Lab 05: Linked Lists

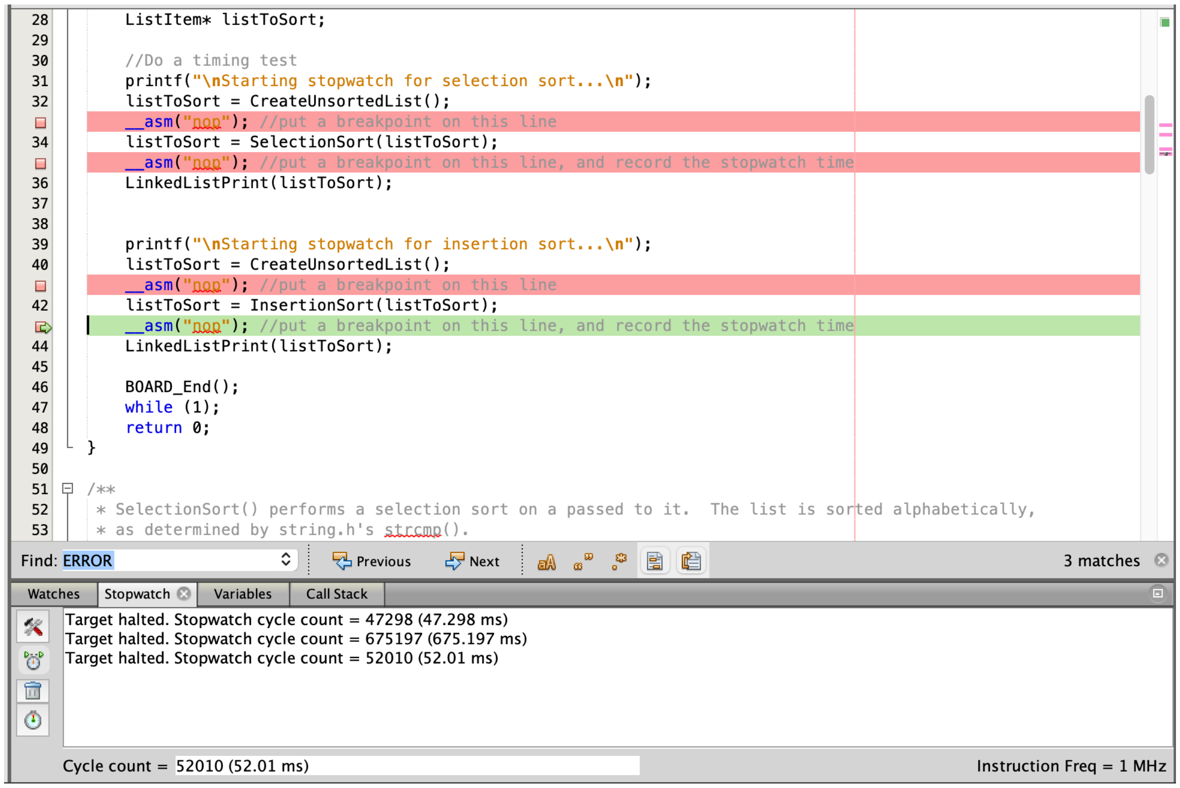
ECE 13L

Mathias Eriksen

Collaborated with: Rikuu Mikami

Due Date: 27 October 2022

Timing Experiment:



```
    ListItem* listToSort;

    //Do a timing test
    printf("\nStarting stopwatch for selection sort...\n");
    listToSort = CreateUnsortedList();
    __asm("nop"); //put a breakpoint on this line
    listToSort = SelectionSort(listToSort);
    __asm("nop"); //put a breakpoint on this line, and record the stopwatch time
    LinkedListPrint(listToSort);


    printf("\nStarting stopwatch for insertion sort...\n");
    listToSort = CreateUnsortedList();
    __asm("nop"); //put a breakpoint on this line
    listToSort = InsertionSort(listToSort);
    __asm("nop"); //put a breakpoint on this line, and record the stopwatch time
    LinkedListPrint(listToSort);

    BOARD_End();
    while (1);
    return 0;
}

/**
 * SelectionSort() performs a selection sort on a passed to it.  The list is sorted alphabetically,
 * as determined by string.h's strcmp().
```

```
Target halted. Stopwatch cycle count = 47298 (47.298 ms)
Target halted. Stopwatch cycle count = 675197 (675.197 ms)
Target halted. Stopwatch cycle count = 52010 (52.01 ms)
```

```
Cycle count = 52010 (52.01 ms)                    Instruction Freq = 1 MHz
```

Fig 1. Timing experiment output

The time for the selection sort was Stopwatch cycle count = 47298 (47.298 ms).

