Lab 3

Introduction to C

CSCI 112, Spring 2015

Objectives

- Practice using selection (`if ... else if ... else`) statements in C
- Practice using `while` and/or `do...while` loops in C
- Practice using `for` loops in C
- Practice nesting a loop within another loop
- Learn about additional formatting codes for `printf`
- Design a robust text user interface that checks input validity

Overview

In this lab, you will write a program that prints out a pattern based on user given input. You will restrict the user input to any odd numbered integer greater than or equal to 1 and less than or equal to 9. For instance, when the user enters a 9, your program will output:

```
        1
      1 2 3
    1 2 3 4 5
  1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
  1 2 3 4 5 6 7
    1 2 3 4 5
      1 2 3
        1
```

This program requires you design and implement three functions: `get_input`, `is_valid`, and `print_pattern`, in addition to your `main` function. Your main function will call `get_input` and `print_pattern`. Your function `get_input` will repeatedly call `is_valid` until the user enters a proper number that passes the validity check described below.

Input Function

Begin by writing a function named `get_input` to retrieve integer input from the user.

1. Your function should prompt the user for an odd number between 1 and 9.
2. Recall that `"%d"` is the code for a decimal integer.
3. You should then call `is_valid` and pass it the user-entered value in as an argument.
4. If the value is invalid, force the user to try again until the number is valid.
5. Your function will return a valid integer: an odd digit, between 1 and 9 inclusive.

PROTIP: It is common to implement input request with a user using a `do...while` loop. This allows you to ask at least once and repeat until the user enters a valid input.

Validation Function

Next, write your function named `is_valid`. This function will be called by your `get_input` function. You will not use a loop in this function.

1. This function should take an integer as input parameter.
2. Your function will determine if the parameter is an odd number between 1 and 9.
3. You should display a clear message whenever the user enters an invalid number:
   - You should tell the user when the number was even (not odd):
     `You have entered an even number.  Please try again.`
   - You should tell them when the number is too large (> 9):
     `You have entered a number greater than 9.  Please try again.`
   - You should tell them when the number is too small (< 1):
     `You have entered a number less than 1.  Please try again.`
4. If the input is an odd number between 1 and 9, you should return 1 (true).
5. If the input is not an odd number between 1 and 9, you should return 0 (false).

PROTIP: To achieve the proper spacing in your `print_pattern` function, you need to explore the `"%*s"` format code for `printf`, which gives variable length padding.

Example: `printf ("%*s", x, "");  // Prints x number of blank spaces`

Print Function

Next, complete your function named `print_pattern`.

1. This function will take in an integer as an input parameter and print the correct pattern based on this number.
2. You can assume the number is an odd digit, between 1 and 9, since you verified its validity by using the function `is_valid` from within your `get_input` function.

3. The function needs to display your rows of numbers such that they form a diamond as shown in the two `Example Executions`.

4. You will accomplish this using `while` loop nested inside a `for` loop. Try to figure this out on your own before reading the substantial HINT below.

PROTIP: Do not use a long series of `if` statements and pre-formatted strings to achieve the variable white space printing. This will be verbose, difficult to follow, inefficient, and not representative of good computer programming. Instead, use the dynamic calculation approach described in the HINT below.

SPOILER ALERT

HINT: You will use two nested loops to accomplish your formatting of the diamond.

The outer `for` loop will be responsible for the number of rows (lines) necessary and will increment the counter by two each time.

For each row, first print out the correct number of whitespaces. To achieve the variable number of white spaces on the left, use the `"%*s"` format code. The number of white spaces will be the difference between your user input number and the outer variable counter. This allows you to determine the number of white spaces being printed as a calculation based on the user number.

For example, consider a user input value of `n=7` and a row counter `i` starting at `i=1`

```
______1           // i=1, 6 blank characters = n - i
____1 2 3         // i=3, 4 blank characters = n - i
__1 2 3 4 5       // i=5, 2 blank characters = n - i
1 2 3 4 5 6 7     // i=7, 0 blank characters = n - i
```

Next, your inner `while` loop will print out the individual numbers on each row. Do not forget to print extra space after each digit.

