Prelab 1: An Introduction to Functions in C

CSE/IT 113L

NMT Department of Computer Science and Engineering

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

— Martin Fowler

"I have not failed. I've just found 10,000 ways that won't work."

— Thomas Edison

"Code never lies, comments sometimes do."

— Ron Jeffries

Contents

1 Introduction

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

2 Overview

The following sections contain general information that you need in order to write your Geometry Calculator in Lab 1.

2.1 Material

In order to complete this prelab, read the following material and complete the exercises in Section 3.

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 prelab 0 that is posted on Canvas.

2.2 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:

```c
#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()`:

```c
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`.

2.3 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;
}
```

3 Prelab Specifications

This section describes what you will write and submit for this lab. **Check that you have met all of the requirements for this lab before you submit it for grading, as the lab will be counted as late if you submit or resubmit it after the due date.**

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

- An understanding of how to compile programs (Prelab & Lab 0)
- Complete the exercises in Section 3 of this prelab, and upload a tarball of your source code file and script file to Canvas before the due date.

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*.

Exercise 1 (prelab1.c, prelab1.script).
For all functions:

- Put all source code into `prelab1.c`, and call the written functions from `main()`.
- Use a combination of `printf()/scanf()` to grab input.
- Use `%d` for integer types (`int`) and `%lf` for floating point types (`double`).
- Create a script file and name it `prelab1.script` (see Lab 0 for details).

1. Write a function called `area_circle()` that calcuates the area of a circle. The radius of the circle should be `int` and the function should return a `double`. Test your function with radii `4` and `7`.

2. Write a function called `area_ring()` that calculates the area of a ring or washer. Just like in Exercise 1, the radius of the circles should be `int` and the function should return a `double`. Test with inner radius `3` and outer radius `8`.

3. A local weatherstation needs a program that converts Fahrenheit to Celsius. Write a function that does this called `f_to_c()` that takes an `int` and returns an `double`. Test with `32` Fahrenheit and `-17` Fahrenheit.

4. Write a function that converts dollars to euros called `euros()`. Use the conversion rate of 1 dollar = 0.76 euros. You will need to use `double` for both currencies. Test with `100` dollars and `12.65` dollars.

Submitting

You should submit your code as a tarball file that contains all the exercise files for this lab. The submitted file should be named (**note the lowercase**)

```
cse113_groupNUMBER_prelab1.tar.gz
```

Replace the `NUMBER` with your assigned group number and have one group member upload the tarball to Canvas before the due date.

Or if you are working along, the submitted file should be named:

```
cse113_firstname_lastname_prelab1.tar.gz
```

Upload your `.tar.gz` file to Canvas.

List of Files to Submit

Exercises start on page 3.