C Programming

Italicized points are intuitive.

1. Use “C style” I/O, including the variants of printf and scanf.
   1. printf and scanf are essentially cout and cin, respectively.
2. *Debug defective C programs that use “C style” I/O*
3. Accepts command line arguments, prints out number of them, their string values, and # of characters in each.
   1. Look at the first program we wrote (C program)
   2. int argc, char \*argv[]
   3. Argc is the number of arguments so: printf (“%d”, argc);
   4. Argv is the string value of each
      1. Using a for loop and another iterator, you can print each character (with an ‘\n’ following the last one) and count each character
4. *Readable and consistent style*
5. Use C’s floating point and/or mixed-mode arithmetic to produce “more meaningful” answers to integer problems.
   1. 1 + 2.5 = 3.5, etc.
   2. Using a float makes it all a float
6. Use WHILE statements
   1. while ( i >= 0)  
      {  
       string1[i] = string2[i];  
       i++;  
      }
7. Use C’s rich set of assignment operators
   1. = assignment x = r
   2. \*= multiplication assignment x \*= 2 (2x)
   3. /= division assignment x /= 2 (x/2)
   4. %= remainder assignment x %= a
   5. += addition assignment x += 1 (x+1)
   6. -= subtraction assignment x -= 1 (x-1)
   7. <<= left-shift assignment etc.
   8. >>= right shift assignment
   9. &= bitwise-AND assignment
   10. ^= bitwise-exclusive OR assignment
   11. |= bitwise-inclusive OR assignment
8. Use Increment and Decrement operators
   1. n++ and n--;
9. *Use IF-THEN-ELSE statements*
   1. *THEN is implied*
10. Use logical operators and complex conditional statements
    1. && and
    2. || or
    3. ! not
    4. Add them together for complex conditional statements!
11. Use FOR statements
    1. Standard for statement, but the int i must be declared before the loop!
    2. For ( i = 0; i < 1000; i++) etc…
12. Use DO…WHILE statements
    1. Remember: “reverse for-loops”
    2. do { … } while ( x > 0);
13. Use SWITCH statements
    1. switch ( x ) {  
       case -1:  
        n++; break;  
       case 0:  
        i++; break; (etc.) }
14. Use BREAK statements
    1. See above. Works in for, do-while, while, and switches
15. Use CONTINUE statements
    1. In a do or while, jumps to the next loop. In a for, it evaluates the conditions and potentially goes again.
16. Include calls to C-supplied functions.
    1. Don’t have to declare it
17. Include user-defined function definitions other than main.
    1. Declare at beginning with ; at end.
    2. Use normally.
18. Include user-defined function definitions in more than one .c file
    1. Include “other.h”
    2. Define in other.h (do not use Class or private or public, just define the function like you would in c++)
    3. Implement in other.c
19. Include user-defined header (.h) files.
    1. See above
    2. #ifndef \_OTHER\_H\_
    3. #define \_OTHER\_H\_
    4. #endif
20. Make significant use of the random function to generate and use random integer data
    1. time\_t t;
    2. srand((unsigned) time(&t));
    3. rand();
21. *Use recursion as a design and programming tool.*
22. Given a multi-function, multi-file program, identify the scope of variable(s) with that program.
    1. Global scope – can be accessed anywhere in that file
    2. Block scope – can be accessed in that block
    3. Function scope – can be accessed in that function
    4. File scope – can be accessed in that file (like global for a specific file)
23. Declaration statements for single and multi-dimension arrays.
    1. type arrayName [ arraySize ];
    2. type arrayName2 [ x ] [ y ];
24. Use #define constants to specify array sizes in declarations and loop bounds in code accessing arrays.
    1. #define NAME value
    2. #define MAX 80
25. *Use C’s [] operator to access both single and multi-dimension arrays.*
26. *Use loops and loop indices in manipulating arrays in C programs.*
27. *Initialize arrays either in loops or in initializer lists within a declaration statement.*
28. Declare both single and multi-dimension arrays in function definitions (as parameters) and in function prototypes
    1. Herp derp.
    2. Double getAverage(int arr[10]);
29. Pass arrays as arguments to functions.
    1. Int \* myFunction()
30. *Debug programs with array access out-of-bounds* *errors*
31. Declare an enumerated type.
    1. *enum DAY {  
       Saturday,  
       Sunday = 0, etc.  
       } workday;*
32. Use an enumerated type
    1. Enum DAY today = Wednesday;
33. Declare a variable to be a pointer.
    1. Int \* p;
34. Dereference pointer variables.
    1. If p in the above example were defined (int \* p = &x)
    2. We can change it then (p = 2);
    3. X is now equal to 2
35. Use pointers to “simulate” pass-by-reference of scalar variables.
    1. See above, this can be done in functions, too.
    2. A function takes pointers, and is called using addresses
    3. We can now edit those variables in main()
36. Describe how X[i] can possibly be interpreted the same as i[X]
37. Given a function that uses a one-dimensional array, convert that function to an equivalent function by changing each array reference to an equivalent pointer dereference. Do not change the array declaration.
    1. Go through the dereferencing stuff above. It makes sense, I promise.
38. Given an expression that uses pointer arithmetic, define the value of the expression in symbolic terms using the address of the pointer variable.
    1. *http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/BitOp/pointer.html*
39. Describe how C typically allows one-dimensional arrays and pointers to be used interchangeably.
    1. The \* points to the first value of the array.
40. Use malloc to allocate enough space to hold a data structure.
    1. Int \*ptr  
       prt = (int \*)malloc(sizeof(int))
    2. It now has enough room to store an int. See?
41. Write a program that uses Standard C Library string functions to manipulate character arrays as strings
    1. Strcpy, strcat, strlen, strcmp, etc.
42. Use gdb to find, and fix, bugs in programs using arrays and pointers.
    1. *http://www.delorie.com/gnu/docs/gdb/gdb\_5.html*
43. Use structures to represent common data structures such as lists and trees.