Problem Set 1: Variables

Please send back to me via NYU Brightspace:

- A zip archive named as
  `PS01_<your name as Last_First>.zip`
  containing the C code files that implements all aspects of all problems.

For each problem of this problem set, you will be asked to write a short C program and answer some questions. Please submit your source code as well as a document containing your answers. Remember to use http://en.cppreference.com/w/c as a reference for how to use e.g. printf or other C library functions. Also, feel free to google if you don't know all the answers – it's part of the learning experience. Finally, please make sure that your program compiles without any errors!

Total points: 40

Problem 1 (10 points): `printf` versus `puts`

`hello.c`

An alternative to `printf` in your `hello_world.c` program is `puts("Hello, World!")`. Write a program `hello.c` that is a new "Hello, World!" program using `puts()`. What are the two main differences between `printf` and `puts`? For these reasons, `puts()` is more appropriate for printing static messages.

Problem 2 (10 points): `sizeof`

`sizeof.c`

Using `sizeof()`, we can know the size (in bytes) of a type or a value. It evaluates to a number in `size_t` type, which is an unsigned integer but whose size is platform-dependent. You can print the number using `printf()` and format string `"%zu\n"`, for example:

```
printf("sizeof char: %zu\n", sizeof(char));
```

Write a program `sizeof.c` printing the sizes of the following types and values. You will need to include `<stdbool.h>` to use `bool` type.

`char`	`int`	`short`	`long`	`long long`
`bool`	`unsigned int`	`float`	`double`	`long double`
`12345`	`123456789012`	`3.14`	`3.14f`	`"hello"`

a. Why are the sizes of 12345 and 123456789012 different?
b. Why are the sizes of 3.14 and 3.14f different?
c. What does the size of `"hello"` mean? If not sure, try with different strings.
d. Why would `bool` need that many bytes, where one bit is sufficient to represent true/false?

Problem 3 (10 points): floating point accuracy

`float.c, double.c`

Create a program `float.c` that prints a `float` value with 20 decimal places using format string `"%.20f\n"`. Using this, print the numbers 1.0, 0.5, and 0.1. Which of the printed numbers is inaccurate? Why would such a simple number like this be stored inaccurately in computer while the others are not? Create a second program `double.c` in which the variables are declared as double. What happens now?

Problem 4 (10 points): nonzero values

`truefalse.c`

We know that 0 is evaluated as false, and 1 is evaluated as true. Write a program `truefalse.c` to find out how the following values are evaluated when used as the condition of an `if` statement.

`2`	`3`	`3.14`	`0.0`	`0.0f`	`"hello"`

To do this, you can use a conditional expression inside a `printf` statement, and see what gets printed.

```
printf("%s\n", 2 ? "true" : "false");
```

What is the criteria for evaluating an `if` condition as true?