CS 1723, Data Structures
Fall Semester, 1998
Programming Assignment 7, Due October 28, 1998
Queues Using Linked Lists

The assignment: For this assignment you are to implement a queue of characters based on a linked list. You should mimic the linked list code for a stack that was passed out in class. See the directory on runner `~wagner/pub/CS1723` for the stack examples, files `lists.text` and `listss.c`. A reasonable approach is discussed at the end of this handout. It is important that you write the linked list code yourself without referring to textbooks.

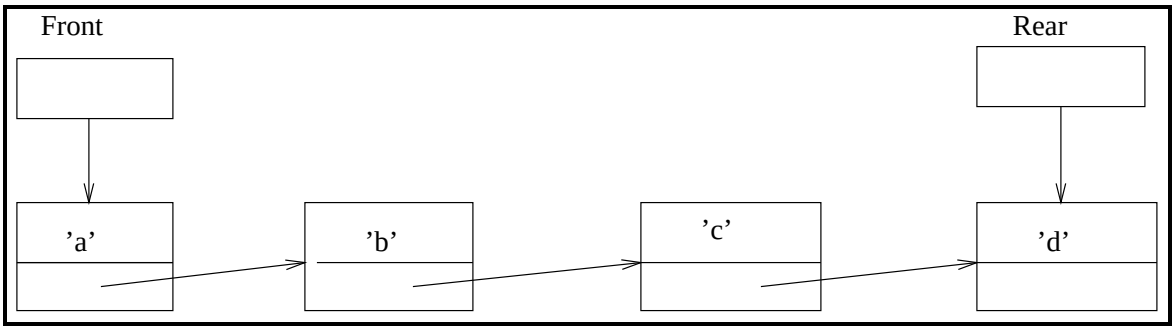
Your program should use the following `queue.h` header file:

```
/*
 * queue.h -- queue header file
 */
char remove(void);  /* remove oldest queue item, from front */
void insert(char);  /* Insert as new item, at rear */
int emptyq();       /* 1 means that the queue is empty */
int fullq();        /* 1 means no more room (which won't happen) */
```

Your `main()` function can be similar to the simple command interpreter in the stack routine. Your program must successfully handle an attempt to delete from an empty queue. You must use separate files for the main function, the queue header `queue.h` and the code for the queue itself, `queue.c`. Everything related to the actual linked list implementation should be buried inside the file `queue.c`. Your program need not free storage that is no longer needed, and it also need not check for a NULL returned by `malloc`. Your program should produce something like the following output:

```
runner% queue
ia
Inserted: a
ib
Inserted: b
ic
Inserted: c
id
Inserted: d
r
Removed: a
r
Removed: b
r
Removed: c
r
```

```
Removed: d
r
Empty queue
ix
Inserted: x
iy
Inserted: y
r
Removed: x
r
Removed: y
q
runner%
```



Linked list implementation: Everything related to the implementation must be hidden in the file `queue.c`, so that you could replace linked lists in this file with a circular array and the code in the header `queue.h` and in the file with the main function would remain unchanged. Your queue should look like the diagram above, with pointers named `Front` and `Rear`. (The diagram shows the result after inserting 'a', 'b', 'c', 'd', in that order. You need to decide how to represent an empty queue. (Perhaps both `Front` and `Rear` equal to `NULL`.) In writing the code for `insert` and `remove`, it is natural to handle the case of an empty queue separately from that of a non-empty queue. (You may even need to handle a queue with one element as a separate case.) Then sometimes it is possible to combine code so that one sequence of code handles all cases, but this is only to make the code look more elegant.

What to hand in: Just a listing of the code, and a run like the one above.