Introduction

For this project you will write a simple module that does queue manipulation. While this program is perhaps not very interesting in its own right, it will allow you to “brush up” on some of the finer points of C — namely structs, pointers, and dynamic memory allocation.

In coding, be sure to follow good programming practices and consistent conventions — we don’t care that your conventions are not our conventions, but your conventions must be both reasonable and consistent. In particular, remember to carefully document each module’s interface, data structures, and specification (what the module does, but not how it is implemented).

The files queue.h and queue.c on the class home page comprise the queue module. queue.h is complete. queue.c contains stub functions. See “Testing” below for what to do with test.c.

The Queue Module

This module implements and maintains a number of queues that can be maintained in either FIFO (first-in-first-out) or key order. The queue elements are of type qnode:

```c
typedef struct qnode {
    struct qnode *next;
    struct qnode *prev;
    int key; \\ /* other things would go here */
} qnode;
```

qnode is defined in the header file queue.h. A queue is implemented as a doubly-linked list using the next and prev fields to point to the succeeding and previous elements, respectively. The queue itself is represented by a pointer to a queue header, a structure of type qnode which is not considered to be an element of the queue. If the queue is empty, the next and prev fields of the queue header point back to the header itself.

The module should have the following externally visible functions:

```c
void initqu(qnode *queue)
```

Initialize the queue header pointed to by queue to be the empty queue. The header node must already exist.

```c
int emptyqu(qnode *queue)
```

Return 1 if the queue is empty, 0 if it is not.
void insqu(qnode *queue, qnode *element)

Insert the element pointed to by element into the queue of which queue is a member, immediately after the element pointed to by queue. This function may be used to insert at either the beginning or the end of a queue. Suppose the queue header is pointed to by a variable named q. Then to insert an element element at the end of the queue use insqu(q->prev, element) and to insert at the beginning of the queue, use insqu(q, element).

qnode *remqu(qnode *element)

Remove the queue element pointed to by element from whatever queue it is in, returning that element itself. Exception: If the element is the only thing in the queue (i.e., the header of an empty queue), then leave the queue unchanged and return NULL. This function may be used to remove from either the beginning or the end of a queue. Suppose the queue header is pointed to by a variable named q. Then to remove the first element in the queue use remqu(q->next) and to remove the last element, use remqu(q->prev). If you are not careful, you can remove the queue header, effectively destroying the queue.

void ordinsqu(qnode *queue, qnode *element)

Insert the element pointed to by element into the queue whose header is pointed at by queue based on the value in the key field. The queue should be maintained in increasing order by key value; if one or more elements already exist with the key value of the given element, insert the new element after the existing elements.

qnode *ordremqu(qnode *queue, qnode *val)

Remove the first element in the queue whose key value is val and return it. If the queue is empty or a element with the given value does not exist, return NULL and leave the queue unchanged.

Implementation and Testing

Create a new Win32 console application project in Visual C++ or use your favorite C compiler. Copy the three files queue.h, queue.c, and test.c into the project directory and add them to the project. Build the project. It should compile and link with a few warnings (remember — queue.c contains stubs). Running it will generate a memory access exception due to a dangling pointer. Complete the stubs in queue.c. test.c is a test driver module. The test module reports on its progress by posting messages to stdout. It will attempt to diagnose errors, but its analysis is primitive.

Project Turn-In

E-mail queue.c to kelliher@bluebird.goucher.edu by the beginning of class on the 7th. A 10% penalty will be applied for each day the project is late. The weekend counts as one day. After three days I will no longer accept projects.