The time for the insertion sort was Stopwatch cycle count = 52010 (52.01 ms).

My results for this experiment were not what I expected, as the selection sort was faster than the insertion sort. From the lab doc, we should expect the insertion sort to be faster than the selection sort, but this was not the case in practice. I implemented both functions using the provided pseudocode, so this issue could be with my debugger or simulation settings. The issue could also be something to do with my hardware. Also, maybe some of my functions from LinkedList.c that are used in insertion sort are slowing down the sort more than they should . But, insertion sort should be faster since it allows you to exit the inner loops without having to iterate through every item in them, since you can break out. The linked list is very helpful here since it is easy to move through the list, and move items around within it, as well as inserting and swapping data easily while preserving the integrity of the list.

Summary:

This lab consisted of creating functions which allow a user to create a doubly linked list, and manipulate the items within a doubly linked list. We were also asked to implement an alphabetical sort of a doubly linked list. The lab consisted mainly of using structs and pointers in order to properly link items within the list. It was also important to understand the basic concepts in c, like how to use a function, and incorporate loops. Finally, another big part of the lab was just understanding the general concept of a linked list, and visually picturing where each pointer should be going within the list.

Methods:

My approach to this lab was to start by researching linked lists, and the implications of a doubly linked list. Once I had a good understanding of how the doubly linked list works in concept, I took a look at the data structure defined in the header file. This was the ListItem structure, and the important takeaway for me is that each item simply stores three pointers, the pointer to the next item, previous item, and the string contained within the item. I had some trouble with the syntax of pointing to items, but once you get used to it it's quite easy to use. Next, I also had some trouble with the more complicated pointer fixing in the create after function, but drawing everything out on paper was very helpful. In my opinion, only the create after function was difficult in the LinkedList.c file, the rest of the functions were very simple for me to implement. They were made a lot easier by writing the LinkedListGetFirst function first, since it is really useful in some of the other functions. I tested each function thoroughly in the test harness, and it was a lot easier for me to just test using the LinkedListPrint function, so most of my tests use this method. Finally, implementing the sort.c functionality was easy for me using the provided pseudocode, but it was useful practice in implementing pseudocode. The extent of my collaboration was helping out a fellow student, Rikuu, with some of the major concepts of the lab, and general debugging help.

Results:

The result of this lab was a function library which allows the user to create a doubly linked list, add items to the list in whatever spot is specified, print the whole list, swap data in the list, and get the first and last items in the list. I ended up using around 8 hours to complete this lab, and the lab was quite enjoyable except for some issues.

Feedback:

The main issue I have with this lab was in the LinkedListCreateAfter function. The header file states that the function should "*If passed a NULL item, CreateAfter() should still create a new ListItem, just with no previousItem.*". Since the case where the item data is null is explicitly stated later, this is not mistyped, and does not refer simply to the data of the item being null. My interpretation is that this says the function should be able to create a new item at the front of the list (no previous item) if passed a NULL address. I spent quite some time contemplating how this could be implemented. But, since a NULL addressed item has no links to any list, I cannot see how this item could be linked to any existing list. If the address passed is NULL, the item at NULL has no nextItem which can be set to the new item's next item. So, I cannot understand how this could be implemented. If possible, I would like to know how we were expected to handle this case.

Lab doc also states on page 11 "You can achieve something similar by removing an item and using CreateAfter() to insert it back in , but even this is not a true insert operation because it cannot insert an item at the beginning of a list."

In my function I just return NULL in this case since I do not know how this new item would be linked within the list.

I also had some issues with the timing part of the lab within sort.c. I implemented the pseudocode for insertion sort line for line and had a TA take a look at the code, and they could not find anything wrong with it. But, when I run the timing test, it is still slightly slower then the selection sort, when it should be faster than it by quite a bit. Possibly this issue is with my debugger or simulation settings?