

Midterm 2 Review Notes

C Programming

Dr. Beeson, Spring 2009

April 7, 2009

1. Using an array, passed as a pointer, to receive an answer. For example, with `add` as in two of your assignments, what is the correct way to call `add`? Or, given the way `add` is specified to behave, when writing `add(char *a, char *b, char *ans)`, what is wrong with a line `line *ans = str1;`?
2. The indirect return mechanism in general (passing an address to receive a value). You must understand how this works, how to write functions that can be called this way, and how to call functions that work this way. `add` is an example.
3. `argc` and `argv`. You should know what they are and how to use them.
4. `sprintf` and `scanf`. Technically these were in the first half of the semester, but many students that I see in office hours don't have a clue about them, so they will be on the second midterm.
5. `printf`. Some students still have to look up `%d` in a book every time they want to print something. The midterm will contain a lot of examples of `printf`, so many that if you have to look them all up in a book you'll run out of time! You are supposed to *learn this stuff by heart*.
6. `typedef`. Know how to use it to give a simple name to a complicated type. An example was given in the matrix multiplication code.
7. `multidimensional arrays`. Know how to declare a two-dimensional (or more-dimensional) array and how to access the items stored in the array. Know how C stores the items in memory.
8. `matrix multiplication`. Study this example carefully.
9. `Recursion`. Know the definition of recursion and how to recognize if a function is recursively defined or not. Be able to write a simple recursive function. We studied, for example, the factorial function.
10. Memoization (or filling in a table). Know how to use this technique to speed up an otherwise-too-slow recursion. For example, to compute the binomial coefficients. These are the numbers in Pascal's triangle— each one is the sum of the two above. The recursion

$$choose(n, k) = choose(n - 1, k - 1) + choose(n, k)$$

is too slow, but not if you use it together with a table of values.

11. `macros`. You must know the correct syntax for defining a macro, including the `#` define part and the necessity of using lots of parentheses.
12. `#` define as used for identifiers, not for macros. What exactly does it do, and when should you use it?
13. `# include`. What should be `#` included? (header files only, NEVER C files).
14. header files. What should and should not be in them?
15. Conditional compilation using `#ifdef`.
16. Left shift and right shift

17. Reading bits
18. Bitwise and, bitwise or, bitwise negation.
19. Exclusive or
20. Setting and clearing bits
21. Using bits to represent sets
22. scope and duration; global variables and local variables
23. static variables versus static functions
24. `malloc`, `calloc`, and `free`.