Priority Queue

Topic(s):	Priority Queues, Code Reusability, More Advanced Makefiles, Debugging, Testing
Date assigned:	Wednesday, October 19, 2016
Date due:	Wednesday, November 2, 2016, 9:15 am
Points:	40 pts

For this assignment, you are to implement a Generic Dynamic Priority Queue (GDPQ) ADT in a file called pqueue.c using the header file pqueue.h. You can find this header file on zeus in **/home/CS300Public/2016/05Files**. All of the data structures and function prototypes are defined in the header file. Further, each function prototype has been described to the point that you should be able to implement each function. The error codes that can be produced are listed for each function. Higher precedence error codes are listed first.

The GDPQ must be implemented using the Generic Dynamic List (GDL) from the previous assignment as the base data structure. We will be reusing this GDPQ later on, so make sure you have completely tested and debugged each operation.

In addition to implementing the GDPQ data structure, you must provide a Makefile and test driver (pqueuedriver.c that produces an executable named **pqueuedriver**) that thoroughly tests your GDPQ. The driver must display to the screen a series of SUCCESS or FAILURE messages, with enough description that a user can quickly spot broken functionality.

You may add any helper functions you need to pqueue.c. These helper functions must be marked **static** so they are not available outside the module. You may not alter pqueue.h in any way.

1. Your code is to be written in C using Eclipse. Programs written in other environments will not be graded. Create an Eclipse project named **GenericDynamicPriorityQ**. This project must contain the directories: **src**, **include**, and **bin** with a **Makefile** at the same level as these directories.
2. The Makefile must contain the necessary targets to build the pqueuedriver as well as a **clean**, **valgrind**, and **tarball** targets. Typing **make** on the command line must build pqueuedriver.
3. Submit a file called cs300_5_PUNetID.tar.gz using the submit script by 9:15am on the day in which the assignment is due. This file must include your GenericDynamicPriorityQ **AND** your updated (if necessary) GenericDynamicList projects.
4. Submit a color, double sided, stapled packet of code by the same deadline in 3. The packet must be in the following order:
   Priority Queue Driver (.h then .c if you have both, otherwise just .c)
   pqueue.c (do not print pqueue.h)
   Any extra .h/.c pairs you have. (do not include any code from the List project)
   Makefile
5. Test one function at a time. This will lessen your level of frustration greatly.
6. You are to use the coding guidelines from V6.3 of the coding standards.
7. The only changes to GDL you can make are to fix bugs in the .c files.

8. You must insert GenericDynamicPriorityQ into your Subversion repository; GenericDynamicList must already be in Subversion and any bug fixes must be committed to Subversion.
9. **IMPORTANT:** When implementing your GDPQ ADT, you are to use the functions from the GDL module and not access any GDL data directly. As an example, the function pqueueSize could access numElements of the GDL directly BUT must not. Instead, the function lstSize is to be used. Failure to access information correctly will result in losing major design points.

Goals for this assignment:

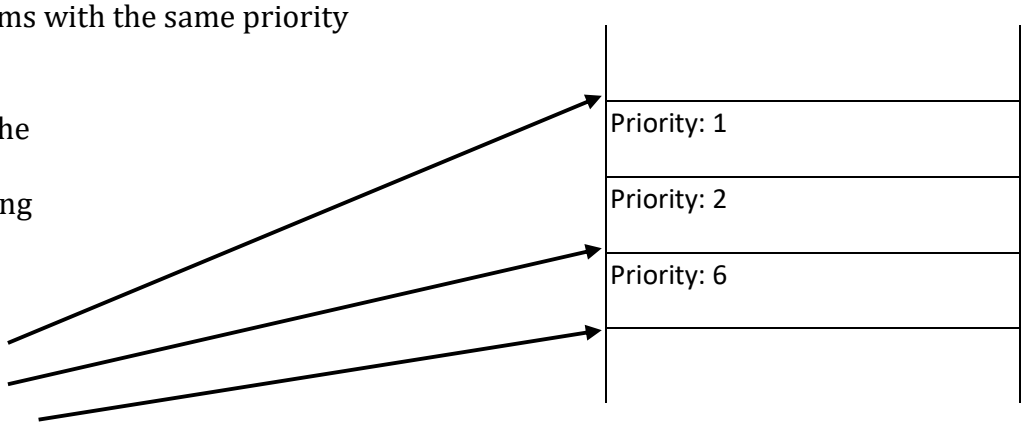
1. Reuse your GDL code.
2. Code and test your program one function at a time.
3. Write efficient/clean code
4. Use the debugger to effectively develop a correct solution
5. Thoroughly test your code.
6. Write code with no Valgrind errors.

Priority in the Queue.

