CSE 303, Spring 2005, Assignment 3
Due: Monday 25 April, 9:00AM

Last updated: April 14

This assignment involves small C programs with pointers, arrays, structs, and I/O. There is no real need for memory management. Each problem requires an incomplete or buggy C file available on the course website.

1. **Midpoints:** `midpoint.c` contains correct C code that defines 3d-points, but you need to add 4 C functions to the file:

   - `midpoint1` should take two 3d-points (arguments of type `p3d_t`, i.e., `struct Point3D`) and return a 3d-point. The returned point should represent the midpoint between the inputs.[1] Do not call any helper functions.
   - `midpoint2` should take pointers to three 3d-points (arguments of type `p3d_t*`, i.e., `struct Point3D*`) and have return type `void`. The function should set the 3d-point the first argument points to to hold the midpoint of the points the second and third arguments point to. Do not call any helper functions.
   - `midpoint3` should be just like `midpoint1` except it should use `midpoint2` as a helper function and *not* read fields of any points directly.
   - `midpoint4` should be just like `midpoint2` except it should use `midpoint1` as a helper function and *not* read fields of any points directly.

   The next page has the expected output.

2. **Permutations:** `permute.c` contains correct C code for printing a random permuation of the first n integers (where n is a command-line argument), but you need to add 2 C functions to the file:

   - `make_array` takes an `int` (call it x), heap-allocates (using `malloc`) an array of integers of length x, initializes the array to hold $1, 2, ...x$, and returns a pointer to the array.
   - `permute_array` takes an `int` (call it *len*) and a pointer to an `int` array) where *len* is the length of the array. It moves the array's elements around to create a random permutation. Use this algorithm (the fact that it is correct is quite interesting):

     "For i=0,1,...,(len-1), pick a random number j in the range [0,i] (inclusive) and swap the i^{th} and j^{th} elements of the array."

     The helper function `randlimit` will prove useful, but note it returns an integer in the range [0,$i-1$].

3. **Typing test:** `typetest.c` contains *buggy* C code for a typing-test program. Here is what it should do: Take a filename on the command line and print a line of it. The user then has to type the line back correctly. If they make a mistake, the same line should be printed again. Otherwise, the program should print how many seconds it took the user and then print the next line of the file. The program should exit when every line of the file has been typed correctly or when the user types `C-d` (which indicates an "end-of-file" for standard input).

   The program uses several library functions (`printf`, `fprintf`, `getline`, `time`, `strcmp`). Learn about them using `man`. For `time`, you should use `man 2 time`, which tells `man` you want the C part of the manual pages (for the `time` library function), not the part that describes the `time` program.

   Fix the file so it works correctly. All the bugs are in the function body of `main`. There are 8 bugs on 8 different lines.

[1]High-school geometry fact: The midpoints' coordinates are the arguments' coordinate summed and divided by two.

Extra Credit: Make an enhanced version of the typing test in a file `typetest2.c`. It should work like the original program except it should print information about the full test before exiting. In particular, it should finish with output like the following:

```
number correct: 7
correct times: 3 4 1 1 2 10 9
number incorrect: 9
fraction correct: .44
mean correct time: 4.29
```

The first line is the number of lines correctly typed (the number of lines in the file unless the user hits `C-d`). The second is a "repeat" of how many seconds each correct line took. (Your program should *not* read the input file more than once. If you run out of room somewhere to store these numbers, you should allocate more space.) The other lines should be self-explanatory. Use `printf` to round the floating-point numbers to two decimal places.

Assessment: Your solutions should be:

- Correct C programs that compile without warnings using `gcc -Wall`.
- In good style, including indentation and line breaks
- Of reasonable size

Turn-in Instructions: Follow the link on the course website and follow the instructions there.

Expected Output for Problem 1:

```
[1,2,3]
[4,9,-7.5]
[2.5,5.5,-2.25]
[2.5,5.5,-2.25]
[2.5,5.5,-2.25]
[2.5,5.5,-2.25]
```