CPE 101
Spring 2012
Lab 4 (Loops)

Updates and Changes

- None yet

Objectives

- To learn about repetition structures in C programs, such as sentinel-controlled loops, counting loops, and end-file loops.
- To write complete C programs from a given specification.

Overview

In this lab, you will be introduced to loops in C. You will write or edit three small programs that use loops to accomplish different tasks.

Rules

- You may choose to work with one lab partner, or alone.
- You must hand in your own copy of the work (even if working in teams).
- You may ask the professor, TA, or classmates for help, but they shouldn't do your work for you.

Part 1: Basic C Program Using Loops

1. Download the contents of the file rainfall.c from:

   ```
   cp ~jworkman/www/101/Labs/Lab04/rainfall.c .
   ```

2. Study the source code to determine the program's purpose. Compile and execute the program with the following data: `55 33 77 -99`. Does it produce the results you expected?

3. "Comment out" the `printf` statement on line 18. Execute the program and enter all the input data on a single line. Does it produce the correct results? Uncomment this line.

   - *"Commenting out" a programming statement means enclosing it in the comment tags `/*` and `*/`. The compiler will ignore all comments when compiling the program, meaning this line will not be executed when the program is run.*

4. Change the sentinel constant to negative one. Execute it with the following data: `-5 -10 -15 -99 -1`. Does it produce the results you expected? Do the results make sense?

5. Change the `while` loop condition (and sentinel if necessary) so that *any* negative number will terminate the loop.

6. Explain the effect of moving line 17 after 19. What will be the output for the following data: `10 15 5 3 -1`? Execute the program with this change and verify your prediction. Return the statement to its original position.

7. Enhance the loop so that it counts the number of rainfall measurements that are entered. Display and label this count after the sum. Execute the program and be sure it works correctly for datasets of size 0, 1, 2, and 3 (among others).

8. Enhance the loop again so that it also prints the average rainfall. This should be displayed to two decimal places. You may edit variables or data types that already exist, if necessary. Make sure this works on data sets of size 0, 1, and higher. This means that your result should always print a number for the average rainfall, never '`nan`'.

9. **Demo your solution to the instructor.**

10. Hand in your program from unix1 using the following command:

    ```
    handin graderjw Lab04-XX rainfall.c
    ```

Note: To receive credit for Part 1 of this lab, you must demo ***and*** *hand in your program.*

Part 2: More Loops

1. Download (or copy/paste) the contents of the file cubesTable.c from:

    ```
    cp ~jworkman/www/101/Labs/Lab04/cubesTable.c .
    ```

2. Study the source to comprehend how the program works.

3. Predict the output for the values `9 3`

4. Compile the program; it should compile with no errors. Execute the program and see if your prediction is correct. Then execute the program and enter values: `-5 5 3`. Make sure you understand the output.

5. Predict how the output will appear if the user enters: `-5 5 -9 3`. Execute the program and see if your prediction is correct.

6. Enhance the `get_first()` function by adding a input validation loop similar to the one in the `get_table_size()` function. The loop should reject negative user inputs.

7. Enhance the `show_table()` function by implementing the code for a `for` loop that will display the desired table of cubes. For example, if the user enters `4 5` as the inputs, the table will look like this:
```
Number  Cube
5       125
6       216
7       343
8       512
```

8. Add a new function `get_increment()` that is similar to `get_first()`. It should display the prompt "`Enter the increment between rows: `". The function should use an input validation loop to reject negative user inputs.

9. Enhance the `show_table()` function so that it takes a third parameter that is the row increment in the table. Modify the counting loop so that it uses the row increment when displaying the table.

10. Enhance the main function so that it calls `get_increment()` and passes the result to `show_table()`. For inputs of `4 5 3` the table should now look like this:

```
Number  Cube
5       125
8       512
11      1331
14      2744
```

11. Enhance the `show_table()` function so that it displays the sum of all the cubes in the table on a separate line below the table, in the format "`The sum of cubes is:` *x*" (where *x* is the sum of cubes).

