Week 4 - Introductory Programming Lab 2

Exercise 2: Variables and (simple) Math

Background

Computers would be boring if they didn't do any computation, and variables make computation a lot easier.

Variables must be *declared* before you can use them. This tells the computer to reserve a piece of memory to store the variable's value.

The computer doesn't care what you call the variable (within reason -- see **identifier** for the exact requirements), but having *descriptive variable names* will make your programs much easier to work on.

For the next few weeks, we will use two types of numeric variables: `int` (an integer), and `float` (a floating-point, i.e. decimal, number).

There is another type of variable, which is very rarely used by itself: `char`. This represents one *ASCII* character; each value (between 0 and 127) represents an entry in the **ASCII table**.

Text variables are called *strings*, short for "a string of characters". Technically, they are stored as an *array* of characters, but we will return to this in the later lab. For now, we will not do any modifications of strings.

For reasons which we will discuss in a later lab, you can get an array of a particular variable type by adding `[x]` to the variable name, where `x` is the number of variables. For *other* reasons which we will discuss later, a string's length must be 1 *more* than the number of characters you want it to hold.

To get a string of 99 characters (recommended), write:

```
char string_variable[100];
```

Technical details

Declaring and printing variables, and basic math:

```
#include <stdio.h>

int main() {
     int x;
     printf("x is currently %i", x);
     printf(", which is completely random!\n");
     printf("  (we probably don't want this)\n\n");
     x = 3;

     float y = 4.1;
     printf("y is currently %f.\n", y);
     printf("Make that shorter: %.2f.\n", y);
     printf("Make that longer: %.8f.\n\n", y);
```

```
	int z;
	z = x/y;
	z = x + z * y - y;
	printf("Why does z = %i?\n\n", z);

	printf("We can print multiple ");
	printf("variables: %i %f %i\n\n", x, y, z);

	char a = 71;
	printf("What about this one: %c\n", a);

	// wait for a keypress
	getchar();
}
```

Examine the above program carefully -- there is a lot of good information in it! Try changing the values of variables, and examine the new output.

We can also declare and print strings.

```
#include <stdio.h>

int main() {
	char name[100] = "Graham";
	printf("Hi there!  %s greets you!", name);

	// wait for a keypress
	getchar(); }
```

For now we will only declare and print strings; we won't modify them until the second lab.

Your task...

Save your introduction/story program from Exercise 1 with a new file name: we will be modifying it, but you don't want to lose all your hard work.

Modify the new version of your introduction/story so that your program:

- Declare at least one int.
- Declare at least one float.
- Declare at least one char[100].
- Use some math to change the value of a variable.
- Prints at least one int (not necessarily the first one you declared).
- Prints at least one float (not necessarily the first one you declared).
- Prints at least one string (exactly the one(s) you declared).
- You may not use any numbers within the "double-quotes" of a `printf` statement -- use variable substitution

(hint: converting units is an easy way to get some math in there. Change between miles + km, pounds + euros, pounds + kg, human years + cat years, etc. You may need to rewrite part of your self-introduction to introduce some units.)

Think about your variable names. Don't write something like:

```
distance = distance * 1.609344
```

Instead, give them more descriptive names, like

```
distance_km = distance_miles * 1.609344
```

(Read more about **good programming style**)

(Optional: invent a new measurement (like "centi-flipsies") and some fancy symbols from the **table of operators** to convert into them.)

Note: Show your work to a demonstrator/GTA.

Exercise 3: Keyboard input

Background

Reading numbers from the keyboard is fairly straightforward: declare the variable (reserve space in memory), then read data into that variable (the space in memory).

The `&` symbol means "the location of memory pointed to by the following variable". Memory management in C is not straightforward, so we postpone this discussing until a later lab. For now, just remember that you need this symbol in front of a *numeric* variable in your `scanf` statement.

Reading text is not quite as simple as numbers. As discussed in Exercise 2, we must tell the computer the maximum number of characters. This maximum value must be passed to the `scanf` function with "%99s".

Specifying the maximum size of keyboard input is *critically* important. Failure to specify the maximum size will allow the keyboard user to *overwrite random pieces of memory*. This can lead to random crashes and security flaws. Approximately 50% of all computer security holes come from this problem, which is ridiculously easy to fix.

Technical details

Reading ints and floats:

```
#include <stdio.h>

int main() {
     int a;
     printf("Enter an integer: ");
     scanf("%i", &a);
     printf("a = %i\n", a);

     float b;
     printf("Enter a float: ");
     scanf("%f", &b);
     printf("b = %f\n", b);


     // wait for a keypress
     getchar();
     // we need two of them because of scanf!
     getchar();
 }
```

Reading strings from the keyboard is more complicated. For reasons that we will discuss in a later lab, we *MUST* specify a maximum size to the `%s` scanf command. This is done with `%99s` (if the variable's size is 100 -- remember that the size must be 1 *more* than the number of characters we want to hold!).

```
#include <stdio.h>

int main() {
     // the sizes must match!
     char text[100];
     // no & for this variable
     scanf("%99s", text);

     printf("%s\n", text);

     // wait for a keypress
     getchar();
     // we need two of them because of scanf!
     getchar();
 }
```

Unlike the int and float, we must not use a & in front of a char[] variable in a scanf.

Your task...

Mad libs (similar to "ad-lib") is a popular party game in North America. Somebody writes a story, but removes a few nouns, adjectives, and verbs. Other people must supply words to replace them (often with a clue, such as "an animal name" or "a movement verb"), without knowing the context in which they would be used. The result is nonsensical and sometimes funny. For example:

One day, a ________________ went to ________ and ______________ a _______.
name of animal place verb (past) noun

(other people supply: "polar bear", "Russia", "ate", and "house".)

Write a program which implements a mad-lib game:

- Unlike traditional mad-libs, your story must use numeric values (e.g., "price in pounds, number of people") in addition to text.
- Declare at least one int, float, and string. The variable names must provide clues about their function, i.e.

```
    char noun_animal_name[100];
    int time_minutes;
```

- Ask the user to supply values for these variables.
- Read the values from the keyboard input.
- Convert at least one numeric variable into a different value.
- Print out a madlib (including numeric values).

(Optional: make it funny)

Note: Show your work to a demonstrator/GTA.