

CSE30 - HW #3

Linked Lists - To Infinity and Beyond!

Due February 21, 2014 by 6:00pm

1 Introduction

There's a decent chance that you've implemented linked lists before. You'll probably implement them again. Knowing how to do so in C is a good exercise in pointer manipulation and dynamic memory allocation, and gives you more practice with the language. Internalizing the logic behind linked list will enable you to create any data structure you want, just by manipulating pointers.

Please remember that this assignment is subject to the style guidelines (available online at <http://cseweb.ucsd.edu/classes/wi14/cse30-c/hw/style.pdf>).

2 Linked List (30 pts)

In this assignment, we are giving you a header file that you must use. This ensures compatibility with our testing framework. There are a total of 15 functions that you are required to implement. In addition, you must submit testing code for those functions.

Full details on the functions required are included in the header file. To get the header file run:

```
$ cp /home/linux/ieng6/cs30w/public/hw/hw3/linked_list.h .
```

from within your homework directory. This header file outlines what code you must write to implement the linked list.

2.1 Tests (12 pts)

You must also write test code in a separate file (`linked_list_test.c`) that compiles into a binary that thoroughly tests all 15 functions of your linked list. Please use `/home/linux/ieng6/cs30w/public/hw/hw3/linked_list_test.c` as the bootstrap for your test cases.

Make sure your code is well tested and you always free any memory you allocate. Try running your test using `valgrind ./linked_list_test` to be sure.

2.2 Makefile (2 pts)

In addition to the source code, you must submit a **Makefile**. This Makefile should have rules such that the following make commands work appropriately (it may have other rules if you desire):

- `make` (or `make all`): your testing code compiles, as well as an up-to-date object file for your linked list implementation (`linked_list.o`).
- `make test`: Your testing code is compiled into `linked_list_test`.
- `make clean`: All binaries and object files are removed.

2.3 Documentation (1 pt)

As always, make sure your code is well documented and explains what you're doing. Include a README file that gives a high-level overview, including how to compile and run the code. If you attempt the extra credit, make sure to mention that in the README.

2.4 ARM Assembly (10 pts)

Hand compile the method `int ll_front(const struct List *list, int * val)`. Assume that the address of "list" is present in the register `r0` and address of "val" in `r2`.

Submit the assembly code in `assembly.S`.

2.5 Extra Credit

(5 pts)

It is possible to get as much as 5 extra credit marks on this assignment by completing the following extra problem. This is also a great thing to do if you want more practice or are bored by the assignment.

Hand compile the methods

`int ll_at(const struct List *list, unsigned index, int * val)` and `int ll_back(const struct List *list, int * val)`. In both cases, you can assume that the address of "list" is present in the register `r0`, `index` is in `r1` and address of "val" in `r2`. Add this code to `assembly.S` above.

3 Submission

As soon as you're finished with the assignment, you may use the `turnin` command to submit your work. The `turnin` command accepts a single file as argument. First, compress the folder into a `tar.gz` file using the command `tar -czf hw3.tar.gz hw3` where `hw3` is the name of your homework directory. Next, execute `turnin -p hw3 hw3.tar.gz` to submit your homework.

You may submit your homework as many times as you'd like, but only the final submission will be recorded. **Make sure that all files needed for the assignment are in the directory given to turnin.** No other files will be submitted.