You must implement priority in your queue by inserting items into the queue using the priority value provided by the user. A priority of **zero** is the **highest priority**. A newly inserted item must be inserted:

1) ahead of all items with a lower priority
2) behind all items with the same priority

For example:
The Priority Queue on the right already contains some data. The following inserts will add data at the marked points.



insert Priority 0
insert Priority 2
insert Priority 10

♦ The function **pqueueChangePriority** accepts an integer (positive or negative) and adds that integer to the priority of *every* item in the queue.

♦ **There is only one deadline**. I expect you to start this project soon. Your priority queue implementation will likely be smaller than your pqueuedriver.

Using Eclipse, Makefiles, and Multiple Projects.

Since your GDPQ relies on your GDL, which is in another Eclipse project, your **Makefile** may contain lines like the following.

```
bin/pqueue.o: src/pqueue.c include/pqueue.h ../GenericDynamicList/include/dlist.h
      ${CC} ${CFLAGS} -c src/pqueue.c -o bin/pqueue.o
```

In this example line, pqueue.o relies on pqueue.c and pqueue.h from the current GDPQ project as well as the header file dlist.h from the GDL project, which exists up a directory (to your workspace root) and then down in the GDL project's include directory. Your driver will also need to depend on the dlist.o file in the GDL project.

If you want to rebuild dlist.o via your GDPQ Makefile you may need a line like this in your GDPQ Makefile.[1]

```
../GenericDynamicList/bin/dlist.o:  ../GenericDynamicList/include/dlist.h  \
                                    ../GenericDynamicList/src/dlist.c
      cd ../GenericDynamicList; make bin/dlist.o
```

This moves to the GDL directory and invokes make. The Makefile in GDL is read and list.o is rebuilt if necessary. Note that make executes *each line* of your file with a new shell so if you **cd** on one line and run a command on the next line, the command is run as if the **cd** had not been run.

You are most likely going to run into Eclipse problems with this project. Namely, Eclipse may not see an update to the GDL data structure while you are coding in the GDPQ data structure and may produce errors *even if the Makefile succeeds in building your .o and executable files.*

If you right click a project, choose Properties, and select Project References you can mark which other projects this project relies on. (GenericDynamicPriorityQ relies on GenericDynamicList, for example). This helps Eclipse determine where to look for data type definitions and header files. Eclipse is not perfect. Sometimes projects get out of sync and you need to: clean and build each project, right click a project, choose Index, and Rebuild.

Another way to set the references is: **Right click on Project > Properties > C/C++ General > Paths & Symbols > References**[2]

If you have Makefile problems, see me early. I do not want to see more code in your project than pqueue.h, a single function pqueueCreate in pqueue.c, and a single call to pqueueCreate in pqueuedriver.c. I'm not trying to be difficult, but the more code you have, the harder it is to fix a Makefile problem especially since this project uses code from another project.

I expect you to start this project early. **This code will be reused in subsequent assignments.** Further, you are reusing code from GDL!!!!! **You've been warned**.

---

1 http://crawlicious.com/wp/2009/06/11/make-change-dir/

2 http://stackoverflow.com/questions/1270799/eclipse-cdt-make-a-project-rebuild-when-a-library-built-in-another-project-was-r

Some helpful information.

It's Thursday, July 14, 2016 and I just finished the Priority Queue assignment. It's pretty brutal. Ummmm.

What makes this assignment difficult?

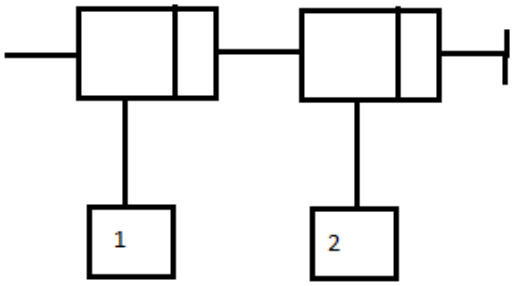
1) The Dynamic List functions are used to implement a Priority Queue. If your list solution is rock solid (it's not), this won't be an issue. Reusing code that isn't bullet proof makes debugging MUCH more difficult.
2) There are void *'s everywhere. Remember a void * is a typeless pointer. You can't just dereference a void * to find out data values of what the void * is pointing to.
3) The debugger is much harder to use with void *'s. You can build some expressions in Eclipse where a void * is cast to, say an int, and then you can dereference the pointer, BUT the void * might be pointing to something that also has a void *.
4) It's really hard to wrap your head around what is going on with all of the pointers being used. What's the big picture?

Here is information that will help you successfully implement your Priority Queue.

