CMPT 128 REVIEW- Remembering what you've already learned.

Here are some basic concepts that you should remember from CMPT 128. Please note that this handout is in no way a complete review of CMPT128 and that you should skim the first 9 chapters of your textbook to review what was covered. This is just to get you thinking about programming structure and syntax again. If you don't know/remember how to do all this, post questions to the bulletin board - you are not being evaluated, so don't worry. ☺

1. How many bits in a byte?
2. How many bytes do each of these types have and are they signed or unsigned: short, int, long, float, double, uint8, uint16, char, void?
3. How many bytes do each of these types have and are they signed or unsigned: short*, int*, long*, float*, double*, uint8*, uint16*, char*, void*? (Note: We'll talk more about pointers in 251.)
4. How many return values can a function have?
5. What's the difference between "call by reference" and "call by value"?
6. When will the following loops exit?
   a. `for( ; ; ) {…}`
   b. `while(1) {…}`
7. What is the purpose of the statement "`break`"?
8. When should you use a `case` statement versus `if-else` statements?
9. What is the purpose of a '`;`' in a program?
10. What is a pointer?
11. What is the difference between a .h and a .cpp file? What should be found in each?
12. What is a global variable? When should you use them?
13. What is a local variable?
14. What is the meaning of the word "context"? For example, you will often hear the phrase, "This variable only exists in the context of this function."?
15. What function needs to exist in at least one of the .cpp files that is being compiled into an executable?
16. What is the terminating character of a string?
17. You are writing a program that requires you to store a series of integers. Should you store them in an array or in a vector? What is the benefit of each?
18. Recall these operators which you learned in CMPT 128: (prefix increment) ++, (prefix decrement) --, (logical NOT) !, (bitwise NOT) ~, (multiplication) *, (division) /, (modulus) %, (addition) +, (subtraction) -, (bitwise right shift) >>, (bitwise left shift) <<, (bitwise AND) &, (bitwise XOR) ^, (bitwise OR) |, (logical AND) &&, and (logical OR) ||. If they are to be used in a single assignment- what is the proper order of operations? (Hint: Check out en.cppreference.com/w/cpp/language/operator_precedence for help).
19. What do the operators << and >> do? Why is this more efficient than division/multiplication?
20. Assume that you have a 32-bit configuration register, write a code snippet that enables you to disable the 5th LSB of the register (setting it to zero) while not changing any of the other bits in the register.
21. Assume that you have a 32-bit configuration register, write a code snippet that enables you to enable the 9th LSB of the register (setting it to one) while not changing any of the other bits in the register.
22. Why do you have to have #include <iostream> at the beginning of your program?
23. How do cin and cout function?

24. What does `#define true 1` do? How is it different than a global variable? (We'll talk more about macros later).

Now you should try some programming problems- remember to actually code, compile, and test the program as most programs that compile don't actually work. ☺

1. Write a program that 1) takes a list of characters from a user, 2) stores them in an ***array***, 3) sorts them in ascending order, and 4) then prints the sorted back to the list. (Note: _Since you are storing your values in an array, you should test your program to ensure that it has a controlled method for handling the situation where a user tries to enter more input values than the maximum size of the array._ Hint: There are multiple ways to do this.)

2. Rewrite the previous program making the following changes: 1) store your input values in a ***vector***, 2) change the type of loop you use (i.e. if you used a `for` loop in part 1, use a `while` loop in this part or vice versa).

3. Review the following reminders about pointers:
   ```
   int *p; //Declares a pointer to an int
   int v; // Declares a variable of type int
   p = &v; //Assigns p the address of v
   *p = 44; // Dereferences the pointer p to assign 44 to the variable v
   void foo (int *value); //The input parameter is an int pointer-> passed as a call by
                          //value; it must be dereferenced to manipulate the int value
   void bar (int &value); //Call by reference, passes the address of the int to the
                          //function, but you don't need to dereference the value to
                          //access the int stored in the variable.
   ```

   Write simple programs to practice manipulating these different types and make sure you clearly understand what happens when you pass pointers by reference versus by value (and what happens to the values they point to in both cases) into functions. (Note: If you find this confusing, we'll talk more about this in 251.)

4. Write a program that allows the user to enter a sequence of numbers (in no particular order). When the user presses the letter 'q', the program should quit. If the user presses any other letter, the largest three values entered to date should be printed to the screen in decreasing order. Assume the user will only enter positive numbers.

5. Rewrite the program in part 4. Assume the user can enter both positive and negative numbers and that you will store the numbers with the three largest absolute values. You will quit the program when a user presses the letter 'q'. If the user presses any other letter, the values of the three largest absolute values should be printed to the screen (indicating their proper sign).

For each of these programming problems, first think about what series of tests you need to test that your programming solution is complete and works under all conditions. Be sure you write them/try them to actually prove that your program works.