CS 2401 – Fall 2022

Project 4 – Template Classes and Doubly Linked Lists

Due: 11:59 PM Wednesday, October 26th

This list will be like the sequence class that we created for Project Two, but it will be implemented differently, and the iterator access will be done as an external iterator. This list will also be implemented as a template, allowing it to hold any type of data. This will be a *universal list*.

Our list will be a *doubly linked* list. In a doubly linked list, each node has a pointer pointing to the next node and another one pointing to the previous node. (The "node" after the last item is NULL and the "node" previous to the first one is NULL as well.) This will require the definition of a different node class from what you used in project 3. So, you should start your project by making a Dnode class for nodes appropriate for use in a doubly linked list. Make this class template so it can hold any kind of data. You will need functions to access *next*, *previous*, and *data* as well as functions to *set_next*, *set_previous* and *set_data*. You should provide a constructor that uses default arguments that will set the pointers to NULL.

Now using this node class, develop a template list class. You will need pointers for head and tail, and these may initially be the only private variables that you need. Your class will have functions for *front_insert*, *rear_insert*, *front_remove*, and *rear_remove*. **Remember** to program incrementally. To help you with this I have written a file called **main1.cc**. When you first open this file, you will see that everything has been commented out, except the default constructor for the Dlist class. Begin by writing just the Dnode class, and then a list class with just a default constructor. Compile and run. Then uncomment the next block of code and write the implementation for the functions that are called in that block. (Note that STL classes do not have "show_all" functions, but I find it very useful to have one as you develop the class). Continue this progress, compiling and running for each block as you uncomment it. If you get a crash go back and fix it before going on. (Compile with `g++ -g main1.cc`)

The list holds dynamic memory (although the node class does not). This means that the default forms of the "Big 3" do not work correctly so you will need to define a Big 3 for this class. The destructor is much the same as one for a singly linked list, but the other two are different because in copying the list you must remember to maintain **all** the pointers (head and tail, next and previous). Work carefully and use drawings. A minor misstep can lead to major seg faults down the road. There is code that does some testing of these in the main1 file as well.

The last thing that you will develop in this process is a *bidirectional, external iterator*. You will need to make an additional class for this, a class which will closely parallel the node_iterator class I presented in lecture. You will then alter the list class to include *begin, end, r_begin,* and *r_end* functions, and finally add *insert_before* and *insert_after* functions that will each take an iterator and an item to be added, and a *remove* function that takes just an iterator as its argument.

The Application – Color Squares

Colors on a computer are frequently represented as a hexadecimal number, such as cc0099. Hexadecimal numbers are base 16 numbers and consist of the digits 0 – f. This number is actually three numbers, with the first two digits representing the intensity of red, the second two the amount of green and the third pair blue. There are 256 possible values for each color. (A total of 16,777,216 distinct colors can be represented in this way with 000000 being black and ffffff being white.)

I have provided a small class to store color swatches much like something used in a paint store. Each "swatch" consists of a hexadecimal number representing the color, and two decimal numbers representing the dimensions of the swatch in millimeters. I have also provided a data file listing a whole collection of these swatches called *swatches.txt*. There are also a couple of executables that will convert these numbers into viewable html files, as well as the application for the program, **main2.cc**. (This part will compile with `g++ -g swatch.cc main2.cc`)

This application reads swatch data from the data file and places the *predominantly green* colors at the start of the list, the *predominantly red* colors at the back of the list, and the *predominantly blue* colors at the spot immediately following the centermost spot in the list. It then:

- Makes a copy of the list using either your copy constructor or your overloaded assignment operator
- Removes the front, back and centermost swatch *from the copy*.
- Outputs the original list frontwards
- Outputs the copy frontwards.
- Outputs the original list backwards.
- Destroys the original list by alternating between removal of the first item and the last item, outputting each item as it is removed.
- Outputs the copy backwards.

Notice that there is no user interaction in this application. It simply runs the test and stops, outputting the results to the screen, one "swatch" per line with two or more blank lines between each of the outputs. Although it is not required you can see the colors by doing the following:

- Redirect the output of your program to a file: `a.out > result`
- You can compile the makinghtml program provided with this assignment (`g++ makinghtml.cc swatches.cc -o makinghtml`) and then run: ./`makinghtml result`
- This will create a .html file which you can open by double-clicking on it in your file explorer.

All code should be adequately documented and nicely formatted. Your submission should include your dnode.h, node_iterator.h, node_iterator.template dlist.h, and dlist.template.