Lab 1: An Introduction to C Programming

CSE/IT 113L

NMT Department of Computer Science and Engineering

"The most important property of a program is whether it accomplishes the intention of its user."

— C.A.R. Hoare

"I think everyone should get a little exposure to computer science because it really forces you to think in a slightly different way, and it's a skill that you can apply to life in general, whether you end up in computer science or not."

— Tony Hsieh

"If it don't work, turn it off and turn it back on again."

— Dyllan Runions

Contents

1 Introduction

To this point, you have dealt with small C programs that printed their results to the screen. This action of printing to the screen is called *output*. There are other methods of output, such as putting data into files, writing data to the World Wide Web, sending data to a printer, and giving data to another application. Notice that all of these examples involve information or data leaving a program and going elsewhere ... *output*.

Lab 1 will deal with basic *input*. Conceptually, input is the same as output but in reverse: instead of information leaving your program, information is coming into your program from elsewhere. The information could be coming in from the keyboard, a file, the World Wide Web, or another program. Lab 1 will only address input from the keyboard.

In addition to input and output, Lab 1 will introduce the basic anatomy of a C program, as well as the very basic data types, variables, and functions.

2 Overview

The following subsections provide general information that will help you complete this lab. You **must** follow any formatting outlined in this section. For help on coding, see **Lab Specifications (Section 3)**. Remember to ask questions in class or through email if you have problems understanding the information.

2.1 File Naming Conventions

Name your source code `lab1.c`, your README file `README`, and your script file `l1-out.script`.

2.2 Programming Notes

Rectangle

In a real calculator, you would use a floating point type for the height and width of the rectangle. As we want you to practice with both input of integral and floating point types, the rectangle's length and width are of type `int`.

The function `area_rectangle()` returns a `double`.

For the function `perimeter_rectangle` simply use the following formula: $perimeterrectangle = 2 * (length + width)$. This function should return an `int`.

Circle

The radius is of type `double`.

Approximate π with the value of 3.141593 in your calculations. The simplest way to implement this would be to add the line `#define PI 3.141593` just below your `#include` statements near the top of your c file. In your area calculation use the function `pow()` from `math.h` to calculate the radius squared.

To compile add `-lm` to the end of the compile command. For example:

```
$ gcc -g -Wall lab1.c -o lab1 -lm
```

Your `area_circle` and `circumference` functions should each return a `double`.

Triangle

Both the height and base of the triangle are of type `double`.

The `area_triangle`, `hypotenuse`, and `perimeter_triangle` functions should all return a `double`.

You will have to calculate the length of the hypotenuse, using the Pythagorean theorem, before you can calculate the perimeter of the triangle. In the body of the `perimeter_triangle()` function, call `hypotenuse` to calculate the length of the hypotenuse.

Regular Polygons

The number of sides of a regular polygon is of type `int`, while the length of a side is of type `double`. **Note Bene:** You have already used the variable name `length` for your rectangle calculations. You cannot have two variables of the same name, even if they are of different types. You may want to add a prefix to the polygon side length variable to differentiate it from the rectangle side length.

The `exterior_angle`, `interior_angle`, and `area_regular_polygon` functions should all return a `double`.

As a reminder, regular polygons have n sides all of which are of equal length. Additionally, all angles of a regular polygon are equal. This leads to some nice properties.

To determine the exterior angle of a regular polygon the formula is $\frac{360}{n}$, where n is the number of sides. The angle is in degrees.

The sum of the interior angles of a regular polygon is $sum = 180(n-2)$, where n is the number of sides. The angle is in degrees. You may notice in the example code there is printed output for the sum of the interior angles, simply add a print statement in your `interior_angle()` function.

From the sum of the interior angles, it is easy to calculate what each interior angle is, simply use the following formula: $interiorangle = \frac{180(n-2)}{n}$. The angle is in degrees. The `interior_angle()` function will return this value to `main()`.

Finally, to calculate the area of a regular polygon, use the formula:

$area = \frac{s^2 \cdot n}{4 \cdot tan(\frac{\pi}{n})}$

where s is the length of a side, and n, is the number of sides. The prototype for the function `tan()` is found in `math.h`. Use the same constant value for π you used in your circle calculations.