12. Test your program thoroughly against the instructor solution version:

```
cp ~jworkman/www/101/Labs/Lab04/cubesTableInst .
```

13. Hand in your program from unix1 using the following command:

```
handin graderjw Lab04-XX cubesTable.c
```

Part 3: Even More Loops

Section 1

- Create a new C file named `findSevens.c`

- Write a program to find the position of the first and last occurrences of the number 7 in a list of positive integers.

- Prompt the user once for a list of integers greater than zero. Keep reading and processing numbers until the user enters something negative.

- The number of positive integers in the input is not known in advance; the list may contain 0 or more positive integers.

- You must output the number of non-negative integers in the list.

- You must output the location of the first and last seven in the list.

- For example, if the list contains the numbers `1 8 7 3 4 7 9 2 4`, the output would be:

  ```
  This list contains 9 numbers.
  First occurrence of '7': 3
  Last occurrence of '7': 6
  ```

  If the list contains no sevens, rather than printing the first and last seven, you should display the message "`This list contains no sevens.`"

- You may not use any magic numbers in your code.

Section 2

- Edit your program to include the following functionality:

- Add a new function with the prototype: `int getNumber();`

  - This function should display the prompt "`Enter a number to find: `" and scan in an integer.
  - This function should use an input validation loop that redisplays the prompt to reject negative user inputs.
  - Your main program should look for this number rather than the number 7.

- Change your program output to display the proper number in single quotes. E.g. with input 4 and the same list of numbers as in Section 1, your output should look like this:

  ```
  This list contains 9 numbers.
  First occurrence of '4': 5
  Last occurrence of '4': 9
  ```

- Rather than printing the message "`This list contains no sevens.`", create a new function with the prototype:

```
void displayMessage(int occurrences, int num);
```

- This function should display a specific message based on the number of occurrences in the given list.
- If the number of occurrences is 0, the message should read "`This list contains no occurrences of <num>.`" Replace <num> with the actual number.
- If the number of occurrences is greater than 10, the message "`Wow, that's a lot!`" should be displayed.
- If the number of occurrences is between 1 and 5 (inclusive), the message "`Not too many in this list.`" should be displayed.
- If the number of occurrences is between 6 and 10 (inclusive), the message "`Quite a few were found!`" should be displayed.
- For example, if the number the user enters is '4' and the list contains the numbers: `1 8 4 9 3 4 4 8 2 4 10 3 4 2 4`, the displayed message would be "`Quite a few were found!`"

Your findSevens program should be tested with various input data, and must match the solution exactly. To diff your results, you may copy the solution executable at the following location to your directory, and compare your output:

```
cp ~jworkman/www/101/Labs/Lab04/findSevensInst .
```

Be sure to test your solution thoroughly!

Here is a sample run of my solution version:

```
Enter a number to find: 4
Enter a list of numbers (negative to quit): 1 8 4 9 3 4 4 8 2 4
10 3 4 2 4 -1

This list contains 15 numbers.
First occurrence of '4': 3
Last occurrence of '4': 15
Quite a few were found!
```

Deliverables (Handin)

- You should have already demoed and submitted `rainfall.c`. If not, demo to your instructor, then hand in using the command in Part 1.

- From unix1, also hand in `cubesTable.c` and `findSevens.c` using the command:
  `handin graderjw Lab04-XX cubesTable.c findSevens.c`

Grading

This lab is worth twenty points, and will be broken up as follows:

- Part 1 (rainfall.c): 4 points
- Part 2 (cubesTable.c): 8 points
- Part 3 (findSevens.c): 8 points

Your output will be graded on accuracy, so be sure to test each of these programs with various data, and compare this data with known good output where appropriate (using diff).

Updated Spring '12 by Julie Workman. Adapted from a previous Lab 4.