CS 247 Spring 2014
Assignment 2 Question 2 --- Specifications

Q2 [80 marks] Exceptions, Smart Pointers, Modules

You are to implement a UserAccount ADT that maintains pairs of userids and passwords.

- Userid is a separate ADT that ensures that userids are unique.
- Password is a separate ADT that ensures that passwords meet some minimum strength requirement (to protect against cracking).
- A UserAccount is a Userid and corresponding Password, implemented as smart pointers to those objects.

Objective

The objective is to practice throwing and catching and recovering from exceptions, using smart pointers, and working with modules and makefiles. Some of the design decisions in the ADTs are less than ideal (e.g., having ADT methods that interact with the user); they are designed so that you write the code that catches and recovers from thrown exceptions. You are to implement the UserAccount ADT, as well as design and implement the Userid and Password ADTs, and provide a makefile that automates incremental compilation (i.e., compiles only the modules that have changed since the last compilation of the program). Each of the ADT's public interfaces should be suitable with respect to whether the ADT is an entity or value-based ADT, and whether the ADT is mutable or immutable. ***(The design decisions you make with respect to the ADTs' public interfaces affect your answers to question Q3 of the assignment. I recommend looking at that question after designing your ADTs but before implementing them.)***

Provided Code

We provide the header file (`UserAccount.h`) for the UserAccount ADT. In addition to implementing the UserAccount ADT, you will also need to provide header and implementation files for the Userid and Password ADTs. ***You cannot change the public interface for `UserAccount.h`, cannot add any data members to `UserAccount.h`, and cannot add any friend declarations.***

We also provide a main program for your solution The main program is a **test harness** that you can use to test your ADT implementations by creating various user accounts and performing operations on those accounts. The test harness is not robust. (It is throwaway code.) If you enter invalid commands, it might cause the program to terminate. It should go without saying (but I'll say it anyways) that your program will be tested only on valid input commands.

Initialization

When the program starts executing, it prints a welcome message and asks for the first command from the user.

Commands

There are 7 valid commands that the test harness will recognize:

n <#>
D <#>
d <#>
c <d#>
a <#>
r <#>
CTL-D

a) n

The test harness creates a new UserAccount object and assigns it to a variable that a referenced by # (= 1 or 2) . The user is prompted for a suitable userid:

```
Enter preferred userid:<sp>
```

The user's response is read up to the first whitespace character and that word is treated as the preferred userid. The Userid constructor throws an exception object if it is asked to construct a new Userid object whose userid is currently in use. An in-use userid becomes available again when its Userid object is destroyed. Other than uniqueness, the constraints on valid userids are pretty lax: a userid can be any nonempty string that includes no whitespace character. Exceptions thrown by the Userid constructor are handled by telling the user that their preferred userid is in use and asking for another userid. This dialogue continues until the user provides a userid that is not currently in use.

```
Userid "<userid>" already exists.  Please try again.
Enter preferred userid:<sp>
```

The user is prompted for a suitable password:

```
Enter preferred password:<sp>
```

The user's response is read up to the first whitespace character and that word is treated as the preferred password. The Password constructor throws an exception object if it is asked to construct a Password object that violates any of the following requirements.

- consists of at least 8 characters
- contains at least two upper-case letters
- contains at least two digits
- contains at least one of the following symbol characters: ~!@#$%^&*()_-+={}[]:;<,>.?/

Exceptions thrown by the Password constructor are handled by telling the user which of the strength requirements their preferred password violates and prompting the user for another password. A password that violates all four strength requirements would result in the following output:

```
Password :
<sp><sp><sp><sp>- must be at least 8 characters
<sp><sp><sp><sp>- must include at least 2 capital letters
<sp><sp><sp><sp>- must include at least 2 digits
<sp><sp><sp><sp>- must include at least 1 symbol
Enter preferred password:<sp>
```

This dialogue continues until the user provides a password that satisfies all of the strength requirements.

b) D

The test harness deletes the object associated with variable number #. This variable number is subsequently not associated with any object. *You do not need to do anything.*

c) d

This command deactivates the UserAccount associated with variable number #, using the UserAccount `deactivate()` method.

d) c

This command checks whether the UserAccount associated with variable number # is deactivated, using the UserAccount `check_deactivated()` method. The test harness prints a message indicating the result of the check.

e) a #

The UserAccount object associated with variable number # authenticates the user by asking for a password and comparing the user-provided password against the object's stored Password. If the passwords do not match, the user is notified and told how many more attempts he has to provide the correct password:

```
Invalid password. You have 1 tries to get it right.
Enter password:<sp>
```

If the user fails to enter the correct password after 3 attempts, the account is deactivated:

```
Imposter!! Account is being deactivated!!
```

An attempt to authenticate a deactivated account (for example, by using the test harness command **a #**) will fail:

```
Account has been deactivated.
```

f) r #

This command reactivates a deactivated account associated with variable number #. Reactivating an account involves setting a new password. As when constructing a new UserAccount,

- the user is prompted for a new preferred password
- the user's response is read up to the first whitespace character and the word is treated as the preferred password
- the Password constructor throws an exception object if the preferred password violates any of the password requirements
- the user is informed of which password requirements their preferred password violates, and is asked to provide an alternative password
- the dialogue repeats until the user provides a password value that passes the strength requirements

g) CTL-D

If you hold down the control key on your keyboard while pressing the D key, you will generate an EOF (end of file) character. When the test harness's input stream sees this, it terminates the program.

Sample Execution

Below is an example partial execution. User input is shown in **bold** font.

```
Test harness for UserAccount ADT:

Command: n 1
Enter preferred userid: jmatlee
Enter preferred password: funny
Password :
    - must be at least 8 characters
    - must include at least 2 capital letters
    - must include at least 2 digits
    - must include at least 1 symbol
Enter preferred password: RR&77ii
Password :
    - must be at least 8 characters
Enter preferred password: RR77&iii

Command: a 1
Enter password: funny
Invalid password. You have 2 tries to get it right.
Enter password: RR&&7ii
Invalid password. You have 1 tries to get it right.
Enter password: RR&&77ii
Imposter!! Account is being deactivated!!

Command: a 1
Account has been deactivated.

Command: r 1
Enter preferred password: RR&&77ii

Command: ^D
```