2.3 An Approach To Coding

While the final program asks for user input, a beginner's mistake is to code the input part of the program first. **This cannot be stressed enough – this is a terrible way to start coding the program**.

Beginners want to code the input section of the program and then write the functions and test them. Experienced programmers know this is a waste of time and that input is the last thing you code. The first thing you code is the functions and you test them with hard-coded values. Only after all functions are tested and working properly is the input part of the program written.

For example, the beginner wastes all their time on input:

```c
#include <stdio.h>

int main(void)
{
        int radius = 0;

        /* beginner mistake -- starts with input before they
         * test any functions */

        printf("Enter the radius of a circle: ");
        scanf("%d", &radius);

        return 0;
}
```

A better approach to coding is to not worry about input at all. Write and test all functions that are required of you using hard-coded values:

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

/* function prototypes */
double area_circle(int r);
```

```c

int main(void)
{
        /* hard coded values for variables. These are
         * test cases with known output */

        int radius = 1;

        /* test the function with known values
         * compare the output of the function with what
         * you as a programmer expect it to be
         */

        /* note the printf is split into three lines
         * for display purposes only */
        printf("The area of circle with radius %d ", radius);
    printf("is %d\n", area_circle(radius));
        return 0;
}

/* function definitions */

/* If you are using VS Code as your text editor, snag
 * the Doxygen comments extension to make adding
 * function comments simple.
 */

/**
 * Calculates the area of a circle
 * @param r the radius of the circle
 * @return the area of the circle
 */
double area_circle(int r)
{
        return 3.141593 * pow(r, 2);
}
```

It is only after all functions are tested with hard coded values that the input (`print()/scanf()` pairs) are added to the program. Start with one function and make sure it works correctly before moving on to the next function.

3 Requirements

The following subsections describe 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.**

3.1 Geometry Calculator

In order to receive full credit for this lab, you must **write a simple geometry calculator program**. The program will read in a variety of inputs and perform some simple geometric calculations.

Exercise 1 (lab1.c, l1-out.script).
The geometry calculator program **must** have these "features":

1. In addition to `main()` you will write 10 separate functions. Use the following names for the corresponding functions:
   - `area_rectangle()`
   - `perimeter_rectangle()`
   - `area_circle()`
   - `circumference()`
   - `area_triangle()`
   - `hypotenuse()`
   - `perimeter_triangle()`
   - `exterior_angle()`
   - `interior_angle()`
   - `area_regular_polygon()`
2. Use function prototyping (see comments in the examples)
3. Instructions on requirements for each of these functions can be found in Part 2: Overview.
4. Use the functions `pow()` to calculate powers and `tan()` to find tangents.
5. With the exception of the `hypotenuse()` function, ALL functions should be called from `main()` and their results returned to `main()`. This means, at minimum, `main()` will gather input, call functions, receive answers, and print those answers.
6. Call your `hypotenuse()` function from within the `perimeter_triangle()` function.
7. Your code must have *comments*! See above and `doxygen.c` on Canvas for commenting style.
8. The source code file needs to follow the format of `doxygen.c`. You need a beginning header with your information, etc.
9. A script of your code **compiling and running** (don't forget to show it compiles!). Name the script file `l1-out.script`. Lab 0 gives directions on how to create a script.
10. a README (an example README can be found on Canvas). Name the README file, just `README`.

3.2 Sample Output

The output of your program will look as follows, where the numbers following colons represent input from the user and not output from your program.

Important: A majority of the output must be calculated by your program. That is the point of this program. Do *not* print the exact output as given here as an example of the program in operation. **Your program should perform the geometry operations based on the numerical values read in from the keyboard.**

```
Welcome to the Wile E. Coyote Geometry Calculator!

Enter the length of a rectangle as a whole (integer) number: 2
Enter the width of a rectangle as a whole (integer) number: 3

Rectangle Calculations---

The area of a rectangle with length 2 and width 3 is 6.
The perimeter of a rectangle with length 2 and width 3 is 10.

Enter the radius of a circle as a floating point number: 2.0

Circle Calculations---

The area of a circle with a radius 2.0 is 12.566371.
The circumference of a circle with a radius 2.0 is 12.566371.

