Homework 3

Assigned: 9/21

Due: 9/27, 11:59pm

Possible Points: 100

Submit on Canvas a zip file containing your solution code for each problem (just the cpp files, please don't submit the solution/project files), turn in one cpp file for each problem. If a problem has multiple parts, it should still be one file. Each problem specifies what you should title the file containing your solution so that the autograder can find it. Incorrectly named programs will not be graded and thus will get 0 points for the problem.

Since the assignment will be graded using an automated grading system you should have your program's output match the sample output shown. If your program fails to compile using Visual Studio 2017 it will recieve zero points. If your program compiles but fails to run properly (runtime errors or wrong output) points will be deducted based on how correct the program is.

If you have any questions or concerns about the assignment do not hesitate to post on the discussion board on canvas, send us mail on canvas, or see us after class or during office hours.

Students are expected to write their source code from scratch, those who copy code from each other or online (including from Canvas examples) will be reported for cheating. We will also look at your code so definitely don't just hard-code the answer and print it.

---

1. Find the Minimum and Maximum of a List of Numbers: 10 points

File: `find_min_max.cpp`

Write a program that reads some number of integers from the user and finds the minimum and maximum numbers in this list. The first number denotes number of cases. The input for each case begins with the number of integers in the list being entered followed by the list of integers. Your program should read in this list into a fixed size array of 256 elements (**the input will not contain more numbers**) and find the minimum and maximum values and print them back out. Do not use the two functions `std::min_element` and `std::max_element` from `<algorithm>` to do this problem. Use alias for array type with `using Array = int[256]` which can be used as `Array numbers;`.

Example Input:

```
4
6
100 -50 14 32 -5 124
4
-502 123 12 -42
12
10 -60 9993 -1230 412 510 -142 -23 90 0 13 -45
7
89 32 -82 16 0 -2 78
```

Expected output:

```
Case 0:
Min: -50
Max: 124
Case 1:
Min: -502
Max: 123
Case 2:
Min: -1230
Max: 9993
Case 3:
Min: -82
Max: 89
```

---

2. Password Strength Calculator: 30 points

File: `password_strength_calculator.cpp`

Write a program that takes a password (a string of characters without spaces), and outputs how strong the password is. The password strength should be calculated in a function called by main.

Start with a base strength which is the length of the password then apply the following rules:

1. If the password contains at least one lowercase character between `a` and `z`, add 1 to the strength.
2. If the password contains at least one uppercase character between `A` and `Z`, add 1 to the strength.
3. If the password contains at least one numeric character between `0` and `9`, add 1 to the strength.
4. If the password contains at least one punctuation character, add 2 to the strength. A punctuation character is one for which the function `std::ispunct` in `<cctype>` returns true.

To compute the final strength apply the following method:

> If more than one rule is satisfied then double the strength for each extra rule. For example, satisfying 2 rules will multiply the strength by 2, satisfying 3 rules will multiply strength by 4 and so on.

Output the final strength for the password.

For this exercise, use the functions provided in the `<cctype>` standard header.

The input to your program for each case will be the password for which you should output the computed strength strength as shown below.

Example Input:

```
4
mypassword
h#a20
H.&a5kg$,o
Boo[]5T!%
```

Expected Output:

```
Case 0:
Strength: 11
Case 1:
Strength: 80
Case 2:
Strength: 120
Case 3:
Strength: 112
```

---

3. Reverse a Number: 30 points

File: `reverse_a_number.cpp`

Write a function that will reverse an integer (with the signature given). Your function should take an int as input and return the reversed int. For each case you'll read an int from `std::cin` and should use your function to reverse it and print the result. You can assume the input number will always be in the range of the type int, however you **cannot** assume all inputs to your program will be ints. Your function should **not** call any std:: functions to convert an integer to a string or vice versa.

Your program must be robust against bad input, any input that's not a positive number should result in 'Invalid input' being printed instead of trying to reverse the bad input. Revisit the I/O examples from class 8 if you need a refresher on recovering from bad input.

Your number reversing function should have the following signature, note that it should not internally use any string manipulation, you must reverse the int.

```
int reverse_number(int number)
```

Example Input:

```
6
123467
4167
lalalalalal
1000
6581029
-1241
```

Expected Output:

```
Case 0:
764321
Case 1:
7614
Case 2:
Invalid input
Case 3:
1
Case 4:
9201856
Case 5:
Invalid input
```

---

4. Frame a Name: 30 points

File: `frame_a_name.cpp`

Lets write a name framing function and use it to provide a fancy greeting to someone using our program. Input to your program for each case will be a person's full name (potentially containing spaces!) which you should print out in a frame as shown below along with the "Hello," and "Welcome to my program" text. Note that the text is also padded by a single space on each side from the framing asterisks.

You'll need to use the string `size` method to find out how long the name they entered is so that you can print a properly sized frame. You should also print the greeting 'Welcome to my program', the frame should adapt properly to names longer and shorter than this greeting. The text should have one line of padding above and below it between the lines of asterisks and a space before and after the text separating it from the vertical columns.

Because the user's name will be their fullname (potentially containing spaces!) don't forget to clear the trailing newline left after reading in the number of cases. Revisit the 'Line Based Input' section of the class 8 notes for a refresher on how to do this properly.

Example Input:

```
3
Rick
Richard Jones-Hairdo the Second
Jim
```

Expected Output:

```
Case 0:
*************************
*                       *
* Hello, Rick           *
* Welcome to my program *
*                       *
*************************
Case 1:
****************************************
*                                      *
* Hello, Richard Jones-Hairdo the Second *
* Welcome to my program                *
*                                      *
****************************************
Case 2:
*************************
*                       *
* Hello, Jim            *
* Welcome to my program *
*                       *
*************************
```