Step 2: Introducing C

This assigment is done in class on (monday)01-13-14 AND (friday)01-17-14.

This assignment is due for monday class 01-13-14 AND for the firday class 01-17-14.

Getting Started

By now you should be all logged on. You should still have a cse251 directory in your home directory from last week. If not, create one:

```
mkdir cse251
```

Please change into that directory. If you don't remember, here are the Unix commands we covered last week:

pwd	Displays the current directory	pwd
ls	List the current directory	ls ls -l ls -a -l : Include file details, -a : Include hidden files, -F : show directories
rm	Remove a file	rm filetonuke.c
mkdir	Create a new directory	mkdir cse251
rmdir	Remove a directory	rmdir cse251 Or rm -r if the directory is not empty.
cd	**Change current directory**	**cd ~/cse251**
cp	Copy a file	cp souce destination
mv	Move or rename a file	mv hello.c newname.c
cat	Display the contents of a file	cat hello.c
less	Display file contents nicely	less hello.c
man	Manual pages	man ls
yelp	Nicer help display	yelp &
top	Displays CPU usage	top q to quit
ps	Lists processes	ps u
kill	Kills a process	kill 15577

You should be able to do a pwd and see that you are in the cse251 directory at this time.

1. My First C Program

Open gedit (gedit &) at the command line. It should still be configured the way it was last week. It will come up ready for you to type a program.

First, enter the #include statement as the first line of the file:

```
#include <stdio.h>
```

This statement tells the compiler that we are using the Standard Input/Output library. If we forget it, our program will fail to compile because it won't know what we mean when we tell it to print something.

Next, enter the comment putting your name in place of (your name here):

```
/*
 * Name : (your name here)
 *
 * This is my first CSE 251 C program!!!
 */
```

This is a comment. It's a note to ourselves. It is ignored by the compiler. Comments in C start with /* and end with */. They can be a single line or multiple lines as you see here.

Now, enter the main function. This is what it looks like. We'll go into more detail on functions later on.

```
int main()
{
}
```

Save your file in the cse251 directory with the name: step2.c (File/Save As). Be sure the c is lower case.

Now compile your program by typing this command in the terminal window:

```
gcc -o step2 step2.c
```

This command compiles your program into machine executable code. If you get an error from this command, you have made a mistake in your program and will have to fix it. Please ask course staff if you need help.

> Common errors you may see:
> step2.c:3:1: error: unterminated comment - You have a /*, but no matching */
> step2.c:11: error: expected declaration or statement at end of input - You forgot a }
> We'll see many more.

Do a directory listing and you should see a file named "step2" with no extension. That's the result of your compilation. Execute the program with this command:

```
./step2
```

Now we'll make it print something. Within the main statement, add this printf statement between the curly braces of the main function:

```
    printf("My first C program\n");
```

At this time your program should look like this:

```
#include <stdio.h>

/*
 * Name : (your name here)
 *
 * This is my first CSE 251 C program!!!
 */


int main()
{
    printf("My first C program\n");
}
```

Compile and run the program. It should print "My first C program".

> The printf statement is inside the main function. To make our programs easier to read, we indent things that are inside of other things. Since printf is inside the main function, it is indented.

Add a second printf statement that prints "Aren't you impressed?\n".

You probably noticed the "\n" on the end of each thing we printed. \n is an escape character. It's a character that expresses something not in the alphabet. In this case, it expresses the idea that we want to go to a new line at that point.

> If you use the up-arrow key on your keyboard, you can repeat previous commands. This will be very handy to avoid retyping commands over and over again.

Try removing the two \n characters and running your program to see what is different. Note that you must compile before you run the program or you won't see any changes.

When done, put these two newline characters back.

2. Variables and Printing

Add these variable declarations to the beginning of your main function, before the printf statements:

```
    double radius = 7.88;
    double height = 12.231;
```

This declares two variables of type *double*. The double type is a 64 bit floating point value. It is the highest standard precision for floating point in C. I have assigned each an initial value.

The variable types we will be using today:

- double - Floating point values (64 bit, high precision)
- float - Floating point values (32 bit, low precision)
- int - Signed Integers (32 bit)

Add this code to print the values:

```
    printf("The cylinder has a radius of %f and a height of %f\n", radius, height);
```

The %f descriptor is for a floating point value. Compile and run this program. It should output:

```
The cylinder has a radius of 7.880000 and a height of 12.231000
```

This is way too many decimal points for most applications. Between the `%` and the `f` you can add modifiers to make a format string. These are (from left to right):

- - : Left justify
- + : Always put a sign (positive or negative)
- 0 : Use leading zeros
- nn : Some width value (one or more digit number).
- .n : Precision (decimal places) (one or more digit number).

