CSE 241 Assignment #6 – Produce and Consume Spring 2015

Due date: May 19 - 20:00

Description

In this project, you deal with a buffer, which is implemented as a circular doubly-linked list (LL or ll) and an iterator to traverse the LL.

- circular doubly-linked list – a linked list is a data structure that has the ability to grow and shrink dynamically and allows direct access to elements using indexing. In a circular doubly-linked list, each node contains two pointers: one to the next node in the list, and the other to the previous node in the list. Moreover, the last node points back to the first node in the list, not to NULL, as the next node, and the first node points back to the last node in the list, not to NULL, as the previous node.
- iterator - a pointer-like object that accesses the elements in a circular doubly-linked list

The buffer class supports two major operations of *produce* and *consume*. For a produce operation, a node is created and added to the linked list, while for a consume operation, a node is deleted from the list.

For this project you will be required, among other things, to write the code for 3 classes:

1. Node
2. Buffer
3. Iterator

In ~cse241/assign6, you can find the header files for the classes. You may **NOT** change the public methods in the header files, but may add private methods as needed. You may **NOT** add private, public or protected data members.

After copying the header files Node.h, Buffer.h, and Iterator.h into your assign6 directory, you should create the corresponding Node.cpp, Buffer.cpp, and Iterator.cpp files and write the code that implements the member functions for these classes.

In each header file, you will find the class with it's member variables and member functions. There is a comment for each member function describing what it should do. You are to implement the member function so that it does what the comment states that it should do and nothing more.

Next you should write 3 unit test files to test each of your classes:

1. testNode.cpp

2. testBuffer.cpp
3. testIterator.cpp

Each unit test file should have a main() for a particular class that calls and tests each public method. It also tests the private methods which are called from public methods. Note that when developing/writing code, you should do so in parts/modules and test each as you create it (keeping backup copies!). So for example, when writing Node, you also write a testNode.cpp that is the main() for Node.cpp. It should call all public functions and include appropriate output to test that all Node.cpp code works correctly.

And finally you should test your code with the main supplied: main.cpp

Example

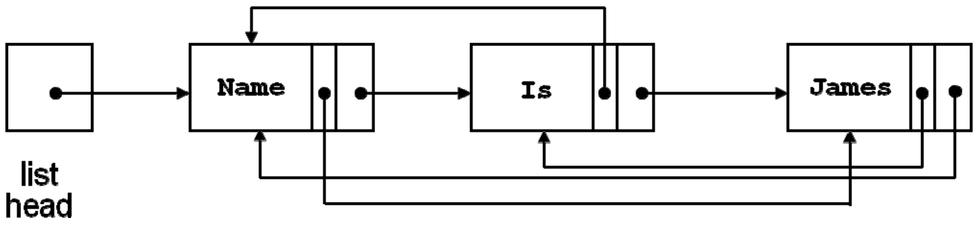

Figure 1. A circular doubly-linked list

Figure 1 shows a circular doubly-linked list with three elements. Each element has two pointers, one to the next node, and the other to the previous node. In case of the first node of "Name", its pointer to the previous node points to the last node of "James" in the list, while in case of the last node of "James", its pointer to the next node points to the first node of "Name" in the list

Implementation and Compilation

1. Create assign6 directory under your home directory.
```
$ mkdir assign6
```
2. Change your current directory to assign6 directory.
```
$ cd assign6
```
3. Copy provided files from ~cse241/assign6
```
$ cp ~cse241/assign6/* ~/assign6
```
4. Implement all the member functions in Node.cpp, Iterator.cpp, and Buffer.cpp. Test each class with your own Node.cpp, testBuffer.cpp and testIterator.cpp, respectively, and test with provided main.cpp. (Note that you do not need to submit your test cpp files.)
5. In order to compile, simply type "make" in the shell prompt as follows:
```
$ make
```
"make" command will create an executable a.out file in your current directory.

6. In order to test, you may change main.cpp, but do not put any required class implementation inside main.cpp, because I will overwrite the main.cpp in order to test your classes with other test cases.
7. Test and debug your algorithm with *lots of examples*.

Submitting Your Code

You should submit the whole assign6 directory, not individual files. Turn in your project using the "submit" command as follows:

Move to your home directory.

```
$ cd ~
```

Submit assign6 directory as follow

```
$ oopsubmit assign6 assign6
```

The second assign6 is the directory that contains your source codes.

Late submissions will lose 20 points per day.

If you do not follow this submission guideline, you will lose 10 points out of 100.

You should also submit a hard copy of your code to TA. Your report must have a cover page with your student ID and name. In the report, your code must be well commented to explain your class. 10 points will be given for the report. Also, the sample input and the output of your implementation must be included in the report.