# CMSC 212 Discussion exercise, Wednesday, October 28 Fall 2009

1. You'll find a compressed tarfile `discussion17.tgz` in our directory `212public/discussions` that you created a symbolic link to. Copy it to your `212` directory and `cd` there.

2. Extract the files using `gtar -zxvf discussion17.tgz`, which will create a subdirectory `discussion17`, so `cd` there.

3. The tarfile contains four programs that use singly–linked lists. They all use the following `Node` structure declaration:

```
typedef struct node {
  struct node *next;
  int data;
} Node;
```

4. Use `make` to build the programs (just running "`make`" will create them all), using the included makefile.

5. The first program will probably run without any errors, but just to get more practice using the debugger with linked structures, trace its execution even if it does work fine.

   Run the first program under gdb and set a breakpoint at line 34, right after the small test list is created. Use gdb to look at the value of the `root` pointer, and to examine the contents of all of the nodes in the list.

   Step through the loop that prints the values in the list, using `display` to view the value of `current` after each line.

   You can also step through the code that builds the list, examining it at each stage, if desired.

6. The remaining three programs should have some type of error when run. **Note**: the mistakes in the programs aren't that difficult or tricky– the intention is not that this be an exercise in trying to find clever bugs, but rather an exercise in learning how to use the debugger as a tool to find problems with linked structures. (After all, code you write might have bugs that are more subtle or harder to find!) Part of what's useful to see is what symptoms different types of program bugs will cause, and what you'll see in the debugger in different cases.

   Run each program under gdb and when it has an error use the "`where`" command, and possibly the "`up`" command one or more times, to see where in the program the problem arose. Try to see what the problem is by printing the values of the variables around that point. If you can't see what the problem is that way, run the program again, either single–stepping through it, or setting a breakpoint partway through the program and single–stepping from that point.

7. Note: just to make the examples shorter, none of the programs free memory that was allocated (that's not one of the errors that any of the programs have).

8. To get credit for the exercise, before submitting just create a file named "`BUGS`" (spelled correctly and in all capital letters as shown)and type a sentence in that file for each program that you found an error in, explaining where and what the problem was.