Do not forgot to print a newline character `\n` at exactly the appropriate place.

You will need a pair of nested loops to print the top half of the diamond, and a second set of nested loops to print to lower half of the diamond. Take care not to reprint the middle line.

Main Function

1. Your main function will first call `get_input` and save the number returned.
2. Next, call `print_pattern` to display your number pyramid.

PROTIP: Never use a `float` or a `double` as a loop counter variable as you may get an infinite loop. Instead use an `int` for your loop counter variables.

Test

Test your program! There are a number of choices you may make during your design and implementation. However, you will lose significant points if you do not properly format your diamond as shown.

PROTIP: Make sure you output what is expected, nothing more and nothing less. Otherwise the automated grading script will subtract points!

Extra Steps

The following are optional enrichment exercises that you may chose to implement in order to practice additional skills. You may chose to implement some, all, or none of them.

1. Restructure your code in such way that you do not use neither <= nor >= relational operators. You will use < or > and/or other operators.
2. Generate the same output, but use only decremented counters (e.g. `i = i - 1`).
3. Modify your code to use only prefix decrements (e.g., `--i`).
4. Change your `is_valid` function to use a `switch` construct instead of `if` statements.
5. Rewrite your `print_pattern` function to use a `for` loop nested within a `for` loop.

PROTIP: If you complete any of the `Extra Steps` above, make sure your output still matches the expected output given in the `Example Executions`.

Example Execution I

```
$ ./program3

Enter an odd number less than or equal to 9 and greater than 0 > 6
You have entered an even number. Please try again.

Enter an odd number less than or equal to 9 and greater than 0 > 11
You have entered a number greater than 9.  Please try again.

Enter an odd number less than or equal to 9 and greater than 0 > 5

    1
  1 2 3
1 2 3 4 5
  1 2 3
    1
```

Example Execution II

```
$ ./program3

Enter an odd number less than or equal to 9 and greater than 0 > 0
You have entered a number less than 1.  Please try again.

Enter an odd number less than or equal to 9 and greater than 0 > 4
You have entered an even number. Please try again.

Enter an odd number less than or equal to 9 and greater than 0 > 7

      1
    1 2 3
  1 2 3 4 5
1 2 3 4 5 6 7
  1 2 3 4 5
    1 2 3
      1
```

Resources

- You should attend the Friday Lab session to seek assistance from the TAs and CAs.
- For general questions, check the D2L `Lab 3 Help Forum` to see if another student has already asked your question. Otherwise, post your question on the D2L forum. This forum is intended for general questions that will benefit all students. Do not paste your code nor give away any specific answers to the lab on this forum.
- For specific questions, attend the weekly lab session or attend the TA's office hours.
- You are encouraged to use resources or tutorials on the internet to learn unix or C. Check the class resource list on D2L for some links to useful resources.

PROTIP: The logic in `print_pattern` function can be a bit tricky to work out. You should come to your lab section this week to receive help and hints from the TAs.

Submission

- ASSIGNED: February 17th
- LAB DAY: February 20th
- DUE DATE: February 27th, 8:00am

1. Make sure you included ample and informative comments – it is 20% of your grade!
2. Rename your C file to `last_first_lab3.c` and substitute your last and first name.

   PROTIP: If you fail to follow the above file naming conventions, your program may not be graded automatically and you will lose points.

3. Submit your `.c` file to the D2L dropbox Lab3.

   PROTIP: Submit only your C file. Do not submit your object file or your executable program. Do not archive (e.g., `zip`) your file.

Each student will complete and submit this assignment individually. Submit your work before the deadline as late work will receive a 50% penalty. Labs submitted more than 24 hours late will not be accepted. The late deadline is Saturday, February 28th, 8:00am.