Prelab 1: An Introduction to Functions in C

CSE/IT 113

1 Introduction

Lab 1 focuses on functions in C. Prelab 1 provides background material on functions.

2 Requirements

In order to complete Prelab 1, you will need the following:

- An understanding of how to compile programs (Lab 0)
- An understanding of how functions work (Prelab 1)

Reading

1. Read/review Chapters 2, 3, 4, and 9.1 in *C Programming: A Modern Approach*
2. Read/review Chapters 3 and 4 in *The Linux Command Line*.

In class before the due date:

1. Answer the questions in Section 5 of this prelab, and upload a tarball of your source code files to Canvas before the due date.

3 Material

In order to complete this prelab, read the following material and answer the questions in Section 5.

Before starting this prelab, please create a new directory inside your `cse113` directory. Call this one `prelab1`. All of the files that you create for this prelab should be stored inside this directory.

If you need help creating a new directory, please refer back to the tutorial from Lab 0 that is posted on Canvas.

3.1 Functions

The basic idea of a function is to write a block of code so that it can be reused. Reusing code allows us to write a function once, and then use it in multiple places inside of our program or in other programs.

Functions should perform one and only one thing. It is the combining of the simple functions that lets you build complex programs.

A function in programming is a lot like a function in math: it takes some input and returns some output. Or put another way, functions *consume* inputs and *produce* outputs.

There are two main parts to calling a function: the caller and the callee. The caller is where the function is called, or invoked. The callee is the function that is called. Consider the following code:

```
#include <stdio.h>

/* This is called a function prototype.
   It declares the function so the compiler won't complain
   when it is used in main().  The definition of the function
   occurs below main() */
int add(int m, int n);

int main(void)
{
        int a;
        int b;
        int result = 0;

        /* grab input from the user */
        printf("calculate the sum a + b, a and b are integers\n");
        printf("enter a: ");
        scanf("%d", &a);
        printf("enter b: ");
        scanf("%d", &b);

        /* this is known as making a function call */
        result = add(a, b);
```

```
        printf("The result is: %d", result);
        return 0;
}

/*
* This is how we will write comments for functions.
* Say what the function does, what parameters (input)
* it takes in, and what it returns (output).
*/

/**
* Adds two integers
* @param m the first addend
* @param n the second addend
* @return the sum m + n
*/
int add(int m, int n)
{
        return m + n;
}
```

You should already be pretty familiar with the function declaration for `main()`:

```
int main(void)
```

The only thing special about `main()` is that when you run your code, the operating system knows to call `main()` for you. In fact, it is the entry point into your program. The main function returns a value of an `int` and in this case takes no parameters.

In the above example, a function is declared called `add(int m, int n)`. The function takes two integers as inputs, which are called formal parameters or arguments of the function. Formal parameters always occur within balanced parentheses and come right after the function name. The return value is placed before the name of the function and is a data type, e.g. `int` or `double`. This value is returned to the calling function (`main()`, in this case). The program can use or discard the return value as needed.

In our `main()` function, we declare three integers, assigning values to all of them. We then use `add()` to add two of them. When `add()` is called, the values of `a` and `b` are assigned to `m` and `n` (`int m = a` and `int n = b`). Note that the values are copied, so that even if we change `m` and `n`, `a` and `b` will not change when the function returns. This is called "pass by value". Upon successful completion of `add()`, the sum of the inputs is returned and assigned to `result`.

4 Format Strings

The first input to the function `printf()` is known as the format string (the stuff inside the quotes). Tokens are used in the format string to print variables. After the format string you give the variable(s) you want to replace the token(s) with. Tokens are replaced in the format string by position. The first token is replaced by the first variable, the second token by the second variable, etc.

What token to use to display a variable depends on the data type of the variable. To print an integer of type `int` (4 bytes of space allocated), you use the %d token. To print a floating point number of type `double` (8 bytes of space allocated) you use the %lf token. To print a new line you use the escape character (\) followed by a n: `\n`. If you want a tab between values, you use `\t`. For example,

```c
#include <stdio.h>

int main(void)
{
        int m = 3;
        double x = 77.7;

        /* print variables one per line */
        printf("m = %d\n", m);
        printf("x = %lf\n", x);

        /* or printing both on one line with a
         * tab between the variables */
        printf("m = %d\tx = %lf\n", m, x);

        return 0;
}
```

5 Questions

For all problems, use a combination of `printf()/scanf()` to grab input. Use %d for integer types (`int`) and %lf for floating point types (`double`).

Make sure you follow the naming scheme for source code files.

Create a seperate script file (see Lab 0 for details) for each program. Name your script file: *program_name*`.script`

Question 1: The interior angles of a triangle sum to 180 degrees. Given two of the triangles interior angles write a program that calculates the missing interior angle in degrees. Write a

function to find the missing angle and call the function from your main function. Find the missing angle when both given angles are 45 degrees. Also find the missing angle when the two angles are 30 and 60 degrees. Print the output to the screen. Name your source code `triangle.c`.

Question 2: A typical exercise in an algebra class is to evaluate an expression like $\frac{n}{3} + 2/3$. Write a program that evaluates this expression. Write a function to do the evaluation and that function is called from your main function. Evaluate the function for $n = 0$, $n = 1$, $n = 5$, and $n = 10$. Print the output to the screen. Name your source code `eval.c`.

Question 3: The local supermarket needs a program that can compute the value of a bag of coins. Define a function called `sum_coins`. It consumes four numbers: the number of pennies, nickels, dimes, and quarters in the bag; it produces the amount of money in the bag. Call the function `sum_coins` from your main function and test it with the following: 1 penny, 2 nickels, 3 dimes, 4 quarters. Also test it with 13 pennies, 7 nickels, 18 dimes, and 27 quarters. Print the output to the screen. Name your source code `coin.c`

Question 4: Write a program that converts dollars to euros and euros to dollars. Write two functions. Name one `euros` that consumes dollars and produces euros. And name the other `dollars`, which consumes euros and produces dollars. Call both functions from your main function. Assume the exchange rate is 1 dollar = 0.75 euros. Test your function for 1 dollar, 10 dollars and 100.25 dollars and with 0.75 euros, 10 euros, and 100.25 euros. Print the output to the screen. Name your source code `exch.c`

Question 5: Ohm's Law states that $V = I \cdot R$, where V is voltage measured in volts, I is current measured in amperes, and R is resistance measured in ohms. Knowing two of the variables you can solve for the other variable. In a single program, write three functions named `volts`, `amps`, and `ohms` that consume the other two variables as input and produce the function name as output. Test with $V = 120$, $I = 15$ and find R. Test with $I = 1$, and $R = 4000$. Test with $V = 15$ and $R = 1000$. Print the output to the screen. Name your source code file `ohm.c`

Submission

Create a tar ball of your source code and script files: Name the tarball

`cse113_firstname_lastname_prelab1.tar.gz`

For example, if your name is John Doe (note the lowercase), you would do the following:

```
$ tar zcvf cse113_john_doe_prelab1.tar.gz *.c *.script
```

And to check that the tarball contains the files you want, run the command:

```
$ tar tf cse113_john_doe_prelab1.tar.gz
```

The output for the command should be

```
coin.c
coin.script
eval.c
eval.script
exch.c
exch.script
ohm.c
ohm.script
triangle.c
triangle.script
```

Upload the tarball to Canvas before the due date.