Step 3: Flow Control and Booleans

This assigment is done in class on (monday)01-27-14 AND (friday)01-24-14.

This assignment is due for monday class 01-27-14 AND for the firday class 01-24-14.

Getting Started

By now you should be all logged on. You should still have a cse251 directory in your home directory from last week. Please change into that directory.

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

1. if Statements

We are going to work on your program from last week: rlc.c. Make a copy of that program called rlc2.c:

```
cp rlc.c rlc2.c
```

If you did not complete the program last week, there is a solution available that you can use instead. To obtain that, do the following:

```
cp /user/cse251/exercises/rlc.c rlc2.c
```

Use gedit to open this file for editing. Remember, to compile this program you'll need this command:

```
gcc -o rlc2 -lm rlc2.c
```

And to run it, you'll need this command:

```
./rlc2
```

You should compile and run this program to be sure this is all working okay.

> A trick to save typing. If you hit the up arrow on your keyboard, it will give you previously typed commands. One up arrow gives you the last command you typed, so you can run it again. Once you've typed in the commands above, you should not need to type them again if you'll just use the up arrows. Try that now to see how it works.

This program as it stands does no checking on the values entered. It won't care if you enter a zero or negative value, even though these values will cause the program to fail due to either a square root on a negative number or a divide by zero. Run the program and try some values such as zero or one of the values being negative. You'll see the answers are either -inf, meaning infinity, or -nan for "not a number". But, we don't really want to see these values. We'll fix this today, but first we'll just add some code to complain about it.

During this step, I'll be adding code to this program. Since you may have named your variables differently, you may have to modify the code I supply after you paste it in. If you are concerned about doing this, just grab my solution from the class account, instead.

Add this code right after you enter the inductance:

```
    /* Test to see if the user entered an invalid value */
    if(l < 0)
        printf("You moron, you entered a negative inductance!\n");
```

This is the most basic *if* statement. I have the keyword *if* followed by a Boolean expression in parenthesis. If the condition is true, the statement following is executed. Note that there is not a semicolon after the *if* expression, only at the end of the statement. Think of the *if* part as something stuck onto the front of a statement. Try this and ensure it works by testing with both a valid and invalid value.

See how I've indented the statement after the if. That's to make it clear to the reader that this statement is part of the conditional and only executes if the condition is true. We will always be careful to indent our programs in this course and the TA will count off for bad indentation.

We are not doing anything to prevent the invalid computation yet, only indicating to the user that they made a mistake. We'll get to that shortly as we add C capabilities.

Only one statement is allowed after the if. So, you can't just add another line of code like you would in programming languages like Python. But, we'll also get to how to do that pretty soon.

You can add an else to an if statement. Adding else adds something that is executed if the statement does not turn out to be true. Add this code right after the code you entered before:

```
    else
        printf("Okay, I guess that's reasonable\n");
```

The entire statement should look like this:

```
    /* Test to see if the user entered an invalid value */
    if(l < 0)
        printf("You moron, you entered a negative inductance!\n");
    else
        printf("Okay, I guess that's reasonable\n");
```

Test this and ensure it works okay.

Of course, there is a value you might enter that's not negative, but also not reasonable. That value is zero. We can add a test for that using *else if*. Between the first printf statement and the else, add this code:

```
    else if(l == 0)
        printf("You are really dumb, you entered zero.\n");
```

The entire if statement with the if, else if, and else parts should look like this:

```
    /* Test to see if the user entered an invalid value */
    if(l < 0)
        printf("You moron, you entered a negative inductance!\n");
    else if(l == 0)
        printf("You are really dumb, you entered zero.\n");
    else
        printf("Okay, I guess that's reasonable\n");
```

Test this and ensure it works okay. Be sure to test all three possible cases of input values.

Note the use of two equal signs: == as a test for equality. This is how we test for equal values. Don't use a single = sign here. Try changing the equality test above to a single equal sign and see what happens.

