ECS 36C: Programming Assignment #2

Instructor: Aaron Kaloti

Winter Quarter 2022

Contents

1 Changelog

You should always refer to the latest version of this document.

- v.1: Initial version.
- v.2: Autograder details. Additional restrictions on `clInsert()` and `slotAlloc()` that probably won't affect most students.

2 General Submission Details

Partnering on this assignment is prohibited. If you have not already, you should read the section on academic misconduct in the syllabus.

This assignment is due the night of Sunday, 01/30. Gradescope will say 12:30 AM on Monday, 01/31, due to the "grace period" (as described in the syllabus). *Be careful about relying on the grace period for extra time; this could be risky.*

3 Purpose of This Assignment

- To make you familiar with a linear data structure that is similar (but not identical) to a linked list.

4 Reference Environment

The autograder, which is talked about at the end of this document, will compile and run your code in a Linux environment. That means that you should make sure your code compiles and behaves properly in a sufficiently similar environment. The CSIF, which I talk about in the syllabus, is one such environment. Do not assume that because your code compiles on an *insufficiently* similar environment (e.g. directly on your Mac laptop), it will compile on the Linux environment used by the CSIF.

*This content is protected and may not be shared, uploaded, or distributed.

You should avoid causes of undefined behavior in your code, such as uninitialized variables. If you have things like this in your code, then your code could end up generating the wrong answer when autograded, even if it generated the right answer when you tested it in a sufficiently similar environment.

5 Cursor List

In this assignment, you will implement a cursor list. I talked about cursor lists during the 01/21 lecture.

Your code for this assignment must be written in C++.

In the `cursor_list.cpp` file provided to you on Canvas, the `NUM_SLOTS` macro, `struct Slot`, and the `SLOTS` array are defined. **You are NOT allowed to modify the definitions of these.** For instance, you cannot add a member to `struct Slot`.

You are NOT allowed to use any of the following:

- Pointers.
- Dynamic memory allocation.
- Global variables, except:
  - The `SLOTS` array provided to you.
  - You can add *one* global variable of type `bool`.
- Any STL container, such as `std::vector`.

Compliance with the above restrictions will be checked manually, and violating any of them may result in a heavy penalty, *perhaps even an automatic score of zero points.*

The *only* file that you will submit for this assignment is `cursor_list.cpp`. On Canvas, you are provided with `cursor_list.hpp` and a skeleton version of `cursor_list.cpp` meant to help you get started. Since the autograder will ignore your `cursor_list.hpp` file if you submit it, you are effectively *not allowed* to modify this header file.

In this assignment, you will implement a cursor list by implementing all of the functions that are declared in `cursor_list.hpp`. Each function declaration in `cursor_list.hpp` is preceded by a block comment explaining what the function does.

In cases in which an exception needs to be thrown, the autograder will not check the exact message that you use, but the message should be sensible.

Update: Whenever a node is allocated, it *must* come from the front of the free list. (This was stated during lecture, but I thought I should put it in writing.) This should only affect `slotAlloc()`, unless you have other functions that are taking nodes off of the free list (which I would not advise).

Update: For `clInsert()`, the node that is grabbed from the free list *must* be the one that eventually contains the inserted data value.

