Mini Project on Input Validation

In practice, program users don't always follow instructions, and you can get a mismatch between what a program expects as input and what it actually gets. Such conditions can cause a program to fail. However, often you can anticipate likely input errors, and, with some extra programming effort, have a program detect and work around them.

Your Task: Develop a library of functions that can be used for various kinds of input validation in different programs that require a particular kind of input. Write a menu driven program with different cases to demonstrate the use of your library.

A robust program should respond to invalid input in a manner that is appropriate, correct, and secure. When your program runs across invalid input, it should recover as much as possible, and then repeat the request, or otherwise continue on. Arbitrary decisions such as truncating or otherwise reformatting data to "make it fit" should be avoided.

Below is a partial list of some checks that you might want to include (Earn bonus marks for additional types of input validation):

- **Correct type of Data:** it should check that input is not an unreasonable type (integer, decimal, string etc.). For example input like 123wg5 for an integer input.
- **Range check** - numbers checked to ensure they are within a range of possible values, e.g., the value for month should lie between 1 and 12.
- **Length check:** variables are checked to ensure they are the appropriate length, for example, a US telephone number has 10 digits and input may include other characters like '-' which should not be counted in the length.
- **Format check** – Checks that the data is in a specified format (template), e.g., dates have to be in the format DD/MM/YYYY.

More critical implication of avoiding input validation:
Any program input – such as a user typing at a keyboard – can potentially be the source of security vulnerabilities and disastrous bugs. If external data is not checked to verify that it has the right type of information, the right amount of information, and the right structure of information, it can cause problems. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Examples of Occurrence:

1. A Norwegian woman mistyped her account number on an internet banking system. Instead of typing her 11-digit account number, she accidentally typed an extra digit, for a total of 12 numbers. The system discarded the extra digit, and transferred $100,000 to the (incorrect)

account given by the 11 remaining numbers. A simple dialog box informing her that she had typed too many digits may have avoided this expensive error. Olsen, Kai. "The $100,000 Keying error" IEEE Computer, August 2008.

2. Web applications are highly vulnerable to input validation errors. Inputting the invalid entry "!@#$%^&*()" on a vulnerable e-commerce site may cause performance issues, or "denial of service", on a vulnerable system, or invalid passwords such as "pwd'" or "1=1— " may result in unauthorized access. *The site xssed.com lists nearly 13,000 vulnerable Web pages, including sites such as yahoo.com, google.com, msn.com, facebook.com, craigslist.com and cnn.com*