Homework 1: Linked Lists Testing

There were 20 test cases. Each test was worth 3 points; to run the test cases:

1. Remove the main routine from your `main.cpp` file.
2. Append the following text to the end of your `main.cpp` file and build the resulting program.
3. For any test case you wish to try, run the program, providing as input the test number.

```
#include <ostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cassert>
#include <algorithm>
#include <csignal>
#include <regex>
#include <type_traits>
#include <set>
#include <iostream>
#include "LinkedList.h"
using namespace std;

class streambuf_switcher
{
public:
	streambuf_switcher(ios& dest, ios& src)
		: dest_stream(dest),
saved_streambuf(dest.rdbuf(src.rdbuf()))
	{}
	~streambuf_switcher()
	{
		dest_stream.rdbuf(saved_streambuf);
	}
private:
	ios & dest_stream;
	streambuf* saved_streambuf;
};

set<void*> addrs;
bool recordaddrs = false;

void* operator new(size_t n)
{
	void* p = malloc(n);
	if (recordaddrs && n == sizeof(Node))
```

```
	{
		recordaddrs = false;
		addrs.insert(p);
		recordaddrs = true;
	}
	return p;
}

void operator delete(void* p) noexcept
{
	if (recordaddrs)
	{
		recordaddrs = false;
		set<void*>::iterator it = addrs.find(p);
		if (it != addrs.end())
			addrs.erase(it);
		recordaddrs = true;
	}
	free(p);
}

void operator delete(void* p, std::size_t s) noexcept
{
	s = addrs.size();   // these two lines do nothing other
than
	s += s;               // getting rid of unused var warning
on g++
	if (recordaddrs)
	{
		recordaddrs = false;
		set<void*>::iterator it = addrs.find(p);
		if (it != addrs.end())
			addrs.erase(it);
		recordaddrs = true;
	}
	free(p);
}

void testone(int n)
{
	LinkedList empty;

	LinkedList l1;
	l1.insertToFront("9"); l1.insertToFront("8");
l1.insertToFront("7");

	LinkedList l2;
```

```
l2.insertToFront("4"); l2.insertToFront("3");

ItemType x;
switch (n)
{
default: {
      cout << "Bad argument" << endl;
} break; case 1: {
      assert(empty.size() == 0);
} break; case 2: {
      assert(l1.size() == 3);
} break; case 3: {
      assert(!empty.get(0, x));
} break; case 4: {
      assert(l1.get(0, x) && x == "7");
} break; case 5: {
      assert(l1.get(2, x) && x == "9");
} break; case 6: {
      assert(!l1.get(3, x));
} break; case 7: {
      LinkedList l3(l1);
      assert(l3.size() == 3);
      assert(l1.size() == 3);
      assert(l1.get(1, x) && x == "8");
} break; case 8: {
      LinkedList l3;
      l3 = l1;
      assert(l3.size() == 3);
      assert(l1.size() == 3);
      assert(l1.get(1, x) && x == "8");
} break; case 9: {
      LinkedList l3(empty);
      assert(l3.size() == 0);
} break; case 10: {
      l1.reverseList();
      assert(l1.get(0, x) && x == "9");
} break; case 11: {
      l1.reverseList();
      assert(l1.get(2, x) && x == "7");
} break; case 12: {
      empty.reverseList();
      assert(empty.size() == 0);
} break; case 13: {
      ostringstream strCout;
      streambuf_switcher sso(cout, strCout);
      l1.printList();
      string str = strCout.str();
```

```
            regex e("7.*8.*9"); // 7 followed by 8 followed by 9
            assert(regex_search(str, e));
      } break; case 14: {
            ostringstream strCout;
            streambuf_switcher sso(cout, strCout);
            l1.printReverse();
            string str = strCout.str();
            regex e("9.*8.*7"); // 9 followed by 8 followed by 7
            assert(regex_search(str, e));
            // make sure they didn't take off const after
printReverse
            assert((is_same<decltype(&LinkedList::printReverse),
void (LinkedList::*)() const>::value));
      } break; case 15: {
            recordaddrs = true;
            {
                  LinkedList l;
                  int oldn = addrs.size();
                  l.insertToFront("1");
                  l.insertToFront("1");
                  l.insertToFront("1");
                  l.insertToFront("1");
                  assert(addrs.size() == oldn + 4);
            }
            assert(addrs.size() == 0);
            recordaddrs = false;
      } break; case 16: {
            recordaddrs = true;
            int oldn = addrs.size();
            l1.append(l2);
            assert(addrs.size() == oldn + 2);
            assert(l1.size() == 5);
            assert(l1.get(0, x) && x == "7");
            assert(l1.get(4, x) && x == "4");
            recordaddrs = false;
      } break; case 17: {
            LinkedList l;
            recordaddrs = true;
            int oldn = addrs.size();
            l.append(l2);
            assert(l.size() == 2);
            assert(l2.size() == 2);
            assert(addrs.size() == oldn + 2);
            recordaddrs = false;
      } break; case 18: {
            l1.swap(l2);
            assert(l1.size() == 2);
```

```
            assert(l2.size() == 3);
            assert(l1.get(0, x) && x == "3");
            assert(l1.get(1, x) && x == "4");
            assert(l2.get(0, x) && x == "7");
            assert(l2.get(2, x) && x == "9");
      } break; case 19: {
            LinkedList l;
            l1.swap(l);
            assert(l.size() == 3);
            assert(l.get(0, x) && x == "7");
            assert(l1.size() == 0);
            assert(!l1.get(0, x));
      } break; case 20: {
            int oldn = 0;
            {
                  LinkedList l;
                  recordaddrs = true;
                  oldn = addrs.size();
                  l.insertToFront("1");
                  assert(l.size() == 1);
                  assert(addrs.size() == oldn + 1);
            }
            assert(addrs.size() == 0);
            recordaddrs = false;
      } break;
      }
}

int main()
{
      cout << "Enter test number: ";
      int n;
      cin >> n;
      testone(n);
      cout << "Passed" << endl;
}
```