Lab : Priority Queues

Information

In-class labs are meant to introduce you to a new topic and provide some practice with that new topic.

Topics: Priority Queue

Solo work: Labs should be worked on by each individual student, though asking others for help is permitted. Do not copy code from other sources, and do not give your code to other students. **Students who commit or aid in plagiarism will receive a 0% on the assignment and be reported.**

Building and running: If you are using Visual Studio, make sure to run with debugging. **Do not run without debugging!**

Using the debugger will help you find errors.
To prevent a program exit, use this before `return 0;`

```
cin.ignore();      cin.get();
```

Turn in: Once you're ready to turn in your code, prepare the files by doing the following: **(1)** Make a copy of your project folder and name it `LASTNAME-FIRSTNAME-LABNAME`. (Example: `HOPPER-GRACE-LAB-UNIT-TESTS`) **(2)** Make sure that all source files (.cpp, .hpp, and/or .h files) and the `Makefile` files are all present. **(3)** Remove all Visual Studio files - I only want the source files and Makefiles. **(4)** Zip your project folder as `LASTNAME-FIRSTNAME-LABNAME.zip`

Never turn in Visual Studio files!

Starter files: Download from GitHub.

Grading: Grading is based on completion, if the program functions as intended, and absense of errors. **Programs that don't build will receive 0%.** Besides build errors, runtime errors, logic errors, memory leaks, and ugly code will reduce your score.

Contents

1.1 About

1.1.1 Priority Queues

A priority queue is a queue where items are added in so that higher priority items are ahead of the lower priority items. This requires some form of sorting on the Push function to make sure that items are added in the right order.

We will learn more about **heaps** later on, which are another way we can implement a priority queue. For now, we are going to implement the priority queue in this lab with a simple array.

1.2 Lab specifications

In this lab, a series of homework assignments are generated, each with a due date, and an amount of work associated with it. The homework assignments will be added into a `PriorityQueue` and then processed based on the priority (in this case, what assignment is due sooner.)

In this assignment, the Processor/Work functions are already implemented and you just have to focus on the Priority Queue functionality.

1.2.1 Job

In this lab, the Job class looks like this:

```
1  struct Job
2  {
3      Job();
4      void Work();
5      void SetFinishDay( int day );
6      void Update( int timestamp );
7
8      int id;
9      string assignment;
10     bool done, overdue;
11
12     int dueOnDay, timeRemaining, dayFinished;
13 };
```

In the Processor, it will keep track of the "day", starting from 0. In the Job, `dueOnDay` corresponds to this day.

The `timeRemaining` variable is the amount of "days" of work remaining for the assignment.

The `dayFinished` is logged as which day the work remaining hit 0.

1.2.2 Priority Queue

The `PriorityQueue` class declaration looks like this:

```
1  const int MAX_SIZE = 1000;
2
3  class PriorityQueue
4  {
5      public:
6      PriorityQueue();
7      void Push( Job* task );
8      void Pop();
9      Job* Front();
10     int Size();
11     void Display();
12
13     private:
14     void ShiftRight( int atIndex );
15     void ShiftLeft( int atIndex );
16
17     Job* m_arr[MAX_SIZE];
18     int m_itemCount;
19 };
```

void PriorityQueue::Push(Job* task)

Error check: If the internal array `m_arr` is full, throw a runtime error that additional items cannot be added.

Locate the index where the new item `task` should be inserted at. Use the Job's `dueOnDay` field to compare to what is already in the array.

In other words, iterate through the array. Once you find an item with a lower priority (the due date is greater) than the new `task`, insert the new

task at this position.

You can access the dueOnDay like this:

In the array: `m_arr[insertAt]->dueOnDay`

The new task: `task->dueOnDay`

Once you find the proper location to add the new item, call `ShiftRight` at that index to make room for the new item. Afterwards, set

```
m_arr[insertAt] = task;
```

and increment the item count.

void PriorityQueue::Pop()

Remove the front-most item from the queue, and push everything forward by one slot. Decrement the item count. You can use the `ShiftLeft` function.

Job* PriorityQueue::Front()

Return the front-most item.

1.3 Example output

1.3.1 Running the program

```
------------------------------------------------------
 | Homework Prioritizer |
 ------------------------


--------------------------------
 How many tasks? (More than 10)

 >> 10


Creating homework...

Filling queues...

Processing with FCFS...

DONE
```

1.3.2 result.txt

```
Doing my homework!

-------------------------------------------------------
DAY: 0

Processing task 2...

ID   DUE ON    REMAINING      FINISHED DAY        OVERDUE
0    15        6
1    37        2
2    13        4
   <<
3    13        1
4    39        15
5    17        5
6    24        9
7    29        11
8    18        5
9    18        7


-------------------------------------------------------
DAY: 1

Processing task 2...

ID   DUE ON    REMAINING      FINISHED DAY        OVERDUE
0    15        6
1    37        2
2    13        3
   <<
3    13        1
4    39        15
5    17        5
6    24        9
7    29        11
8    18        5
9    18        7

(etc)

-------------------------------------------------------
DAY: 64

Processing task 4...

ID   DUE ON    REMAINING      FINISHED DAY        OVERDUE
```

```
0     15          0                11
1     37          0                50                  overdue
2     13          0                4
3     13          0                5
4     39          1                                    overdue
   <<
5     17          0                16
6     24          0                37                  overdue
7     29          0                48                  overdue
8     18          0                21                  overdue
9     18          0                28                  overdue


------------------------------------------------------------
DAY: 65

Processing task 4... done!

ID    DUE ON      REMAINING        FINISHED DAY        OVERDUE
0     15          0                11
1     37          0                50                  overdue
2     13          0                4
3     13          0                5
4     39          0                65                  overdue
   <<
5     17          0                16
6     24          0                37                  overdue
7     29          0                48                  overdue
8     18          0                21                  overdue
9     18          0                28                  overdue
```