Below are examples of how your code should behave. You can find the demo files on Canvas. **In the output, there are some parts that your version need not match exactly.**

```
$ cat demo_cursor_list.cpp
#include "cursor_list.hpp"

#include <iostream>

int main()
{
    int l1 = clCreate();
    int l2 = clCreate();
    int l3 = clCreate();
    std::cout << clLength(l3) << ' ' << clIsEmpty(l3) << '\n';  // 0 1
    printSlots(0, 7);
    std::cout << "==========\n";
    clAppend(l1, 10);
    clAppend(l2, 20);
    clAppend(l3, 30);
    clAppend(l1, 40);
    std::cout << clLength(l1) << ' ' << clIsEmpty(l1) << '\n';  // 2 0
    std::cout << clFind(l1, 30) << '\n';  // 0
    std::cout << clFind(l1, 40) << '\n';  // 7
    printSlots(0, 10);
    std::cout << "**********\n";
    clInsert(clFind(l1, 10), 50);
    clInsert(clFind(l1, 10), 60);
    clPrint(l1);  // 10 60 50 40
    std::cout << "$$$$$$$$$$\n";
    clDelete(l1, 50);
    clPrint(l1);  // 10 60 40
```

```
    std::cout << clLength(l1) << '\n';  // 3
    std::cout << "@@@@@@@@@@\n";
    int l4 = clCopy(l1);
    clPrint(l4);
    std::cout << "&&&&&&&&&&\n";
    printSlots(0, 30);
    std::cout << "##########\n";
    std::cout << "Free list:\n";
    clPrint(0);
    clDestroy(l1);
    printSlots(0, 15);
    std::cout << "~~~~~~~~~~\n";
    std::cout << "Free list: (data doesn't matter)\n";
    clPrint(0);
    std::cout << "!!!!!!!!!!\n";
    clAppend(l4, 70);
    clInsert(l4, 80);  // insert at front
    std::cout << "Before reverse:\n";
    clPrint(l4);  // 80 10 60 40 70
    std::cout << "After reverse:\n";
    clReverse(l4);  // 70 40 60 10 80
    clPrint(l4);
}
$ g++ -Wall -Werror -std=c++14  demo_cursor_list.cpp cursor_list.cpp -o demo_cursor_list
$ ./demo_cursor_list
0 1
SLOTS[0]: 0 4
SLOTS[1]: 0 0
SLOTS[2]: 0 0
SLOTS[3]: 0 0
SLOTS[4]: 0 5
SLOTS[5]: 0 6
SLOTS[6]: 0 7
==========
2 0
0
7
SLOTS[0]: 0 8
SLOTS[1]: 0 4
SLOTS[2]: 0 5
SLOTS[3]: 0 6
SLOTS[4]: 10 7
SLOTS[5]: 20 0
SLOTS[6]: 30 0
SLOTS[7]: 40 0
SLOTS[8]: 0 9
SLOTS[9]: 0 10
**********
SLOTS[1]: (header)
SLOTS[4]: 10
SLOTS[9]: 60
SLOTS[8]: 50
SLOTS[7]: 40
$$$$$$$$$$
SLOTS[1]: (header)
SLOTS[4]: 10
SLOTS[9]: 60
SLOTS[7]: 40
3
@@@@@@@@@@
SLOTS[8]: (header)
SLOTS[10]: 10
SLOTS[11]: 60
SLOTS[12]: 40
&&&&&&&&&&
SLOTS[0]: 0 13
SLOTS[1]: 0 4
SLOTS[2]: 0 5
SLOTS[3]: 0 6
SLOTS[4]: 10 9
SLOTS[5]: 20 0
SLOTS[6]: 30 0
SLOTS[7]: 40 0
SLOTS[8]: 50 10
```

```
SLOTS[9]: 60 7
SLOTS[10]: 10 11
SLOTS[11]: 60 12
SLOTS[12]: 40 0
SLOTS[13]: 0 14
SLOTS[14]: 0 15
SLOTS[15]: 0 16
SLOTS[16]: 0 17
SLOTS[17]: 0 18
SLOTS[18]: 0 19
SLOTS[19]: 0 20
SLOTS[20]: 0 21
SLOTS[21]: 0 22
SLOTS[22]: 0 23
SLOTS[23]: 0 24
SLOTS[24]: 0 25
SLOTS[25]: 0 26
SLOTS[26]: 0 27
SLOTS[27]: 0 28
SLOTS[28]: 0 29
SLOTS[29]: 0 0
##########
Free list:
SLOTS[0]: (header)
SLOTS[13]: 0
SLOTS[14]: 0
SLOTS[15]: 0
SLOTS[16]: 0
SLOTS[17]: 0
SLOTS[18]: 0
SLOTS[19]: 0
SLOTS[20]: 0
SLOTS[21]: 0
SLOTS[22]: 0
SLOTS[23]: 0
SLOTS[24]: 0
SLOTS[25]: 0
SLOTS[26]: 0
SLOTS[27]: 0
SLOTS[28]: 0
SLOTS[29]: 0
SLOTS[0]: 0 7
SLOTS[1]: 0 13
SLOTS[2]: 0 5
SLOTS[3]: 0 6
SLOTS[4]: 10 1
SLOTS[5]: 20 0
SLOTS[6]: 30 0
SLOTS[7]: 40 9
SLOTS[8]: 50 10
SLOTS[9]: 60 4
SLOTS[10]: 10 11
SLOTS[11]: 60 12
SLOTS[12]: 40 0
SLOTS[13]: 0 14
SLOTS[14]: 0 15
~~~~~~~~~~
Free list: (data doesn't matter)
SLOTS[0]: (header)
SLOTS[7]: 40
SLOTS[9]: 60
SLOTS[4]: 10
SLOTS[1]: 0
SLOTS[13]: 0
SLOTS[14]: 0
SLOTS[15]: 0
SLOTS[16]: 0
SLOTS[17]: 0
SLOTS[18]: 0
SLOTS[19]: 0
SLOTS[20]: 0
SLOTS[21]: 0
SLOTS[22]: 0
SLOTS[23]: 0
```