Now, do a few extra things:

Add similar code to test for invalid capacitor values.

Add code that, given a negative or zero inductance value will change the value of l to be a default value of 1. You'll need another if statement to do this. You won't be able to add this to the existing if statements just yet. The operator to test for less than or equal to in C is: <= There should not be a space between the < and = in this operator.

Then add similar code to set the capacitance value to 1 if an invalid value is entered.

2. Using Blocks and exit(1);

Our program proceeded to compute the result no matter what the user entered. Instead, we would like to exit the program if the user enters invalid data. There is a function you can use to exit a program at any time: exit(0);

First, add this new header to the beginning of your program. You'll need it to use the library function exit():

```
#include <stdlib.h>
```

Now, right after the tests for the validity of the inductance, add this line of code:

```
    exit(0);
```

If you run your program now, it should request the inductance, check its validity, then exit without computing a result. That's what the exit function does. It exits your program. Remove the exit function call now.

> The exit function accepts a single integer parameter. That is an exit code is an indication to some other process using your program of success or failure. If you call exit(0), that means you have exited successfully. If you call exit(1) that means your program exited with an error.

If we enter invalid data, we want to print an error message (which we are doing already) and we want to exit the program. Since we want to do two things for an if statement, we'll use a *block*. Replace this code:

```
    if(l < 0)
        printf("You moron, you entered a negative inductance!\n");
```

with:

```
    if(l < 0)
    {
        printf("You moron, you entered a negative inductance!\n");
        exit(1);    /* Exit with an error indicated */
    }
```

If the expression is true, the block is executed. This means each statement in the block is executed. Note that there is no semicolon after the block. A block is equivalent to a complete statement including the semicolon.

It is perfectly reasonable to have a block with only one statement and that may sometimes be easier to read:

```
    if(l < 0)
    {
        printf("You moron, you entered a negative inductance!\n");
    }
```

Make your program exit with an error code of 1 on any invalid input.

You can put any statements you like inside a block, including more if statements. Here's an example:

```
    if(bobsAge != suesAge)  /* != means "not equal" */
    {
        printf("Bob and Sue are different ages\n");
        if(bobsAge > suesAge)
        {
            printf("In fact, Bob is older than Sue\n");
            if((bobsAge - 20) > suesAge)
            {
                printf("Wow, Bob is more than 20 years older\n");
            }
        }
    }
```

Do you see what this code does? Do you see how important the indentation is to readability?

3. Boolean Expressions

Add this header to your program:

```
#include <stdbool.h>
```

This allows us to use bool, true, and false in our programs. In particular, I'm going to want to use bool in this section.

Your code to test for valid inductance values should look something like this:

```
/* Test to see if the user entered an invalid value */
if(l < 0)
{
    printf("You moron, you entered a negative inductance!\n");
    exit(1);    /* Exit with an error indicated */
}
else if(l == 0)
{
    printf("You are really dumb, you entered zero.\n");
    exit(1);
}
else
    printf("Okay, I guess that's reasonable\n");
```

Add this new variable to your program at the beginning of the main function:

```
bool valid = true;  /* Until we know otherwise */
```

We are going to keep a flag (Boolean valued variable) that indicates if the input is valid. We will assume it is until we know otherwise. Change the code for the inductance test to this:

```
/* Test to see if the user entered an invalid value */
if(l < 0)
{
    printf("You moron, you entered a negative inductance!\n");
    valid = false;
}
else if(l == 0)
{
    printf("You are really dumb, you entered zero.\n");
    valid = false;
}
else
    printf("Okay, I guess that's reasonable\n");
```

Now, where you compute the resonant frequency, change the code to this:

```
if(valid)
{
    omega = 1.0 / sqrt((l / 1000) * (c / 1000000));
    f = omega / (2 * M_PI);
    printf("Resonant frequency: %.2f\n", f);
}
```

This works just as before, but only computes the result if the valid variable is equal to true. Note that we now have a program that only computes the resonant frequency when the inductance is valid, but does not use an exit call to end the program.

