patient number to be generated.

There are 9 events that may happen to any Patient in the ED:

An **Arrival** occurs when the Patient arrives in the ED. If an Assessment nurse is available, you should schedule a **StartAssessment** event; otherwise the Patient is entered into a queue of Patients waiting for their turn. (Use a strict FIFO queuing discipline since their priority is unknown at this point.)

A **StartAssessment** models meeting with an Assessment (triage) nurse, who assesses the patient's treatment needs. In our ED model, nothing actually happens here (because the treatment needs are spelled out in the data file), but we still include this in our model because this is what happens in a real ED. The only thing to actually do is to schedule a **CompleteAssessment** event (at the current time plus the assessment time, which is specified in the input file).

A **CompleteAssessment** event occurs at the end of the Patient's assessment. It is at this point the patient will be scheduled for any necessary diagnostics, followed by treatment. If a patient requires more than one diagnostic test, they can be done in any order, depending on whether or not there is a technician available to perform the test. For example, if a patient requires both blood work and an X-ray, if there is no available blood technician but an X-ray technician is available, then schedule a **StartXRay** event immediately. If there are no available technicians, the patient must wait (use a priority queue). If a patient is waiting for multiple technicians and they are all busy, then they can go in an arbitrary PQ for one of the needed technicians. If several technicians are available, one may be chosen arbitrarily.

Patients who don't require any diagnostic work (e.g., a patient who may just need a few stitches) may be scheduled for a **StartTreatment** immediately (if a doctor is available), or wait for treatment (again, using a priority queue).

The **StartBloodWork** and **StartXRay** events model the start of the particular diagnostic operation. Again, there is nothing to do here (the length of time needed is a constant specified for you, see Starting Code below) except to schedule a corresponding **CompleteBloodWork** or **CompleteXRay** event.

The **CompleteBloodWork** and **CompleteXRay** events occur when a diagnostic event is complete. Be sure to check if there are Patients waiting to start a diagnostic operation; if so, schedule the corresponding Start event.

The processing of these events is identical to the **CompleteAssessment** event. Determine if the patient requires further diagnostics or treatment; schedule a Start event or add the patient to the proper priority queue.

The **StartTreatment** event models the start of treatment. Again, there is nothing to do here (the length of time needed is already known) except to schedule a corresponding **Discharge** event.

The **Discharge** event occurs when the patient's stay in the ED is complete. Final statistics are gathered about this patient (for producing a summary report). Any patient waiting to start treatment should be scheduled.

You will need to maintain a list of future events in order by time. During the simulation process, at any point where the time unit is the same for two events, the Patient with the higher priority should be processed first. If there is still a tie, the Patient that arrived earlier should be handled first. This means that as the simulation goes on you will be maintaining a list of pending events that is ordered by primarily by time, but within time, ordered by priority and Patient number.

Starting Code

You will be provided with two files: `A2main.cpp` and `A2Const.cpp`. You must use these files.

The file `A2main.cpp` contains the main function; you may add code to it, but the code currently there must not be modified. This will help the marking by ensuring everyone's program operates in the same way. The main function accepts one command-line parameter (the name of the input file), and prints its output to standard output.

During program development, you can modify how the name of the input file is given to the main function (e.g. you can hard-code it), but be sure to comment out that code before you hand in. Your program should write to standard (console) output, not to an output file (again this makes things easy on the markers because they can just read output in a terminal window).

The file `A2Const.cpp` contains program constants for the ED simulation. The constants specify the ***number*** of resources available (assessment nurses, diagnostic technicians, doctors) and the ***duration*** of diagnostic procedures. If you are using separate compilation, you must move part of this file into a header file (`A2Const.hpp`).

Data File

The data file contains information on Patients and their treatment requirements. Each Patient is on one line of the file. Each line consists of a series of integers/Strings that describe (in order):

- The arrival time of the Patient (an integer).
- The assessment time (an integer): the time required to complete the initial assessment
- Patient priority (an integer): what level of priority this patient has when receiving diagnostics/treatment. The lower the number, the higher the priority (1 is the highest priority).
- Diagnostic requirements (a String of length 2,'B' or '-', and 'X' or '-'), indicating which types of diagnostics are required for this patient.
- Treatment time (an integer).

Lines that begin with a hashtag (#) are comments and are ignored.
For example:

```
# our first patient
2 8 2 B- 45
```

This describes a Patient arriving at time 2, requiring 8 units of assessment time, a priority of 2, requiring both blood work (but no X-ray), 45 units of treatment time, and is discharged from the ED.

The dataset descriptions above should be enough for you to generate your own test data to start with. Official test data will be provided several days before the assignment is due.

Data Structures

Your data structures must be your own, and you cannot use the C++ standard template library: you must make generic data structures (a Queue and an ordered list/priority queue) using your own linked structures and making use of C++'s object-orientation features. **In particular, you**

should have a polymorphic hierarchy of data items to go into generic data structures, and a polymorphic hierarchy of Events.

Output

Your program should produce output that indicates the sequence of events processed and when they occurred in order by time. That is, for each event, produce a single line of output indicating the time, and what occurred at that time.

At the end of the simulation you will produce a summary table. The summary table should indicate the following statistics for each station of the ED:

- Total number of patients served in the station.
- Total time the station was active (i.e., someone was being treated by the station)
- Total time patients were waiting at the station.
- Average service time per patient in the station.
- Average waiting time per patient in the station.

The four stations are Admissions, Blood lab, X-ray and Treatment room.
A sample input file with sample output will be made available later.

Hand-in

Submit all your source code for all classes. Each class should be in its own file. You should also submit two text documents:

- The output for the official test data.
- A short English description of how the interaction between objects in your project. This should answer questions like "Which class is initially called?", "Which method(s) process the input?", and which objects deal with tasks in the code. This description should be at most one page of text.
- Submit all files on UM Learn via the Assignment 2 Folder. You must hand in a compressed file (a zipfile, no rar files). You must complete the Honesty Declaration checklist before you are able to access the Dropbox.