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

Unix Command of the Week

Read the man page on `wc`. Find out how to use it and what it does. In particular, learn what the flags `-l, -c, -w` do. You will be quizzed on this!

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 `lab1.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 height and width are of type `int`.

The function `diagonal_rectangle()` returns a `double`. You will have to include the standard library `math.h` to use the `sqrt()` function to calculate the diagonal of a rectangle. To compile add `-lm` to the end of the compile command. For example:

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

Circle

The radius is of type `double`.

Approximate π with the value of 3.141593 in your calculations. In your area calculation use the function `pow()` from `math.h` to calculate the radius squared.

Triangle

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

Variables cannot be named the same, even if they are different types. Height is used in both the rectangle and triangle calculations. You may want to use a prefix so you can distinguish between the height of the rectangle and the height of the triangle. For example:

```
1  int rect_height = 1;
2  int rect_width = 1;
3  double tri_height = 1.0;
```

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`.

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.

From the sum of the interior angles, it is easy to calculate what each interior angle is: $interior\_angle = \frac{180(n-2)}{n}$. The angle is in degrees. It is not necessary that you create a separate function for this, but feel free to if it is easier for you.

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, n, is the number of sides. The prototype for the function `tan()` is found in `math.h`. Use the same value for π you used for the 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**.

Beginner's 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 his/her time on input:

```c
#include <stdio.h>

int main(void)
{
        int radius = 0;

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

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

        return 0;
}
```

Rather the correct way to code 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
 6
 7  int main(void)
 8  {
 9          /* hard coded values for variables. These are
10           * test cases with known output */
11
12          int radius = 1;
13
14          /* test the function with known values
15           * compare the output of the function with what
16           * you as a programmer expect it to be
17           */
18
19          /* note the printf is split into three lines
20           * for display purposes only */
21          printf("The area of circle with radius %d ", radius);
22          printf("is %d\n", area_circle(radius));
23          return 0;
24  }
25
26  /* function definitions */
27
28  /* write comments as you write the function, you will
29   * not want to comeback and write them */
30
31  /**
32   * Calculates the area of a circle
33   * @param r the radius of the circle
34   * @return the area of the circle
35   */
36  double area_circle(int r)
37  {
38          return 3.141593 * pow(r, 2);
39  }
```

It is only after all functions are tested with hard coded values that the input (`print()`/`scanf()` pairs) is 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, as the lab will be counted as late if you submit or resubmit it after the due date.**

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, lab1.script, README).
The geometry calculator program **must** have these "features":

1. In addition to `main()` you will write 11 separate functions. Use the following names for the corresponding functions:

   ```
   area_rectangle()

   perimeter_rectangle()

   diagonal_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. Use the functions `pow()` to calculate powers and `tan()` to find tangents.

4. Call your `hypotenuse()` in `perimeter_triangle()`.

5. Your code must have *comments*! See above and `doxygen.c` on Canvas for commenting style.

6. The source code file needs to follow the format of `doxygen.c`. You need a begininng header with your information, etc.

7. A script of your code **compiling and running** (don't forget to show it compiles!). Name the script file `lab1.script`. Lab 0 gives directions on how to create a script.

8. 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 the numbers following colons represent input from the user and is 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 Wile E. Coyotes Geometry Calculator!

Enter the height 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 height 2 and width 3 is 6

The perimeter of a rectangle with height 2 and width 3 is 10

The length of the diagonal of a rectangle with height 2 and width 3 is 3.605551

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

Circle Calculations---

The area of a circle with radius 2.0 is 12.566371

The circumference of circle with 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 of a regular polygon with 8 sides sums to 1080.000000
↪ degrees.

Each interior angle of a regular polygon with 8 sides is 135.000000 degrees.

The area of a regular polygon with 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 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 main in this style of coding */

/**
```

```
36    * Adds two integer variables
37    * @param a the first addend
38    * @param b the second addend
39    * @return the sum a + b
40    */
41   int add(int a, int b)
42   {
43           return a + b;
44   }
45
46   /**
47    * Prints out the integer equation and answer
48    * @param a the first integer input
49    * @param op the operator
50    * @param b the second integer input
51    * @param answer the answer to the equation
52    * @remarks note that this comment is missing the @return
53    * tag as void functions do not return a value. Therefore
54    * the @return tag is unnecessary.
55    */
56   void print_answer(int a, char op, int b, int answer)
57   {
58           printf("The answer to %d %c %d is %d.\n", a, op, b, answer);
59   }
```

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 a two floating point numbers: ");
4   scanf("%lf %lf", &x, &y);
```

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
```

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

List of Files to Submit

Exercises start on page 4.