CSC 112 - Functions and Multiple File Coding

Reading and Background

zyBook and lectures as appropriate.

Other reference as you deem appropriate to develop the system.

Part 1: Getting started and calling a function from a function.

In this part, you will work with functions and functions that call functions. All code will be in `main.cpp`. Testing and grading will be via zyLab. Start with the template in zyBook. Write the following function:

`FindMax4(double num1, double num1, double num3, double num4)`

that returns the max of the arguments. Implement `FindMax4()` such that the body of the function comprises only a `return` statement that invokes `FindMax()` multiple times.

The program should input four values and output the max of the four.

Example

Input

```
1 2 3 4
```

Output

```
Enter 4 numbers:
maxVal is: 4
```

Part 2: Overloading function names.

In this part, you will deal with overloaded function names. All code will be in `main.cpp`. Testing and grading will be via zyLab.

Start with the template in Part 0. Change `FindMax()` and `FindMax4()` so that the arguments cannot be changed by the functions.

Write two additional overloaded functions, `FindMax()` and `FindMax4()` that have string arguments.

Input a char, `typeChoice` that will be '`d`' or '`s`' or `'x'`. If `typeChoice` == `'d'`, then input four doubles and output the max. If `typeChoice` == 's', then input four strings and output the max. If `typeChoice` == `'x'`, then end the program. If `typeChoice` is neither '`d`' nor '`s`' nor `'x'`, then request `typeChoice` again until a correct choice is indicated. Use a `switch` statement to implement `typeChoice` branching. Loop requesting `typechoice` until `'x'` is input.

Part 3: Validating a userID.

In this part, you will continue to develop your skills in writing functions as well as bringing together other material you have covered previously. Additionally, you will begin to develop code in multiple files, which is how software is generally developed in the real world. Coding will be done using clion. Testing and grading will be via zyLab; however, you should thoroughly develop your own tests paying particular attention to "boundary conditions."

Develop a program, using functions, to validate a userID. Valid userID specifications:

- 5 - 10 characters long.
- must begin with a letter.
- must contain at least one upper case letter.
- must contain at least one lower case letter.
- must contain at least one decimal digit.
- must contain at least one of the following special characters: #_$
- must not contain any other characters than those specified above.

The main program should loop, inputting userIDs and outputting the userID followed by "**`valid`**" or "**`not valid`**" as appropriate until the user enters "END". At that point the program ends.

You may only use the following system include files: `<iostream>`

You should develop a comprehensive test set of userIDs. Include this test set in a block comment at the end of your main program. I.e., in the **`main.cpp`** file.

Write two functions as described below. These should be properly entered into the two files: `ID_Functions.h` and `ID_Functions.cpp`. I.e., two functions go into one file.

Do not use a "**`using namespace`**" in the header, **`.h,`** file. Instead, use explicit namespace resolution as needed. E.g., you will have to use **`std::string`** to declare a string.

Example input and output follows template code below.

```
bool ContainsAnyOf ( … ) {
    // Two string parameters, passed by reference, and indicate they cannot
    //   be altered  by the function. Name the parameters:  anyOf, testStr
    // Returns true if any of the characters in the first string argument
    //  occur in the second argument.
    // Otherwise return false.
    // You may find it useful to use:  string::npos
    //  as discussed in class.
    // May not use any c++ functions that are not available by including only
    //  <iostream>
}
```

```
bool IsValidID ( … ) {
    // Single string parameter, which is to be tested as a valid user ID.
    // Returns true if the string argument is a valid user ID.
    // Otherwise false.
    // Must use function ContainsAnyOF to check for existence
    //   of required character types as well as proper first character.
    // If needed, strings can be concatenated using the plus, +, operator.
}
```

Example
Input

```
Kitty_8
Horse99
END
```

Output

```
Enter userID:
Kitty_8 valid
Enter userID:
Horse99 not valid
Enter userID:
```