Introduction

Many software developers find programming to be incredibly rewarding, but also frustrating at times. Much of this frustration happens when you write a program and it either does not compile or the program does not work as expected. Today's lab will give you experience in fixing problems when writing software, as well as learning more about creating interactive C programs!

Note: As you continue to develop your programming skills, spending time planning your program *before* you sit down to write any code becomes more and more important. Often times, spending a few minutes thinking about your problem and the potential solution can save you lots of time by avoiding making mistakes.

Lab Objectives

By successfully completing today's lab, you will be able to:

- Examine and modify a program that receives input from a user using scanf()
- Create a program using variables
- Use the UNIX commands cp (or scp) to copy and rename a file
- Fix a program that has compilation errors.

Prior to Lab

- This lab corresponds with zyBooks chapter 2.

Deadline and Collaboration Policy

- This assignment is due by 11:59PM on Friday (Sept. 6, 2019).
- Use Canvas to submit your code for Exercise 1.
    - More instructions on what and how to submit are included at the end of this document.
- You should write your solutions to the exercises in this lab by yourself. It is okay to talk, at a high level, with someone else in the class about a solution. However, you should not show or discuss your code to anyone but a course instructor or lab teaching assistant.

Lab Instructions

Part 0: More C Terminology

Let's take a look at the Hello World program we wrote during the last lab, specifically at the first line.

```
#include <stdio.h>

int main()
{
        printf("Hello, world!!!!!\n");
        return(0);
}
```

What is stdio.h? In C terminology, it's a **header file**, which is essentially a way to reference using additional resources. (When you're an advanced C programmer, you will likely create your own header files to allow other programs the ability to use resources that you create!).

For the time being, just understand that by using the #include <stdio.h> statement, you are informing the C run time system to open up two files:

- stdin: standard in, which is getting input from the keyboard that's typed into the terminal window
- stdout: standard out, which is displaying information to the terminal window that's open

Part 1: Variables, Reserved Names, and Integer Arithmetic

In lecture, you've learned about **variables**, which are symbolic names for certain places in the computer's memory. When you program, you typically create variables to store values that you will retrieve and do something with in your program. Languages like C allow the programmer to name variables in a meaningful way. For instance, if I want to keep track of the number of parking tickets received on campus, I could name a variable int parkingTickets. That's a lot easier to remember than a specific memory location, which may look like 0xffff12.

In a programming language like C, every variable needs to be **declared** by listing the **type** of variable. In the example above, the variable parkingTickets is declared to be of type int, which is an integer. Always try to choose a variable name that's meaningful—it will help you keep your train of thought as you program.

Recall from lecture that you can name variables most anything, with a few exceptions:

- Variables must start with a letter or an underscore.
- They cannot use selected **reserved** words including, but not limited to, int, void, while, continue, or double.

Thus, we could create our variable to keep track of the number of parking tickets received with each of the following:

```
int parkingTickets;
int _parking_tickets;
int parking_tickets;
```

As you continue to develop your programming skills, you will likely adopt some sort of style to combine multiple words into one variable, whether you use camelCase or snake_case, which uses the underscore.

In today's lab, we will focus solely on integers. That is, these are integral numbers, or numbers without decimal points. To describe how we declare and **initialize** (or giving a value to) an integer variable, let's create a program that calculates the area of a square. For this example, we've added line numbers to our program so we can more closely follow what is happening.

```
1 #include <stdio.h>
2
3 int main (void) {
4   int squareSide = 3; //declare a variable squareSide and initialize to 3
5   int squareArea;
6
7   squareArea = squareSide * squareSide;
8   printf("the area of a square with a side of length %i", squareSide);
9   printf("is: %i \n", squareArea);
10
11  return 0;
12 }
```

In the above example, we create two variables (e.g., declare) and assign values (e.g., initialize) to the variables at different points:

- In line 4, a variable squareSide is declared and initialized to the value of 3;
  - Note that we can provide **comments** to help document what we are doing by using //
- In line 5, a variable squareArea is declared but is not initialized until line 7.

In lines 8 and 9, we use a method called printf() (e.g., print formatted) that will display on the screen the integer variables squareSide (line 8) and squareArea (line 9) using an integer placeholder %i. Note that:

- For printf statements, you can use either %i or %d as your placeholder for a variable, where both refer to a placeholder for a signed (e.g., positive or negative) integer value.
  - % is a **special character** that indicates a **placeholder**, and the i or d indicate the type of placeholder.

We are also able to save the values of basic arithmetic operators into an integer variable:

```
1 int varA = 10;
2 int varB = 5;
3 int sum = varA + var B; //sum is assigned the value of 15
4 int diff = varA – var B; //diff is assigned the value of 5
5 int product = varA * var B; // product is assigned the value of 50
6 int quotient1 = varA / var B; // quotient1 is assigned the value of 2
7 int quotient2 = varB / var A; // quotient2 is assigned the value of 0
8 //****SEE NOTES FOR WHAT HAPPENED IN #7****
```

In line 7, we are attempting to assign the value of 5/10 (or 0.5) into quotient 2. With integer arithmetic, if the result of an expression has decimal values, all of the decimal values will be lost. This process is called **truncation**.

