COMS W3157 Advanced Programming, Lab #3

Please read this assignment carefully and follow the instructions EXACTLY.

Submission:

Please refer to the lab retrieval and submission instruction.

This lab consists of multiple parts. Your code for each part must be in a subdirectory (named "part1", "part2", etc.)

If a part asks for a program, you must provide a Makefile. Please refer to lab1 for the requirements for Makefiles. The TAs will make no attempt to compile your program other than typing "make".

Please include a README.txt in the top level directory.

Checking memory errors with valgrind

You will be heavily penalized if your program contains memory errors. Memory errors include (among other things) failure to call free() on the memory you obtained through malloc(), accessing past array bounds, dereferencing uninitialized pointers, etc.

You can use a debugging tool called "valgrind" to check your program:

```
valgrind --leak-check=yes ./your_executable
```

It will tell you if your program has any memory error. See "The Valgrind Quick Start Guide" at http://valgrind.org/docs/manual/quick-start.html for more info.

You must include the output of the valgrind run for EACH PART in your README.txt. In addition, TAs will run valgrind on your program when grading.

Part 1: Implement a singly linked list (70 points)

(a)

Your job is to implement a generic singly linked list that can hold any data type. The interface has been specified and provided to you in a header file called mylist.h. So your job is to write mylist.c that implements each function whose prototype is included in mylist.h. Specifically, you are asked to write the following functions:

```
struct Node *addFront(struct List *list, void *data)

void traverseList(struct List *list, void (*f)(void *))

void flipSignDouble(void *data)

int compareDouble(const void *data1, const void *data2)
```

```
struct Node *findNode(struct List *list, const void *dataSought,
        int (*compar)(const void *, const void *))

void *popFront(struct List *list)

void removeAllNodes(struct List *list)

struct Node *addAfter(struct List *list,
        struct Node *prevNode, void *data)

void reverseList(struct List *list)
```

The header file contains detailed comments specifying the behavior of the functions. Your implementation should follow the specified behavior.

In addition, I provide you with a test driver program, mylist-test.c, which produces the following output for a correctly implemented linked list:

```
testing addFront(): 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
testing flipSignDouble(): -9.0 -8.0 -7.0 -6.0 -5.0 -4.0 -3.0 -2.0 -1.0
testing flipSignDouble() again: 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
testing findNode(): OK
popped 9.0, the rest is: [ 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0 ]
popped 8.0, the rest is: [ 7.0 6.0 5.0 4.0 3.0 2.0 1.0 ]
popped 7.0, the rest is: [ 6.0 5.0 4.0 3.0 2.0 1.0 ]
popped 6.0, the rest is: [ 5.0 4.0 3.0 2.0 1.0 ]
popped 5.0, the rest is: [ 4.0 3.0 2.0 1.0 ]
popped 4.0, the rest is: [ 3.0 2.0 1.0 ]
popped 3.0, the rest is: [ 2.0 1.0 ]
popped 2.0, the rest is: [ 1.0 ]
popped 1.0, the rest is: [ ]
testing addAfter(): 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
popped 1.0, and reversed the rest: [ 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 ]
popped 9.0, and reversed the rest: [ 2.0 3.0 4.0 5.0 6.0 7.0 8.0 ]
popped 2.0, and reversed the rest: [ 8.0 7.0 6.0 5.0 4.0 3.0 ]
popped 8.0, and reversed the rest: [ 3.0 4.0 5.0 6.0 7.0 ]
popped 3.0, and reversed the rest: [ 7.0 6.0 5.0 4.0 ]
popped 7.0, and reversed the rest: [ 4.0 5.0 6.0 ]
popped 4.0, and reversed the rest: [ 6.0 5.0 ]
popped 6.0, and reversed the rest: [ 5.0 ]
popped 5.0, and reversed the rest: [ ]
```

This model output is also provided to you in mylist-test-output.txt.

I recommend you implement the functions in the order listed, and test each function as you go. You can start by commenting out the code in main() of mylist-test.c and uncomment the code one block at a time to test each list function you implemented, comparing your output with that of mylist-test-output.txt. The ’diff’ UNIX command may come in handy.

Note that mylist-test.c may not test every single function. You are still responsible for correct implementations of all functions.

Don’t forget to run valgrind at each step to make sure you don’t have a memory bug, and don’t forget to include the valgrind output in your README.txt when you’re done.

(b)

Modify your Makefile to produce a static library named 'libmylist.a' that contains your linked list object files. Your test program, mylist-test, must link with the library file, not the mylist.o file.

You can learn how to make a library file here:

    http://randu.org/tutorials/c/libraries.php

Note that we are making a static library, not a shared library.

Part 2: Using the linked list library for strings (30 points)

In this part, you will use the linked list library that you implemented in part1 to write a program called 'revecho' that simply prints out the command line arguments in reverse order. In addition, it will look for the word "dude" (case-sensitive) among the command line arguments you passed, and report whether it's there or not.

For example,

```
    ./revecho hello world dude
    dude
    world
    hello

    dude found
```

Another example:

```
    ./revecho hello world friend
    friend
    world
    hello

    dude not found
```

Here are the program requirements and hints:

- Your program should simply put all the argument strings into a list WITHOUT duplicating them. There should be no malloc in your code. Just call addFront() for all strings.

- To print out the strings, you can either use traverseList() or you can traverse the list by yourself by following the next pointers, printing out each string.

- Don't forget to initialize the list and remove all nodes at the end to prevent memory errors or leaks. Make sure you include valgrind output in your README.txt.

- To find 'dude', you can either traverse the list yourself, or use findNode(). Either way, strcmp() function will come in handy. If you want to pass strcmp to findNode(), you will have to cast it to

the correct function pointer type because the signature of strcmp() is slightly different from the signature of 'compar' argument of findNode(). K&R2 section 5.11 has an example of casting a function pointer.

- You must use libmylist.a that you built in part1 from the part1 directory. Do not copy any files from part1 directory to part2 directory. The part2 directory should contain only 2 files: Makefile and revecho.c. (You will have intermediate files while you're testing obviously.) Your Makefile must reference "../part1" as the directory to look for mylist.h and libmylist.a. "-I" option and "-L" option do the trick, respectively. (The gcc man page tells you about the options.) If you modeled your Makefile after the lab1 solution as I recommended, you can put the options into INCLUDES and LDFLAGS, respectively.

  The directory that contains mylist.h and libmylist.a should be expressed in a relative path from the directory containing the part2 Makefile. The path should NOT include "~" anywhere. The path should NOT begin with "/". If you do not follow this instruction, your code will fail to build when the graders try to build your code in their home directories. These requirements apply to all future labs that uses the mylist library.

  Note that all you need to use the linked list is the header file and the library file (that are residing somewhere else). This is in fact what it means to use a 3rd party library in your code.

--

Good luck!