CSC A48 Assignment 1

Wacky Queue

Due: Saturday, February 24, 2018 at 5:00pm.
(*later than originally planned because of term test 1*)

⋆ introduction

For this assignment you will implement an ADT called the *wacky queue*. Here is its description.

⊙ wacky queue data

- A sequence of objects, each with an integer *priority*. The objects are ordered as follows.
  - Objects with higher priority are ahead of objects with lower priority within the sequence.
    - Larger (more positive) priorities correspond to higher priorities.
    - Smaller (more negative) priorities correspond to lower priorities.
  - When objects have equal priority, those inserted earlier are ahead of those inserted later.

⊙ wacky queue operations

`insert(obj, pri)`: Insert object `obj` with priority `pri` into the wacky queue.

Duplicates are allowed. I.e., a wacky queue may contain multiple copies of the same object, with same or different priorities.

`extracthigh()`: Remove and return the first item in the wacky queue.

Requires: The wacky queue is not empty.

`isempty()`: Return (a boolean indicating) whether the wacky queue is empty.

`changepriority(obj, pri)`: Change the priority of the first copy of object `obj` to `pri`.

The wacky queue is unchanged if `obj` is not in it or already has priority `pri`.

If the priority of `obj` is changed, then the insertion time of `obj` is taken to be the time of the `changepriority` operation.

`negateall()`: Negate the priority of every object in the wacky queue.

For this operation the order of insertion times of objects in the wacky queue is reversed. Thus the order of objects with equal priority is also reversed. *You should thank us that we made it this way!*

`getoddlist()`: Return a pointer to a linked list of `WackyNode`s containing every other object in the wacky queue, starting with the first object.

I.e., the returned list must contain, in order, the first object, the third object, the fifth object, etc.

If there is no first object, then an empty list is returned.

Each `WackyNode` has exactly the following attributes.

`_item`: Object in the node.

`_priority`: Priority of object in the node.

`_next`: Pointer to the next node.

`getevenlist()`: Return a pointer to a linked list of `WackyNode`s containing every other object in the wacky queue, starting with the second object.

I.e., the returned list must contain, in order, the second object, the fourth, the sixth, etc.

If there is no second object, then an empty list is returned.

See `getoddlist` operation for description of `WackyNode`.

◯ additional wacky queue implementation requirements

The client who wants a wacky queue has additional requirements regarding time and space (memory use) efficiency.

- The time taken for the `getoddlist` and `getevenlist` operations must be independent of the number of objects in the wacky queue. For example, `getoddlist` and `getevenlist` must return in about the same amount of time whether the wacky queue contains two items or two billion items.
- The amount of memory used for any operation, above and beyond that which is used to store the data, must be independent of the number of objects in the wacky queue.
- Memory used to store the data must be minimized. In particular, there must be no redundant instance variables in the implemenation of the wacky queue.

These additional requirements should more or less dictate the representation invariant used in any implementation. *Can you see why we must have the following?*

- Items in a `WackyQueue` are stored in two linked lists of `WackyNode`s.
- The only instance variables are two pointers to the heads of the two linked lists. There cannot be any other class variables.
- No operation is allowed to use an iterated type (like list or tuple) to temporarily store data.

⋆ what to do

1. Design a suite of test cases for a `WackyQueue`.
2. Download the file `wackynode.py`.
   It contains the declaration of the `WackyNode` class, which is used for the linked lists returned by the `getoddlist` and `getevenlist` operations. Read and understand this code.
3. Download the file `wackyqueue.py`.
   Add your name to the license at the top to indicate that you have added intellectual value to it. You must not add any `import` statements, or change the one there. You may only distribute this file, or a modified version of it, with the same license, and along with the file `GNUlicense`.
4. Complete the implementation of the `WackyQueue` ADT in `wackyqueue.py`.
   As always, use good proramming style and documentation.
5. Test your `WackyQueue` code with your test cases from step 1.
6. Submit your `wackyqueue.py` to Markus.