CS202 Lab 8 - Dlists

- Jim Plank

- Directory: **~jplank/cs202/Labs/Lab8**
- Last modification date: *Tue Oct 26 18:03:37 EDT 2021*

---

dlist.cpp

Your job is to write **dlist.cpp** so that it properly implements all of the methods of the **Dlist** class, as described in the lecture notes on linked data structures.

You are *not* allowed to have any STL data structures or header files in your implementation -- you may only include **<iostream>** and "**dlist.hpp**". The implementation should be as described in the lecture notes: as a circular list with a sentinel node.

The gradescript will test all of the **src/dlist_rev_*x*.cpp** programs from the lecture notes, plus the program **src/dlist_editor.cpp** that I describe below.

You should doublecheck your **include/dlist.hpp** and make sure that it is the same one as in this lab directory. In particular, you should not use an old one that doesn't have the **Next()** and **Prev()** methods.

The list editor

This is yet another command-line program that lets me test your code. There is always a current list that is being edited. The list holds strings that are single words, and the words in a list must be unique.

The program takes a prompt on the command line (use "-" for no prompt). You can see a list of the commands by entering a question mark:

```
UNIX> echo '?' | bin/dlist_editor
CLEAR:               This calls the Clear() method, clearing the list.
DESTROY:             This deletes the list and creates a new one with new.
PRINT_FORWARD:       This prints the list, all on one line in the forward direction.
PRINT_REVERSE:       This prints the list, all on one line in the reverse direction.
PRINT_COPY:          This prints the list using a procedure which calls the copy constructor.
PUSH_BACK s:         This calls Push_Back on the string s.
PUSH_FRONT s:        This calls Push_Front on the string s.
POP_BACK:            This calls Pop_Back and prints the string
POP_FRONT:           This calls Pop_Front and prints the string.
ERASE s:             This calls Erase on the pointer to the node that holds string s.
                     If s is not on the list, this does nothing.
INSERT_BEFORE s1 s2: This calls Insert_Before(s1, d),
                     where d is the pointer to the node that holds string s2.
                     If s2 is not on the list, this does nothing.
INSERT_AFTER s1 s2:  This calls Insert_After(s1, d).
EMPTY:               This returns whether the list is empty.
SIZE:                This returns the list's size.
AO:                  This tests the assignment overload by copying to a second list,
                     and then copying back.
?:                   Print these commands.
QUIT:                Exit.
UNIX>
```

So, for example:

```
UNIX> bin/dlist_editor 'Editor>'
Editor> PUSH_BACK a
Editor> PUSH_BACK b
Editor> PUSH_BACK c
Editor> PUSH_BACK d
Editor> PRINT_FORWARD
a b c d
Editor> PUSH_FRONT z
Editor> PRINT_FORWARD
z a b c d
Editor> PRINT_REVERSE
d c b a z
Editor> POP_BACK
d
Editor> POP_FRONT
z
Editor> PRINT_FORWARD
a b c
Editor> SIZE
3
Editor> EMPTY
No
Editor> INSERT_BEFORE xxx b
Editor> PRINT_FORWARD
a xxx b c
Editor> INSERT_AFTER yyy b
Editor> PRINT_FORWARD
a xxx b yyy c
Editor> ERASE b
Editor> PRINT_FORWARD
a xxx yyy c
Editor> SIZE
4
Editor> CLEAR
Editor> EMPTY
Yes
Editor> PUSH_BACK a
Editor> PRINT_FORWARD
a
Editor> QUIT
UNIX>
```

Use COPY, DESTROY and AO to test your copy constructor, destructor and assignment overload.