The big picture is …

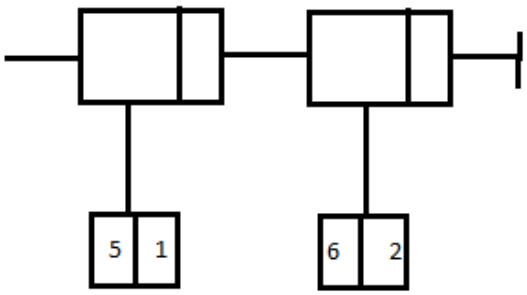
For a linked list given:

```
int value = 1;
lstInsertAfter (psList, &value, sizeof (int));
++value;
lstInsertAfter (psList, &value, sizeof (int));
```



For a priority queue given:

```
int value = 1;
pqueueEnqueue (psQueue, &value, sizeof (int), 5);
++value;
pqueueEnqueue (psQueue, &value, sizeof (int), 6);
```



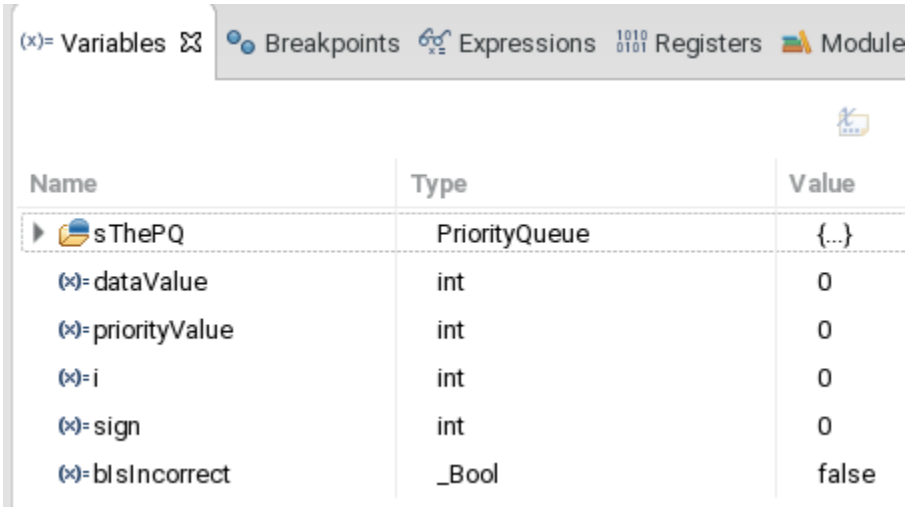
What I did was allocate enough space for the priority AND the data which in this case is a simple integer. Remember, the data can be any contiguous memory, such as an int, char, float, struct, array, BUT cannot be a pointer such as int * or void *. For the above priority queue example, in priority queue I allocated enough space for the priority AND data. Then I copied the priority into the first 4 bytes and the data into the next 4 bytes before inserting into the list, so for example, the 5 and 1 is a priority of 5 and a data value of 1. The 5 & 1 is a contiguous block, not a Node even though it kind of looks like a Node.

I had several tough errors and the debugger is your friend even though you might not think so. Specifically, I had problems with pqueueDequeue and more specifically, this lead to problems in my list routines; therefore,

1) set a break point at pqueueDequeue function call that is causing the problem.

