# MAT 4370: Programming Assignment 9

Due: Wednesday, March 6

## The Problem

For the previous homework exercise, you were asked to implement a C function that would read a line of characters from a `FILE` stream, storing it in a character array. The prototype for this function was:

```
char *readLine(char *s, int n, FILE *fp, bool *overflow);
```

You can find an implementation of this function in the file `readfile4.c` on the course website. (See the "Code Examples" section, under Homework 8.)

Although correct, this implementation has some shortcomings, including the following:

- The user of this function must be able to estimate a maximum line length.
- Characters are truncated if the maximum line length is too small.
- The number of parameters to this function makes it difficult to understand. (As a general rule, as the number of parameters increases, it is often more difficult. Imagine a function with twenty parameters!)

For this exercise, we will re-implement `readLine()` in a way which removes these shortcomings.

## What to Do

1. Re-read the sections in our book which describe `malloc()` and `realloc()`. Implement "safe" versions of each of these. The idea is that if either memory allocation or reallocation fails we want the program to stop. (See the solution `pmsort.c` to see how to do this.)

2. Re-implement `readLine`, using the following prototype:

   ```
   char *readLine(FILE *fp);
   ```

   Here's the result we want from the revised `readLine()`. If the given file pointer is at the end of file, then `readLine()` should return `NULL`. Otherwise, it should allocate just the right amount of memory—no more and no less—to store the next line of input. All characters up to and including the newline should be read and consumed; everything except the terminating newline should be stored in the allocated memory. As in the previous implementation, the string of characters should be terminated with the null character.

   If there is not enough space available to store such a string, `readline()` should abort the program.

   For efficiency, use the doubling method of allocation discussed in class. Also, for debugging purposes, your function should display the number of bytes allocated or reallocated for the character string. This should be output immediately before the function is to return.

   Demonstrate your function by using a modified version of `readfile4.c`.

## What to Submit

When your sort implementation works to your satisfaction, place it in a folder named `hw09`. (Source code only please; no other files need to be submitted.) Drag this folder onto the SUBMIT icon, found in the Applications folder.