```
SLOTS[24]: 0
SLOTS[25]: 0
SLOTS[26]: 0
SLOTS[27]: 0
SLOTS[28]: 0
SLOTS[29]: 0
!!!!!!!!!!
Before reverse:
SLOTS[8]: (header)
SLOTS[9]: 80
SLOTS[10]: 10
SLOTS[11]: 60
SLOTS[12]: 40
SLOTS[7]: 70
After reverse:
SLOTS[8]: (header)
SLOTS[7]: 70
SLOTS[12]: 40
SLOTS[11]: 60
SLOTS[10]: 10
SLOTS[9]: 80
```

6 Grading Breakdown

As stated in the updated syllabus, this assignment is worth 6% of your final grade.

7 Autograder Details

If you haven't already, you should read what I say in the syllabus about the Gradescope autograder.

As mentioned above, the *only* file that you should submit to Gradescope is `cursor_list.cpp`. Any other file that you submit will be ignored. The autograder will compile your code using the same flags used in the examples above.

Your output must match mine *exactly*.

The autograder will grade your submission out of 54 points.

`clReverse()` will be graded manually (after the deadline) and not by the autograder. It will be graded out of 6 points, bringing the total points to 60.

Even if you do not implement all functions correctly, you still should be able to get points for some of the test cases. For example, if you implement all functions correctly except for `clCopy()`, then you still should get a good score, since most of the test cases do not use `clCopy()`. However, there are some core functions (e.g. `clPrint()`, `clInsert()`) where if you do not implement them properly, you will lose a lot of points. There is no way to design an autograder that avoids this. After all, if inserting into your cursor list does not even work, then how can much else be tested?

7.1 Released Test Cases vs. Hidden Test Cases

As mentioned in the syllabus, you will not see the results of certain test cases until after the deadline; these are the hidden test cases, and Gradescope will not tell you whether you have gotten them correct or not until after the deadline. One purpose of this is to promote the idea that you should test if your own code works and not depend solely on the autograder to do it. Unfortunately, Gradescope will display a dash as your score until after the deadline, but you can still tell if you have passed all of the visible test cases if you see no test cases that are marked red for your submission.

7.2 Specific Details About the Test Cases

See `visible_cases.cpp` on Canvas.

UCDAVIS
COMPUTER SCIENCE