Lab 6 - Input Validation

Due: end of class

OBJECTIVE:

Secure programming requires validation of inputs, both for types and values. For example, when inputting a variable that should be an integer between 0 and 23, the program needs to validate that the input is indeed an integer (e.g., not a string with non-numeric characters) and that it is in [0,23]. We have done only the latter in the past, and this lab will concentrate on the former.

To do this, we will augment the shapes lab with input validation, and add features along the way.

TASK 0

In the shapes lab, you drew a number of shapes as 5 separate programs (since we didn't have functions at that time). Rewrite the code into 5 separate functions that take appropriate arguments. These functions will only have side effects, returning void.

Then, write a function that does the following:

1. Present a menu: "Enter (R)ectangle, (L)owerTriangle, (T)rapezoid, C(ircle), or (E)nd"
2. Input a character corresponding to the menu choice
3. Depending on whether the character is R, L, T, or C, input appropriate parameters
4. Call the appropriate drawing function

The function will do these steps repetitively until "E" is input.

For this task you only need to validate values (not types)

TASK 1

Run your above program, typing in a non-numeric value at a parameter prompt. State what happens in comments. Also state what happens if you enter a number causing overflow. You may need to terminate the program. In Linux this is done by typing ^C. Alternatively, you can open another window, list all processes by typing "ps -e" (grepping the output), and killing the appropriate process by typing "kill -9 <pid>".

Now we wish to validate the input for types as well as values to avoid this problem. There are 3 major concepts to be aware of:

1. The >> function returns false when an input failure occurs. One possible cause of input failure is a type error.
2. Since the input stream library maintains an internal state, it needs to be reset after an error. This is done by calling cin.clear().
3. Since there may have been additional data typed on the error line (e.g., "foo 17" on a line expecting 2 integers), the rest of the line needs to be cleared before retrying. The cin.ignore() function indicates that the rest of the input line already read should be ignored.

Although we won't use it in this program, recall that the cin.fail() and cin.eof() functions can also be a major component of input validation.

Identify places in your Task 0 code that need additional input validation, and rewrite the code to handle this. For example, you may write something like the following pseudocode:

```
output prompt
while input n is false
  reset/clear system
  output an error message
  output prompt
// assert n contains valid input
```

HAND IN

Your 136 instructor will tell you what to hand in and how.

GENERAL COMMENTS FOR ALL PROGRAMS THIS SEMESTER

You should have the following header on all programs:

```
/*
  Author: <name>
  Course: {135,136}
  Instructor: <name>
  Assignment: <title, e.g., "Lab 1">

  This program does ...
*/
```

GRADING

All 135 and 136 programs this semester will be graded on:

- Correctness: Does your program work?
- Testing: Have you generated sufficient and good test data to give reasonable confidence that your program works?
- Structure: Have you structured your code to follow proper software engineering guidelines? This includes readability and maintainability.
- Documentation: How well documented is your code? Good documentation does not repeat the code in English, but explains the point of each code block, highlighting any design decisions and/or tricky implementation details.