CpSc 1111 Lab 2
Understanding Compiler Errors & Debugging

Overview

The purpose of this lab is to demonstrate and/or give you practice with:

- debugging and fixing a program that has errors in it while in a lab setting with someone there to help
- understanding compiler errors
- using an editor to modify and add code/comments to a file
- using Unix commands to copy files (`cp`) and rename (`mv`) files
- operator precedence
- integer truncation
- limiting the floating point precision printed to the screen
- using the `const` keyword to declare a constant variable
- using the `math.h` library

Background Information

For this lab, you will download a program that contains errors that you will have to fix. The program contains variables and also the `scanf()` function, which will allow the user to enter values from the keyboard. Background information about these concepts can be found in chapters 1-3 and pages 346 - 358 (printf and scanf) in our course textbook. Also, some explanation is below.

- -

Variables

A variable is a symbolic name for a memory location where values are stored. We, as programmers, can access the memory locations needed in our programs much easier by name rather than by the actual address of the memory location.

In the C language, every variable must be declared. To declare a variable, its ***type*** must be indicated: `int`, `float`, `bool`, `double`, `char`, etc. It is preferable to use meaningful variable names, which improves program readability. (For the `bool` data type, you need to `#include <stdbool.h>`.)

There are rules for forming variable names (page 29 in Chapter 3 of the book). For example, variable names can be comprised of lower and uppercase letters, digits, and also can contain underscores. Variable names must start with a letter or an underscore. Variable names cannot be ***reserved*** words, e.g. `int`, `void`, `while`, `return`, `continue`, etc. (see Table 1.2.2 page 428 in Appendix A). All of the following are valid variable names:

```
value1
_sum
rectangleWidth
user_input
```

- -

Integer Variables

The `int` type is used for integral numbers, or integers – numbers without decimal places. The following are examples of `int` variable declarations:

```
int sideLength;
int area;
```

Variables should usually be declared at the beginning of a function. For example:

```
int main(void) {
      int sideLength;
      int area;

      // rest of program code here

      return 0;
```

```
        }
```

Variables that have not been given a value at any point contain garbage – they will have the value of whatever had been in that memory location the last time it was used. In C, you must never assume that variables have an initial value of 0 upon declaration.

You can ***initialize*** a variable upon declaration (such as `sideLength` below), or at some point later in the program (`area` below).

```
        int main(void) {
              int sideLength = 3;
              int area;

              area = sideLength * sideLength;
              printf("The area of a square with side length of %i ", sideLength);
              printf("is:  %i \n", area);

              return 0;
        }
```

Basic math operations can be performed on integer variables. The following table shows a few examples.

Example	**Description**

`y = 3;`
`x = 5;`
	assigns the value 3 to y and 5 to x

`z = x + y;`	assigns the sum of x and y to z
`z = x + 5;`	assigns the sum of x and 5 to z
`x += 4;`	same as `x = x + 4;` adds 4 to x
`a = x - y;`	assigns the difference of x and y to a
`z -= 1;`	same as `z = z - 1;` subtracts 1 from z
`z *= 2;`	same as `z = z * 2;` multiplies z by 2
`z = x * y;`	assigns the product of x and y to z
`z = x / y;`	assigns the integer dividend of x and y to z *** Be CAREFUL with integer division!**

* Note that with `int` arithmetic, if the result is a decimal value, all decimal portions are lost. For example, `25 / 2` would normally be 12.5, but if you run this in a program using `int`s, it would give you `12`, because it ***truncates*** the value – it drops everything after the decimal point.

- -

`printf()` and `scanf()`

The C run time system automatically opens up 3 files (that are defined in `<stdio.h>`) when a program is run:

1. `stdin` – input from the keyboard
2. `stdout` – output sent to the terminal window
3. `stderr` – most error messages produced by the system are sent to an error file, which is also usually associated with the terminal window

We have already seen in lab and lecture the function that sends output to `stdout`, `printf()` for showing output data to the user.

The `printf()` function prints to standard output (the screen). As we saw in the last lab, `\n` prints a new line. The `\` is a special character. Another special character is `%`. In the following print statement:

```
printf("x is:  %d", x);
```

the `%d` is the ***format string***, indicating the type of the value that will be printed. For integers, `%d` or `%i` can be used as the format string. The entire line in quotes will be printed to the screen, and any format strings will be replaced with the value that comes after the comma. For the following example:

```
int x = 12;
printf("x is:  %d", x);
```

the following will be printed to the screen:

```
x is:  12
```

We can use the `scanf()` function for capturing input from `stdin` (the keyboard) and storing those values into variables. The following example prompts the user to enter an integer, and then uses `scanf` to capture that value and store it into the memory location (variable) that we are calling `userInput`.

```
int main(void) {
      int userInput;
      int sum = 0;

      printf("Enter an integer. ");
      scanf("%i", &userInput);      // userInput gets its initial value here

      // rest of code here

      return 0;
}
```

Notice two things:

1. Format strings are used in `scanf` statements the same way as in `printf` statements (`%d` or `%i` for integers).
2. The `&` is the "***address-of***" operator. In other words, the above `scanf` statement is saying "*scan the integer value entered by the user and store it into the memory location of (or address of) the variable called* `userInput`".

Lab Assignment

For this lab, you will copy a file from some other location to your account. Log in and navigate to a directory where you will be working on this lab, perhaps create a lab2 directory for this week's lab.

Get the following file (e.g., from Canvas):

```
errors.c
```

When you open the file with your editor of choice, you will see some instructions at the top of the file in a header comment. ***Make sure you follow all those instructions given at the top of the file.*** Before you submit your lab work, don't forget to add the correct header comment information at the top. It would be fine (a good idea to clean up the comments before submitting any work) to delete the instructions at the top (after you are done with them) before submitting your work. Recompile one last time before submitting to be sure you didn't unintentionally delete anything important when you cleaned up the comments/code.

Reminder About Style, Formatting, and Commenting Requirements

- The top of your file should have a header comment, which should contain:
    - Your name
    - Date
    - Lab section
    - Lab number
    - Brief description about what the program does
    - Any other helpful information that you think would be good to have.
- Variables should be declared at the top of the main function, and should have meaningful names.
- Always indent your code in a readable way. Some formatting examples may be found here: https://people.cs.clemson.edu/~chochri/Assignments/Formatting_Examples.pdf

Turn In Work

1. Before turning in your program, make sure you have followed **all** of the instructions that were at the top of the file. Always test, test, and retest that your program compiles and runs successfully on our Unix machines before submitting it.
2. Show your ta that you completed the assignment. Then submit your `lab2.c` program using the handin page: http://handin.cs.clemson.edu . ***Don't forget to always check on the handin page that your submission worked. You can go to your bucket to see what is there.***

Grading Rubric

For this lab, points will be based on the following:

Functionality	85 (all errors are fixed and program works, -5 if warnings when compile)
Formatting & comments	15 (comments for fixed errors, cleaned up comments at top, other formatting)