For example, %+8.3f will display a floating point number that is 8 spaces wide, has 3 values after the decimal place, and will always have a sign (+ or -).

Change the printf statement to this:

```
    printf("The cylinder has a radius of %5.2f and a height of %8.1f\n", radius, height);
```

Try this. The output has a lot of space before the height. Why is that?

Some of the available descriptors are:

- %f - Floating point decimal.
- %e - Floating point expoential
- %d - Decimal
- %s - String
- %c - Character

A variable of type int can be printed using %d, but NOT %f or %e. A variable of type float or double can be printed using %f or %e, but not %d. %s is only used to print strings and %c is only used to print characters.

Now, create two integer variables named: wins and losses. Intialize wins to 11 and losses to 2. Create a printf statement to output these values so the output looks like this:

```
MSU had an 11-2 season in 2010!
```

3. Expressions

Add these variable declarations. It is a convention in the C language to keep all variable declarations together at the beginning of the function. So, put these at the front. Always declare before you use any variable.

```
    double volume, diameter, hypot;
    int games;
```

Now add this function to compute the volume:

```
    volume = radius * radius * M_PI * height;
```

Try to compile this. It will fail with a message like this:

```
step2.c: In function 'main':
step2.c:24: error: 'M_PI' undeclared (first use in this function)
```

The constant M_PI (for PI, of course) comes from the math library. To use it, we need to include the header that defines the math library. Add the following line of code to the beginning of your file right after #include <stdio.h>:

```
#include <math.h>
```

This should compile and run okay. Add code to print the value of volume.

Add this function to compute the number of games played and then add code to print the number of games played.

```
    games = wins + losses;
```

Now add these variables:

```
    double near = 10;
    double far = 22.5;
```

Add this code to compute the hypotenuse of a triangle:

```
    hypot = sqrt(near * near + far * far);
```

When you try to compile this, you'll get this error:

```
/tmp/ccrw34Js.o: In function 'main':
step2.c:(.text+0xb4): undefined reference to `sqrt'
```

This tells you that the function sqrt was not available to the compiler. There are two levels of libraries in C. There are those that are already available, like the standard input/output library, and those that have to be *linked* to your program. The math library is like that. Some simple things, like M_PI are available always, but the functions need to be made available. We do that by adding the -lm switch (add library libm, the math library) to the command to compile our programs. Use the following command to compile now:

```
gcc -o step2 -lm step2.c
```

This should work okay.

Some other functions you may find useful:

- pow(value, power) : Computes value to a given power. Example: pow(far, 2)
- sin(angle) : Computes the sine of an angle in radians
- cos(angle) : Computes the cosine of an angle in radians

Suppose I want to know how many glasses of milk I can get from a container. I would like that to be an integer, since it's a number. Add this code to your program at the appropriate places:

```
    double bottleVolume = 1700;  /* Milliliters */
    double cupVolume = 350;      /* Milliliters */
    int numCups;

    numCups = bottleVolume / cupVolume;
    printf("Number of cups: %d\n", numCups);
```

When I run this program, I get a value of 4. Yet, the actual value of 1700/350 is 4.857. But, because numCups is an integer, it is rounding down. Conversion to an integer is done with a "trunc" that removes the decimal places.

Some compilers will warn you when you try to convert a double or float to an integer like we are doing here. The reason is that we are losing precision. It's always okay to increase prevision (int to float or double), but decreasing precision should only be done if we know what we are doing. To explicitly tell the compiler to convert to an integer, we use a cast. Replace the numCups assignment with this version:

```
    numCups = (int)(bottleVolume / cupVolume);
```

The (int) forces the result to be converted to an integer. Note how I put the computation in parenthesis. Otherwise, it would have forced the value of bottleVolume to an integer, then done the division.

4. Input

The resonant frequency of an RLC circuit is determine using the following function:

$$\omega_c = \frac{1}{\sqrt{LC}}$$

In this function, L is in Henrys and C is in Farads. The result is in radians per second. There are 2pi radians in a cycle, so the conversion from radians to Hertz is f = w / 2pi. Write a program named rlc.c that inputs the capacitance in microfarads (1/1,000,000th of a Farad) and the inductance in millihenrys (1/1000 of a Henry). The program should output resonant frequency in Hertz. The program must use the scanf() function to get the capacitance in microfarads and the inductance in millihenrys from the user. The program should execute like this (also, try some other values and verify with a calculator):

```
Input Capacitance (microfarads): 30
Input Inductance (millihenrys): 30
Resonant Frequency is 167.764
```

And We Are Done!

Go to http://www.cse.msu.edu/handin. Select CSE 251 and Step 2 and hand in the file rlc.c. Then we are done for today.

CSE 251