Part 2: Basic Input and Output (I/O)

Interaction between a computer and a human allows software to adapt to different conditions. For instance, think of every time you send a text message on your phone (which is essentially a computer that fits in your pocket):

- You type or swipe on a virtual keyboard to send a message (or Emoji! 👏)
- Your phone takes the input and sends it to someone else through your carrier's network.

- The recipient's phone receives the emoji and displays it to the user.

The basic way that you will create software that interacts with a human is by using scanf() and printf() to get values from a human via the keyboard and be able to display integer variables on the screen. In the last section, we used printf() to print out the values of a basic calculation.

In Lab 1, you used \n within a printf statement to print a new line on the screen. In C, the backslash (\) is a special character that starts an **escape sequence** for putting in a new line. Escape sequences are a sequence of characters that do not represent itself when used inside of character string, but instead get translated into another character or a sequence of characters that would be hard to represent (such as a new line). In C, all escape sequences consist of two or more characters where the first character is the backslash.

Common C Escape Sequences	
\n	Newline (e.g., a line feed)
\t	Tab (horizontal)
\\	Backslash
\'	Single quotation mark
\"	Double quotation Mark

The method scanf() reads data from standard input (e.g., the keyboard), and we can also use placeholders to capture certain values. Note, however, that for scanf, %i and %d act slightly differently:

%d	Is the preferred placeholder for you to use right now because it represents a signed decimal integer in the typical base-10 counting system that you're used to.
%i	Also represents a signed decimal integer, but you can print out the integer values in other base systems such as hexadecimal (which we'll play around with in the future).

Putting it all together, here's a sample program that requests a user to enter in two values for the length of two sides of a rectangle:

```
1 #include <stdio.h>
2
3 int main( void) {
4   int w, l;  // two integer variables not initialized
5   int area = 0; // one integer variable initialized to zero
6   int perimeter = 0; // one integer variable initialized to zero
7
8   printf("Enter width, please\n");
9   scanf("%d", &w);
10  printf("Enter length, please\n");
11  scanf("%d", &l);
12
13  perimeter = w + w + l +l;
14  area = w * l;
15  printf("The perimeter of the rectangle is %i", perimeter);
16  printf("The area of the rectangle is %i", area);
17
18  return 0;
19 }
```

Couple of quick things to note:

- The ampersand (&) in lines 9 and 11 is the **address operator**. To simply put, line 9 is stating "Scan for the integer value entered by the user's keyboard and store it into the memory location where variable w is stored).
- Whenever a program finishes, it finishes in either an "okay" or "not okay" state. The statement of return 0 on line 18 is stating that the program has finished correctly.

Part 3: Putting it All Together (With More UNIX Practice!)

In this part, you will copy today's lab file from a remote location into your account. In the terminal, create a folder for your lab 2 (pro-tip: look at how you structured your lab 1 and consider using a similar structure---this will help you stay organized). Once you are there, run the following command:

```
scp /group/course/cpsc1011/public_html/f19/lab02/lab02_errors.c .
```

Below is an example of the parameters for the scp command, so you can see how it works for future use.

scp source_file_name username@destination_host:destination_folder

The scp command will securely copy from the **source** (/group/course/cpsc1011/public_html/f19/lab02/lab02_errors.c) to the **destination** (which is the currently directory represented by the ., as we discussed in last week's lab).

Lab Exercise

Your goals for today's lab are to:

1. Rename the file to be called lab02.c (using the mv command)

2. Open the file using gedit (or your other preferred text editor), and:

    a. Follow the directions in the file including formatting directions that stipulate the top of your file should have a thorough and descriptive comment header including:
        i. Your first and last name
        ii. Your CID
        iii. Your lecture and section and your lab and section
        iv. Lab 2 – Exercise 1

        Thomas Clemson
        C1234567
        CPSC 1010-003 and 1011-001
        Lab 2 – Exercise 1

    b. Fix any errors in how the program operates. Make sure any of your changes or additional code meets best practices for formatting (e.g., variables are declared at the top of the file, variables are named in meaningful ways, you indent your code consistently in a way that promotes readability).

3. Compile, run, and test your program to make sure the output is correct

When running, your program output should look like the following:

```
./a.out

Please enter the distance in miles: 1

1.00 mile(s) is equal to 1.61 kilometer(s)
```

Submission Guidelines

- Submit your lab02.c program to the Canvas system.

Grading Rubric

- If your code is not submitted successfully to the Canvas system by the due date, you will not receive credit for this lab
- Tentative point distribution
  - Program operates correctly 50 points
  - Program compiles without errors or warnings 30 points
  - Proper formatting and commenting 20 points