SIT102 Introduction to Programming

Credit Task 3.3: Validating User Input

Overview

Functions are one of the main tools for managing complexity in procedural programs. In this task you will build your own set of functions to handle user input. You will see how building more complex functions on top of existing functions can greatly help reduce the complexity of your program's code.

Submission Details

Use the instructions on the following pages to create your own user input functions.

Submit the following files to OnTrack.

- Your program code
- Reflections on the use of functions to manage complexity

The focus of this task is on both the use of control flow as well as the idea of functional decomposition to help manage complexity. See how you can use loops and branches to validate user input, and how you can combine together multiple functions to simplify the coding of the task.

Instructions

This task builds upon the user input functions within our own custom terminal user input file and header.

Lets get started.

1. Watch the Validating User Input video, and make the changes shown to your own **Hello User** project.

   Skip to the **Your Task** section once you have followed all of the steps in the video.

   Here are some notes related to these steps:

   - Make sure to copy your terminal user input cpp and header file from the previous task.
   - SplashKit includes a number of functions that can help:

Function	Task
`bool is_integer(string text);`	Checks if text only contains an integer (so will convert without error).
`bool is_number(string text);`	Checks if text only contains a number (so will convert without error).
`string trim(string text);`	Removes spaces from the start and end of a string, returning the result as a new string.
`string to_lowercase(string text);`	Converts the text to lower case, and returns the result as a new string.
`string to_uppercase(string text);`	Converts the text to UPPER CASE, and returns the result as a new string.

2. Make sure you have the code from the video working before you continue.

   Remember you can use the Programming Hub and discussion board to get help if you get stuck. Try to work it out yourself, but ask for help before it starts to take too long.

Your Task

1. Extend the terminal user input code to add in some additional validations.

    - Update your `read_double` to make sure it checks if the user has entered a valid number before doing the conversion. It should not crash when an invalid value is entered.
    - Create a new `read_double_range` to read a number between minimum and maximum (inclusive) values. For example, read a double between 0.0 and 1.0.
    - Create a `read_boolean` function to ask "yes/no" type questions. User should be able to type "yes" or "y" (for true) and "no" or "n" for false. (This should not be case sensitive, and should work if there are leading or trailing spaces).

      There are some hints after the instructions.

2. Try the sample program on the following page.

3. Compile and run the program and check that it works correctly... then grab a screenshot to upload to OnTrack.

    Show that you get an error message when the user enters an invalid number, or when they enter a value outside of the range (for `read_double_range`). Also check you can use any case for the `read_boolean`, so strange things like "yEs " and " nO" should still work.

Remember to backup your work when you finish.

Task Discussion

Discuss the following with your tutor to demonstrate your understanding of the concepts covered.

- Explain how control flow has made these functions more useful. How have their responsibilities been expanded?
- How do these different functions work together? Explain how the responsibilities of the different functions build upon earlier functions - making it easier to create more complex validations.

```
#include "terminal_user_input.h"

int main()
{
    string name;
    int age;
    double height;
    bool test;

    bool again;

    do
    {
        name = read_string("Enter your name: ");
        write_line("For read integer test errors for real numbers and text");
        write_line(" - eg '3.1415' and 'fred'");
        age = read_integer("Enter your age: ");
        write_line("For read double test errors for text - eg 'fred'");
        height = read_double("Enter your height: ");
        write_line("read boolean test errors with text other then yes and no");
        test = read_boolean("Testing read boolean: ");

        write_line("Got values: ");
        write("String: ");
        write_line(name);
        write("Integer: ");
        write_line(age);
        write("Double: ");
        write_line(height);
        write("Boolean: ");
        if ( test )
            write_line("yes");
        else
            write_line("no");

        again = read_boolean("Try this again: ");
    } while ( again );


    return 0;
}
```

Hints for Read Boolean

Firstly, understand what you are trying to create. This is a **general** function to read a boolean - so it could be asked any kind of yes no question the caller wants to know about. Your job is to create steps for the computer to follow for it to ask the question the user wants, and get the user to respond with either a yes or no. The last thing the function will do is return a boolean result (a true or false) to indicate what the user entered.

Basic logic

The basic logic for this will be the same as with the other terminal user input functions.

1. Read in a string with the line entered in response to the question.
2. While it is not a valid value.
    1. Show an error
    2. Read in a string with the line entered in response to the question.
3. Return true if they entered "yes" or they entered "y".

Making it case in-sensitive

Once you read the string, convert it to all lowercase and remove any spaces, and save the result. This will mean you can just check lowercase versions of the data you are after. For example **line = trim(to_lowercase(line));** - make sure you do this any time line is updated.

Remember with the checks you will need to check each part individually. For example, you can check if **line** is "yes" or "y" using **line == "yes" or line == "y"**.