Enter the height of a right triangle as a floating point number: 1.0
Enter the base of a right triangle as a floating point number: 1.0

Triangle Calculations---

The area of a right triangle with height 1.0 and base 1.0 is 0.500000.
The perimeter of a triangle with height 1.0 and base 1.0 is 3.414214.

Enter the number of sides of a regular polygon as an integer: 8
Enter the length of the side of a regular polygon as a floating point number: 5.0

Regular Polygons---

The exterior angle of a regular polygon with 8 sides is 45.000000 degrees.
The interior angles sum of a regular polygon with 8 sides is 1080.000000 degrees.
Each interior angle of a regular polygon with 8 sides is 135.000000 degrees.
The area of a regular polygon with 8 sides, each 5.0 long is 120.710678.
```

3.3 Sample Code

To help get you started, let's work with some code that takes in user input!

```c
#include <stdio.h>

/* These are function prototypes. It is a way to declare
   the function before they are used. The compiler requires that
   all functions be declared before they are used.

   You will find the definition of the functions below main()
*/

int add(int m, int n);
void print_answer(int m, char op, int n, int answer);

int main(void)
{
        int m;
        int n;
        int result;

        /*Ask user for input*/
        printf("Please enter 2 integers: ");
        scanf("%d %d", &m, &n);

        /* This calls the add function */
        result = add(m, n);

        /* This calls the print_answer function */
        print_answer(m, '+', n, result);

        return 0;
}

/* function definitions begin after the main function in this style of coding */

/**
* Adds two integer variables
* @param a the first addend
* @param b the second addend
* @return the sum a + b
*/
int add(int a, int b)
{
        return a + b;
}

/**
* Prints out the integer equation and answer
* @param a the first integer input
* @param op the operator
```

```
49  * @param b the second integer input
50  * @param answer the answer to the equation
51  * @remarks note that this comment is missing the @return
52  * tag as void functions do not return a value. Therefore
53  * the @return tag is unnecessary.
54  */
55  void print_answer(int a, char op, int b, int answer)
56  {
57          printf("The answer to %d %c %d is %d.\n", a, op, b, answer);
58  }
```

In order to read in numbers from the user, we use the function `scanf()`. It works similarly to `printf()`:

```
1  scanf("%d %d", &m, &n);
```

The first parameter to `scanf()` is a format string, similar to the one used in `printf()`. The format string in the above example means that we are scanning *standard input* (what you type on the terminal) for two integers (`%d %d`) and saving them into the variables `m` and `n` respectively (`&m, &n`). The ampersand (&) is important when using `scanf()`. If you forget it, you will get errors when you try to compile your program. The & operator says to use the *address* of the variable, not its contents. For the time being, just accept the fact we need an ampersand to make `scanf()` work.

For input of floating point numbers from the user, you will use the token %lf (again just like what you do for `printf()` to print `double` data types). For example, to enter two floating point numbers, your `scanf()` statement will look something like this:

```
1  double x;
2  double y;
3  printf("Enter two floating point numbers: ");
4  scanf("%lf %lf", &x, &y);
```

4 README

Exercise 2 (README).
Every lab you turn in will include a README file. The name of the file is README with no file extension.The README will always include the following two sections *Purpose* and *Conclusion*:

- **Purpose:** describe what the program does (i.e. what problem it solves). Keep this brief.
- **Conclusion:**
  - What did you learn. What new aspect of programming did you learn in this lab? This is the portion of the lab where you want to be analytical about what you learned.
  - Did the Pair Programming prelab help you in solving this solo lab?
  - Did you encounter any problems? How did you fix those problems?
  - Did your code still contain bugs in your final submission? If so, list those bugs.
  - What improvements might you make? In other words, if you were to refactor your lab code, what would you update?

The conclusion does not have to be lengthy, but it should be thorough.

Some labs will require a special **Pseudo-code** section. If a lab requires a pseudocode section, it will be explicitly mentioned in that lab.

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_firstname_lastname_lab1.tar.gz`

If you require a refresher on generating a tar archive file, instructions can be found in Section 6 - Creating Tar archives of Prelab 0.

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

List of Files to Submit

Exercises start on page 4.