Add code to your program such than when it computes the resonant frequency, it outputs the following statement:

> This frequency is one I can hear!

when the frequency is in the range 20HZ and 20000Hz inclusive.

4. Final Task

Read the instructions carefully before writing your code. Try to figure out what variables, if, and switch statements you need to create. Write a small part at a time and get it working.

i. Create a new file called tax.c. This program will compute the amount of tax owed by a taxpayer based on annual income and amount of tax exemptions.

ii. Prompt the user to enter his/her annual income.

> If you get a segmentation fault, it is probably because you did not put the & on the variable in your scanf statement: scanf("%lf", &annualIncome);

iii. If the annual income is less than $9350, the amount of tax owed is zero. Output zero and exit the program. You should get this part working before proceeding.

> Incremental Development means you write as little code at a time as you can and test it and get it working. Often it is useful to add in printf statements just to ensure a value is what you are expecting, then remove the statement when you are done.

If you see this error:

```
tax.c:13: warning: incompatible implicit declaration of built-in function 'exit'
```

You forgot to #include <stdlib.h>

iv. Prompt the user to specify filing status, either as: (1) single, (2) married, filing jointly, or (3) married, filing separately

If the user chooses option (1) or option (3), set the number of dependents at 1 and the standard deduction to 5700.
If the user chooses option (2), set the standard deduction to 11400 and prompt the user to enter the number of children. In this case, set the number of dependents to 2 + number of children. Hint: Use switch-case to implement the selection options.

v. Multiply the number of dependents by 3650 and add the standard deduction to determine the total deductions.

vi. Compute the taxable income as the annual income minus the deductions computed in (v) using the following table:

2010 Federal Income Tax Table

If taxable income is:

Over	But Not Over	The Tax Is	Of The Amount Over
$0	$16,750	$0 + 10%	$0
$16,750	$68,000	$1,675 + 15%	$16,750
$68,000	$137,300	$9,362.50 + 25%	$68,000
$137,300	**And Over**	$26,687.50 + 28%	$137,300

vii. Display the amount of taxes owed on the screen.

As an example, suppose we have married with 3 children making $75000 per year. The annual income is 75000. The number of dependents is 2 + 3 = 5. The standard deduction is 11400. Hence, the total deductions are 11400 + 5 * 3650 = 29650. The taxable income is 75000-29650 = 45350. In the table we see that this amount is over 16750, but not over 68000, so the tax is 1675 + 15% * (45350 - 16750) = 5965. The program should output 5965.

> This is a reduced tax table to shorten the project and in the real world there is a different tax table for each filing status. But, you get the idea, right?

Some other example outputs:

```
cbowen@ubuntu:~/cse251$ ./tax
Enter your annual income: 75000
What is your filing status?
1) single
2) married filing jointly
3) married filing separately
Please enter a number: 2
How many children do you have? 3
5965.00
```

```
cbowen@ubuntu:~/cse251$ ./tax
Enter your annual income: 25000
What is your filing status?
1) single
2) married filing jointly
3) married filing separately
Please enter a number: 1
1565.00
cbowen@ubuntu:~/cse251$ ./tax
Enter your annual income: 35000
What is your filing status?
1) single
2) married filing jointly
3) married filing separately
Please enter a number: 3
3010.00
cbowen@ubuntu:~/cse251$ ./tax
Enter your annual income: 100000
What is your filing status?
1) single
2) married filing jointly
3) married filing separately
Please enter a number: 2
How many children do you have? 2
10862.50
cbowen@ubuntu:~/cse251$ ./tax
Enter your annual income: 250000
What is your filing status?
1) single
2) married filing jointly
3) married filing separately
Please enter a number: 2
How many children do you have? 1
51985.50
```

And We Are Done!

Go to http://www.cse.msu.edu/handin. Select CSE 251 and Step 3 and hand in the file tax.c. Then we are done for today.

CSE 251