P2-1 Doubly Linked List Implementation and Test

Building to a given specification (the Application Programmer Interface, or API) and verifying correctness of your implementation are important concerns in software engineering. APIs establish a sort of contract between the creators and users of a software library about its expected behavior and functionality. The interface also allows developers to separate the concerns of design and implementation of large projects. Early on, design can proceed without distraction from the details of implementation by instead producing a specification of required components and their interfaces. Later, the implementation of each module can proceed more independently from the design of the project as a whole.

Through the end of the course, we will explore this idea by creating and later using a C library based on a pre-specified API. In this project, you will create a Doubly Linked List library that will later be used as a component of P2-2, a handheld video game. Compliance to the specification is particularly important, so we'll take a look at automated testing as a tool for verifying implementation correctness.

Doubly Linked List Library

A doubly linked list is a set of node structures that are linked to each other in a linear chain. Each node has a pointer to the next node and a pointer to the previous node in the chain. The node at the head has a next node, but its previous node field is NULL. Similarly, the node at the tail has a previous node, but its next node field is NULL.

The detailed API for your linked list library can be found in `doubly_linked_list.h.` The functions in this module create and manipulate doubly linked lists and nodes. (e.g., inserting/deleting nodes, accessing them, creating/destroying'reversing the list). The files `doubly_linked_list.h` and `doubly_linked_list_shell.c` provide a shell implementation of this API – rename the shell code to `doubly_linked_list.c`. Note that the function `create_dlinkedlist()` and a few others are provided for you and you need to complete the remaining functions.

The .h file is the single source of truth for how you implementation should behave! Any code that later uses the library will assume that it behaves according to this API.

Automated Testing

Up to this point, your testing of projects has likely been mostly ad hoc and manual, such as trying some different inputs by hand. For this project, we introduce more powerful tools for writing automated tests. By generating a comprehensive test suite that can run automatically, you can be confident that your implementation meets the API specification. Automated tests are also useful during development, since they let you evaluate changes quickly and alert you immediately if anything breaks.

We'll be using a combination of two tools for this part of the project: Google Test Framework, a library for writing and automatically executing tests; and `make`, a build tool to ease compilation.

A simple test suite has been provided in `dll_tests_shell.cpp`. This test suite has a few cases that exercise parts of the doubly linked list implementation, but they are provided mainly as examples of how to use the framework. As you implement your doubly linked list library, you'll need to add many more tests to exercise all the edge cases for each function. This is the same framework we will use to grade your submissions, so it is to your advantage to test thoroughly.

Refer to the `Unit-testing-tutorial-dll.pdf` for information on how to add to and run the testing framework.

A makefile is provided to facilitate using the testing framework. After navigating to the directory where your P2-1 code is located, you can compile the module and the test bench by using the Unix/Linux command :

> *make*

Note: Rename files w/ "shell" in their name or you'll get an error "No rule to make target..."

Note2: Depending on your environment, you may need to include `-pthread` in `CXXFLAGS`. If you get pthread errors when you run make, edit this line in your Makefile to use the flag:

```
      CXXFLAGS += -g -Wall -Wextra -pthread
```

You can use the following command to remove all the files generated by make:

> *make clean*

Project Submission

In order for your solution to be properly received and graded, there are a few requirements. **For P2-1**: Upload the following files to Canvas before the scheduled due date:

- A zip file named dll.zip containing the following files:
  - doubly_linked_list.h
  - doubly_linked_list.c
  - dll_tests.cpp

You should design, implement, and test your own code. There are many ways to code this project. Any submitted project containing code (other than the provided framework code) not fully created and debugged by the student constitutes academic misconduct.

In addition, you must not share any code with others. This includes code written by you (or our shell code). It is a violation of the academic honor policy for this class to post any code for this class on any public forums such as github.

Happy Coding!