CS-102 Design Principles: Project 6

Due date: Monday, April 3 at 11:00pm

In this project, you will use the standard library's vector class to simulate part of a card game called Scout. This game was designed by Kei Kajano and was released in 2019. If you'd like to learn more about it, you can visit

https://boardgamegeek.com/boardgame/291453/scout

You may also volunteer to get your ass kicked at Scout by your professor by sending him a private message.

You may use any code that we have written together in class or posted on Discord, but not any code taken from anywhere else.

In Scout, each player is dealt a hand of cards that have the numbers `1` to `9` on them; it is illegal to rearrange the cards. Sometimes during the game, the player picks a card from the table and inserts it into their hand. An inserted card may go anywhere in the hand.

For example, if the player's hand is

```
4 8 8 6 5 4 5
```

and the player wishes to insert an `8` card, they could elect to place it between the second `8` and the `6`, so now their hand would look like

```
4 8 8 8 6 5 4 5
```

It is also fine to insert the new card at the far left or far right; for example, inserting a `7` at the far right of this most recent hand would result in

```
4 8 8 8 6 5 4 5 7
```

A wise player will choose an insertion point to make the best possible *set* or the best possible *run*. A *set* is a group of adjacent cards that all share the same value; for example, in the most recent hand shown above, the set `8 8 8` has size three, and the set `2` has size one. A *run* is a group of adjacent cards which count by ones either up or down; for example, in the most recent hand shown above, the run `6 5 4` is a run of size three, the run `4 5` is a run of size two, and the run `8` is a run of size one.

Note that a group like `7 5 6 4` is *not* a run of size four, because these cards do not count by ones either up or down.

In Scout, the following rules explain how we can determine which of two sets and/or runs is more *powerful*. A larger sized set or run is always more powerful than a smaller sized set or run. If a set and a run have the same size, then the set is always more powerful. When comparing two sets of the same size, then the set with the higher value is more powerful. When comparing two runs of the same size, then the run with the higher values is more powerful.

Here are some examples, using > to indicate *is more powerful than*:

```
6 5 4 3 > 9 9 9
  1 1 1 > 7 8
  1 1 1 > 7 8 9
    6 6 > 4 4
7 6 5 4 > 5 4 3 2
```

As an aside, note that these instructions are *overloading* the operator > to mean something other than *is greater than*. In this project, you will need to overload that operator in the same way, using code.

Example input

The input will be in this form:

```
hand_size 9
hand 4 8 8 8 6 5 4 5 7
insert 3
```

The first line indicates how many values will appear in the second line. The second line is the player's hand. The third line indicates a card that the player has picked up from the table, and must now insert into the given hand.

Example output 1

The output should be a sequence (separated by spaces, and not ending with a space) of the indices at which the insertion card can be inserted so that the hand contains the most powerful possible set or run. For the example input above, the output should be

```
7
```

because inserting the `3` at index 7 yields the hand

```
4 8 8 8 6 5 4 3 5 7
```

containing a run of size four: `6 5 4 3`. No more powerful set or run can be achieved by inserting the `3` somewhere else.

Example output 2

There may be multiple insertion points that yield the most powerful possible set or run. For example, if the input is

```
hand_size 10
hand 6 4 4 4 2 9 9 8 5 1
insert 4
```

then the output should be

```
1 2 3 4
```

because all of these choices result in a hand that has a set `4 4 4 4` of size four.

Example output 3

One more example that shows a more complex situation: if the input is

```
hand_size 8
hand 6 5 4 3 7 7 9 9
insert 8
```

then the output should be

```
0 4 5 6 7 8
```

because a run of size four already exists in the hand, and we simply want to avoid disrupting it with our inserted `8`.

These three examples do not cover all possibilities. Feel free to discuss other examples on Discord with each other!

Requirements

You must overload the operator > to mean *is more powerful than*, and you must use this operator when you compare two appropriate objects (as examples, one hand of cards compared to another, or one run of cards compared to another). The choice of classes that you use is up to you, but you must use at least one class.

Here are some additional requirements:

- ☐ Each class has its own `.h` and `.cpp` files; your class declaration should be in the `.h` file, and your method definition(s) should be in the `.cpp` file.
- ☐ Put header guards in the `.h` file, using the convention `FILENAME_H`.
- ☐ Compile all of the `.cpp` files to `.o` object files for each class.
- ☐ Link in the object files when compiling your code.

Getting started

Please start by accepting the assignment at:

`https://classroom.github.com/a/ZoiCF4r3`

which gives you a GitHub repository for your project.

How to submit:

- ☐ Add/commit/push your final code to your GitHub repository; this is the code that will be graded.
- ☐ On Gradescope, click on the assignment for this project, and submit your GitHub repository. (Select *master* as your branch.)

Grading

Your grade will come from:

75%	correctness + passing the tests
25%	style

There will be 8 tests:

- ☐ Three tests will have a single entry in the output.
- ☐ Three tests will have two entries in the output.
- ☐ Two tests will have any number of entries in the output.

Projects passing zero tests will receive at most 70% of the project grade (assuming perfect style). Passing tests increases the grade proportionally up from 70%; however, this assumes that your code isn't written to circumvent testing (for example, by "hard coding" answers to the tests), which is a violation of the honor code.

The elements of style that we will assess on this project are:

- ☐ You used at least one class.
- ☐ You overloaded the operator > to mean *is more powerful than*, and used the overloaded operator appropriately.

- ☐ You used header files (and so on) as described in the Requirements section.

  We will give you one aggregate grade for the rest of these style elements:

- ☐ Use `const` where appropriate, both for functions and for parameters.
- ☐ Your helper functions should be no more than about 20 lines long (not including spacing for curly braces, and not counting comments); each function should have one purpose, and be named according to that purpose.
- ☐ Use appropriate programming constructs; e.g., don't use a loop when an if statement is sufficient, a while loop when a for loop is better, and so on.
- ☐ Prune your dead code: remove commented out code, unused variables, other code not contributing to the overall project.
- ☐ Include all file and function header comments, using the required format.
- ☐ Declare variables near their initial use.
- ☐ Indent properly and use correct placement of curly braces and parentheses.
- ☐ Use correct spacing around operators.
- ☐ Use reasonable variable names in proper naming style: using `lower_case` naming in local scope, and `UPPER_CASE` naming for global constants.