CSCB09 - Lab 5: Memory Leaks

Introduction

This week, we'll practice finding and fixing `malloc`-related memory leaks.

Remember, the TAs are there to help you. You should work in pairs and are welcome to ask other students for help too.

The C programming concepts you will be using in the lab today are:

- `free`
- `malloc`
- pointers

Preparation

Copy the files from `/courses/courses/cscb09w19/nizamnau/labs/lab5/` into a new directory.

There are three .c files, one .h file and a makefile. Run `make` to compile the provided code.

Understanding the code

The file `list.c` implements functions for working with linked lists very similar to the code you have seen in Lab 4. The associated `struct` is defined in `list.h`. `list.c` provides a function to add a new node at the front of a list (`add_node`), which is similar to the function you implemented last week in Lab 4. It also provides a function `tostring` that concatenates all the values stored in the nodes of a list and returns the resulting string. More interestingly, it also provides a function `remove_node` for removing a node from a list.

Take a look at list.c and list.h and make sure you understand the structure of a node and particularly the `remove_node` function.

The remaining two .c files implement the main program (in `testlist.c`) and a function to test the list code (in `test1.c`).

Fixing Memory Leaks

Remember from class that all memory that you allocate through malloc remains allocated until your program stops running, unless you explicitly use a call to `free` to release this memory. If there is memory that you acquired through malloc, but that your program is not using anymore, this memory should be freed. Also, remember that memory that has been allocated through malloc, but has never been freed, and that cannot be accessed anymore by a running program, because the program has not kept any pointers to it, is called a memory leak.

There are some memory leaks in `list.c`. Find them and fix them by adding the appropriate `free` call(s). You'll also have to create a function in `list.c` that deallocates all nodes of a list; call this function as part of your memory cleanup in `test1.c`.

There is a program called `valgrind` that you can use to check to see if you have fixed the memory leaks. Run it on your program as `valgrind ./testlist`. Be careful: it only reports on the parts of the code that ran, so if you aren't testing the functions with memory leaks, it won't report them. Your goal is to get a message like the following from `valgrind`:

```
==7685== All heap blocks were freed -- no leaks are possible
```

Adding a New Test

The test function in `test1.c` does not test the removal of nodes. Add a new test function `test2` that creates a test case similar to the one in test1.c, but also tests the function `remove_node`. The new function `test2` should be implemented in a separate file called `test2.c`. Add the call to `test2` to `testlist`. Take a look at the makefile and modify it so that it will include and compile the new code, including your new `test2`.