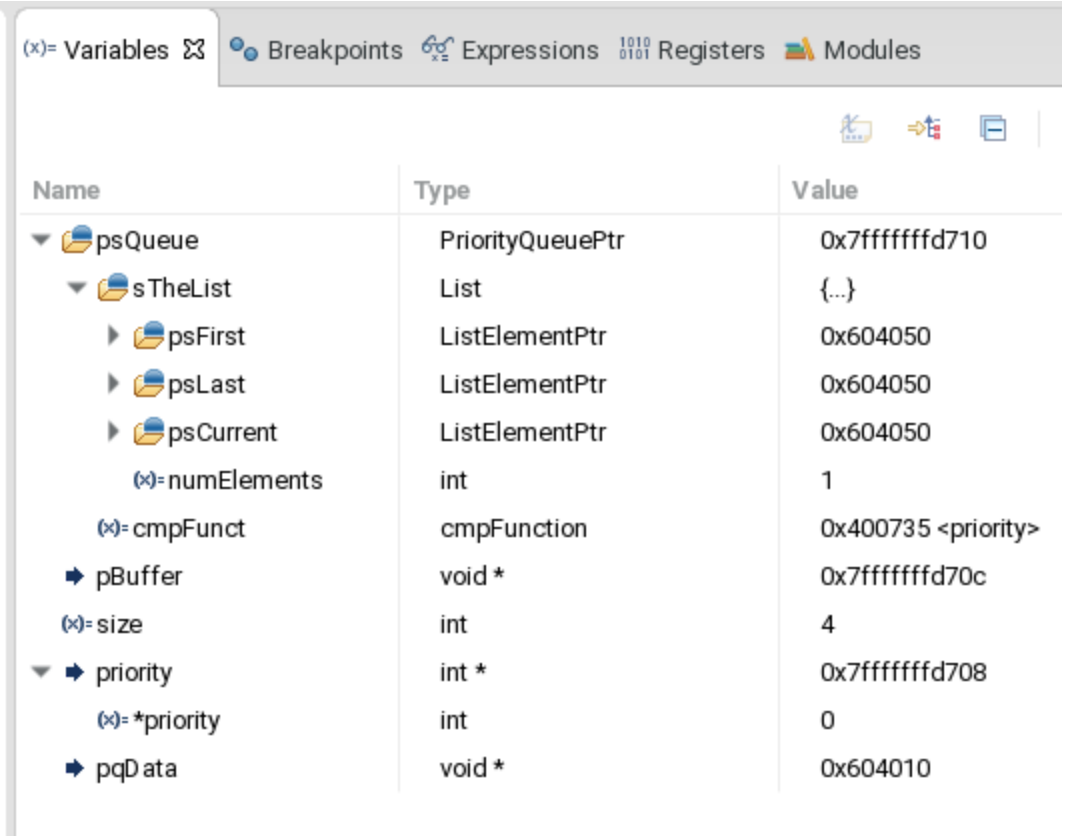
```
151   priorityValue = 0;
152   dataValue = 0;
153   pqueueDequeue (&sThePQ, &dataValue, sizeof (int), &priorityValue);
154
```

Notice that priorityValue and dataValue are being passed to pqueueDequeue as 0. These variables are typed and show up in the Variables tab quite nicely.



2) step into pqueueDequeue .... OUCH!!!!! The void *'s are just pointers. Notice that priority is an int * and the value of *priority shows up quite nicely, but what about pBuffer?

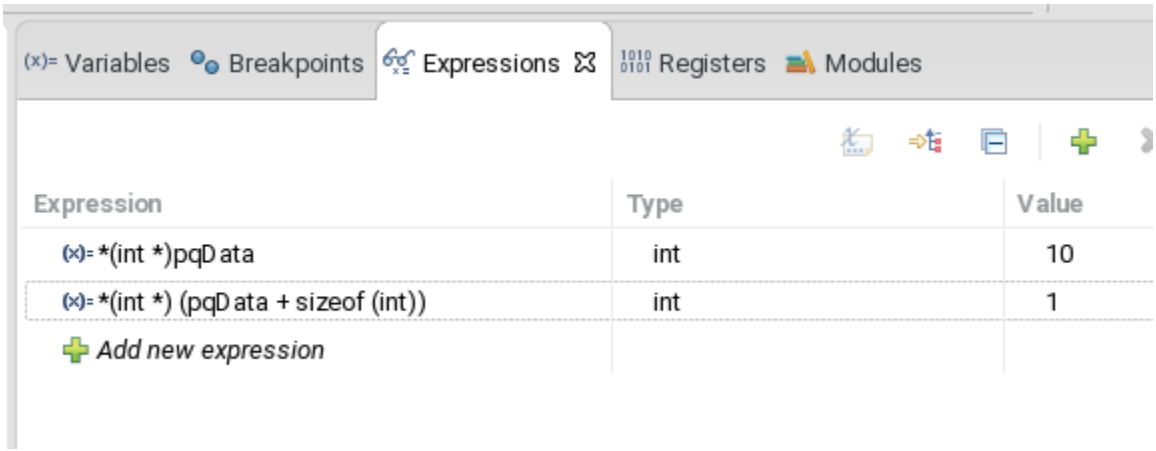
```
180 void * pqueueDequeue (PriorityQueuePtr psQueue, void * pBuffer, int size,
181                        int  *priority)
182 {
183   void * qData;
184
185   if (NULL == psQueue)
186   {
187     processError ("pqueueDequeue", ERROR_INVALID_PQ);
188   }
```



3) So a little further down in pqueueDequeue I have a statement that looks like:

```
198   pqData = malloc (sizeof (int) + size);
199   lstFirst (&(psQueue->sTheList), pqData, sizeof (int) + size);
```

Well pqData is a void * so what now? I know the first 4 bytes are the priority and the next size bytes is the data. Since my data is only integers right now, after the 4 bytes of priority data, the next 4 bytes are the integer data, so what is being returned. Click on Expressions and enter expressions that will tell you.



I know what the queue should look like and the results returned from the list operation are indeed correct. If not, DEBUG DEBUG DEBUG. Did I mention this isn't easy, so GET STARTED ASAP.