Goals
Understand how to execute a basic block iteratively using loops.

Review
• Reading integer values from the keyboard:
  int x; // receives input value
  int howmany; // how many values were read successfully
  howmany = fscanf(stdin "%d", &x);
  x = x + 5;

• Printing integer values to the screen:
  fprintf(stdout, "The value of x is: %d\n", x);

• Printing x first in decimal and then in hexadecimal on the same line with specified field widths:
  (To have the output of multiple fprintfs() on the same line, omit the 'n character from all but the final fprintf().)
  fprintf(stdout, "X in decimal: %8d", x);
  fprintf(stdout, "X in hexadecimal: %6X\n", x);

• What is the purpose of the 8 and the 6 in the above two print statements?

• What is the difference in output between the following two lines?
  fprintf(stdout, "X in hexadecimal: %6X\n", x);
  fprintf(stdout, "X in hexadecimal: %6x\n", x);

Iterative execution (loops)
It is often necessary to repetitively execute a statement, or block of statements, while some condition
remains true, or until some condition becomes false.

Because of all of the possible formulations of conditions, it is the case that there are multiple correct
ways to construct such iterations. And, alas, there are even more incorrect ways to construct them.

There are three different iteration constructs in C:
1. while loop
2. do-while loop
3. for loop
1. First, we will start with **while** some condition **remains true**. This structure is commonly called the "**while loop**".
   - Structure:
     ```c
     while (condition)
     {
       one-or-more statements;
       loop_expression; // expression that changes the condition
     }
     ```
   - Important notes:
     - The statement or statements controlled by the **while()** are called the **body of the loop** and are enclosed by curly braces.
     - It is necessary that the **body of the loop modify the value of the condition**. Why?
     - Proper use of indentation is **critical** for human readability.
     - But, indentation is **completely irrelevant** to the C compiler.
     - Example: Print the integer values between 10 and 1 in decreasing order. (Note: This is **not** a complete C program; it needs to include stdio.h and it needs a proper main() function.)

     ```c
     int value;
     value = 10;
     fprintf(stdout, " Table of values \n");
     while (value > 0)
     {
       fprintf (stdout, "%3d\n", value);
       value = value - 1;
     }
     ```
   - In the above code, we only want the heading to be printed **one time**, so the **fprintf()** is placed **outside the loop**.

   - Thought experiment: What does the following code segment do?
     a. Refuse to compile because of missing {} ?
     b. Compile fine and work fine?
     c. Compile fine but work not so fine?; if so how would it work not so fine??
   - Do you see how, if proper indentation is used, it would be so much easier to answer these questions?

     ```c
     int value;
     value = 10;
     fprintf(stdout, " Table of values \n");
     while (value > 0)
     {
       fprintf (stdout, "%3d\n", value);
       value = value - 1;
     }
     ```
Another type of loop is the “**do-while**” loop, which looks like the following:

```java
do
{
    one-or-more statements;
    loop_expression;
} while (condition);
```

**Important notes:**
- The above loop is guaranteed to execute the statement(s) at least once. Do you see why?
- As with the `while()` loop, it is necessary that the body of the loop modify the value of the condition so that it is not an infinite loop.

The third loop structure is commonly called the "**for loop**".

**Structure:**

```java
for (init_expression; loop_condition; loop_expression)
{
    one-or-more statements;
}
```

**Important notes:**
- The statement or statements controlled by the `for()` are called the body of the loop and are enclosed in curly braces.
- The `init_expression` gives the starting value for the control variable being used by the loop. The control variable needs to be declared somewhere above the `for` loop, probably at the top of the program where the other variables are declared.
- Example:
  ```java
  int i;
  for (i = 0; loop_condition; loop_expression)
  ```
- The `loop_condition` will be evaluated before each iteration of the loop; if it evaluates to true, an iteration of the loop body will occur; if it evaluates to false, no code inside the body of the loop will execute.
- Example:
  ```java
  int i;
  for (i = 0; i <= 3; loop_expression)
  ```
- In this code above, as long as the value of `i` is less than or equal to 3, the statements inside the body of the loop will be executed.
- The `loop_expression` will change the value of the loop control variable so that a stopping point may be reached in the `loop_condition`.
- Example:
  ```java
  int i;
  for (i = 0; i <= 3; i++)
  ```
- In the code above, `i` will increment by 1 each time immediately after an iteration of the loop, allowing for the `loop_condition` to eventually evaluate to false (assuming a proper condition exists), thereby ending the loop. (`i++` is the same as `i = i + 1`)
• Example: The earlier while() loop is equivalent to:
  
  for (value = 10 ; value > 0; value--)
  {
    fprintf(stdout, "%3d\n", value);
  }

• Note: value-- is the same as: value = value - 1

• The starting value for the control variable called value in this case is 10, and it decrements by 1 after each iteration of the loop. The loop will continue to iterate as long as the value of the control variable is greater than 0. So, how many iterations will occur?

Assignment

Write a program called lab3.c that does the following:
• Reads two integer values from the standard input, one for length and one for width.
• Print to the user what the area of the rectangle with that given length and width showing the decimal value, hexadecimal value, and octal value as formatted below (the right-most digit of each value is 5 spaces in from the left margin, resulting in a tabular looking format).
• Then, using for loops, print out a hollow rectangle shape with asterisks ‘*’ using the values entered for length and width. For example, a sample run of your program should look similar to this:

Enter an integer for length: 9

Enter an integer for width: 12

The area of your rectangle is:
  108 in decimal
  6C in hexadecimal
  154 in octal

**********
*   *
*   *
*   *
*   *
*   *
*   *
*   *
*   *
*   *
*   *
*   *
**********

Turn In Work

Show your ta that you completed the assignment, then turn in your lab3.c program using the handin page:  http://handin.cs.clemson.edu