

\$Id: lab6c-malloc-free.mm,v 1.12 2012-02-09 19:14:37-08 - - \$  
/afs/cats.ucsc.edu/courses/cms012b-wm/Labs-cmps012m/lab6c-malloc-free

## 1. Overview

This lab introduces you to `malloc(3)`, `free(3)`, and `strdup(3)`, linked lists in C, dangling pointers, and memory leak. There are two programs in this assignment, each of which will read in data, insert the data into a linked list in sorted order, and print out the data. One program will use `double` and the other will use character strings represented as `char*`. See the `code/` subdirectory. Also, read the man page: `man -s 3 malloc`

## 2. Program `nsort` specification

As usual, the program is presented in the form of a Unix `man` page specification.

### NAME

`nsort` – sort numbers read from `stdin`

### SYNOPSIS

`nsort [-d]`

### DESCRIPTION

Reads in numbers from `stdin`, and stops at end of file. Error messages are printed for any word that is not recognized by `scanf(3)` as a number. The numbers are printed sorted in ascending order, one per line, to `stdout` using the `printf(3)` format `"%24.15g\n"`.

### OPTIONS

Options are scanned by `getopt(3)`.

`-d` Output is printed in debug format with pointers in `"[%p]"` format.

### EXIT STATUS

0 No errors were detected.

1 Syntactically incorrect input numbers were detected.

## 3. Program `lsort` specification

As usual, the program is presented in the form of a Unix `man` page specification.

### NAME

`lsort` – sort lines read from `stdin`

### SYNOPSIS

`lsort [-d]`

### DESCRIPTION

Reads in lines from `stdin`, stopping at end of file. The lines are printed sorted in ascending order to `stdout`. The output should be the same as that from `sort(1)`.

### OPTIONS

Options are scanned by `getopt(3)`.

`-d` Output is printed in debug format with pointers in `"[%p]"` format.

### EXIT STATUS

0 No errors were detected.

1 An unterminated line or a line longer than 72 characters was detected.

## 4. Preparation

Before you begin, study the code provided.

- (1) The file `code/Makefile` is complete, but no source code has been provided for the two C programs.
- (2) Read the `man(1)` pages for `malloc(3)`, `free(3)`, and `strdup(3)`.
- (3) Study the code in the `misc` subdirectory. None of the code in this directory should be directly copied into your programs, but parts of it will be adapted for use.
- (4) Study the sample code `sortlist.java` which shows you the exact algorithm you are to use for your program. It is exactly equivalent to `lsort` when you run without options. It is similar to `nsort`. Adapt its algorithm line by line.

## 5. Program `nsort` implementation

This program deals only with lists of numbers.

- (1) Study the file `scanf.c` to see how to use `scanf(3)` to read numbers from `stdin`.
- (2) Study the file `numlist.c` to see how to declare a linked list of nodes containing numbers and how to use `malloc(3)` to allocate space and how to use `free(3)` to free up the nodes when done. C does not have a garbage collector.
- (3) Print out the numbers either in ordinary format or in debug format.
- (4) Instead of inserting the new node onto the front of the list, put in the list in the proper position. See the program `sortlist.java` for the insertion algorithm.
- (5) Use the command `valgrind --leak-check=full` to check for memory leak.

## 6. Program `lsort` implementation

This program deals with lists of strings, so that each node uses two heap objects.

- (1) To begin work, `cp nsort.c lsort.c`. Change the declaration of the item field from `double` to `cstring` and add the C declaration: `typedef char *cstring;`
- (2) Look at `strlist.c` to see how strings are handled. Create a character buffer of dimension 72 (`char buffer[72]`) and read in lines. Insert each line into the list. Print an error message for any line that is not terminated by a newline character (`'\n'`).
- (3) When comparing strings in the insertion routine, use `strcmp(3)`. This works very much like `compareTo` in Java.
- (4) Use the command `valgrind --leak-check=full` to check for memory leak.

## 7. Running `lint(1)`

Make sure that neither `checksource` nor `lint(1)` complains about any of your code. However, it is acceptable if `lint` does generate the following messages for standard library functions, such as `printf`, `fprintf`, `sprintf`, or `fclose`:

function returns value which is always ignored

function returns value which is sometimes ignored

If it complains about functions like `malloc`, then you are doing something wrong. You must still fix your program if it complains about your own functions.

## 8. What to submit

Submit the files `nsort.c`, `lsort.c` and `Makefile`. Your `Makefile` should build the executables `nsort` and `lsort` from the target `all`. If you are doing pair programming, see [/afs/cats.ucsc.edu/courses/cms012b-wm/Syllabus/pair-programming](http://afs/cats.ucsc.edu/courses/cms012b-wm/Syllabus/pair-programming) and follow those instructions as well.