CS 150 Lecture 21 Exercises

Complete each of the exercises below and upload to Canvas before the deadline.

A. Partially-Filled Array Basics

Open **partial.cpp** and complete the following three **template functions** which should work for any **numeric type**. Use **make** **test** to test your code.

**toString()**

* **Parameters**: the array and the number of elements
* Returns a string with each element separated by a space (no commas).  
   The elements should be surrounded by brackets **[ ]**

**readArray()**

* **Parameters**: the array (of type **T**) and the **capacity** (reads from **cin**)
* Returns the **size** of the array (as **size\_t**)
* A non-numeric sentinel or filling the array will end the input.

**indexOf()**

* **Parameters**: the array, a pointer to the number of elements, and the value to find
* Returns the index of the first occurrence (as a **size\_t**)
* If not found, return the constant (**NOT\_FOUND**) defined in **partial.h**

You'll find the prototypes in **partial.h**. Test with **make** **test**.

| Copy and paste a screenshot of the functions here. |
| --- |

| Copy and paste a screenshot the tests running |
| --- |

B. Partially Filled Array & Erasing

Open **partial.cpp** and complete the **erase()** **template function** which searches for an element in the partially filled array and removes it if found, keeping the array in order. Use **make** **test** to test your code.

**erase()**

* **Parameters**: the array, the number of elements (output parameter), value to remove.
* Returns **true** if an item was removed, **false** if not.

You'll find several examples inside **run()**.

| Copy and paste a screenshot of the erase function here. |
| --- |

| Copy and paste a screenshot the tests running |
| --- |

C. Partially Filled Array & Insertion

Open **partial.cpp** and complete the **insert()** **template function** which adds an element to the partially filled array keeping the array in order. Use **make** **test** to test your code.

**insert()**

* **Parameters**: the array, the number of elements (output parameter), capacity and value to insert.
* Returns a pointer to the inserted element if successful or **nullptr** if not.

You'll find several examples inside **run()**.

| Copy and paste a screenshot of the insert function here. |
| --- |

| Copy and paste a screenshot the tests running |
| --- |

D. 2D Arrays

Open **a2d.cpp**, **a2d.h** and **sample.cpp**. The sample program creates a 2D array of **int**, then passes it, twice, to a printing function. Look in the header file, and notice that **aout()** takes a default argument, **cols**, set to the declared column size of the 2D array. Notice also, that while we can't leave the declared size of the array blank, the actual **col** parameter may be a different size.

Using **aout()** as a mode,l write three overloaded **average()** functions. Each will return the average value (as a **double**) in the array. The first, will visit all of the elements specified by **rows** and **cols**. Notice that for the column size, you don't need to use the declared constant. In our case, the declared column size of our 2D array (set in the header file) contains elements we don't want to process. Our actual array, (**values**), only has 4 meaningful elements in each row, so we'll want to use that when we process it.

The second version of **average()** is used to average the elements in a row. Even though your data is stored in a 2D array, you can pass the individual rows to a function that works with 1D arrays.

Finally, do the same thing with a function that averages all the elements in one column of a 2D array. Of course there is no convenient notation to pass only a single column to your function, so it will have to take the same arguments as your first **average()** function, along with a final argument indicating which column should be averaged. When you are finished, run the tests.

| *Copy and paste a screenshot of the source code for your average functions here.* |
| --- |

| *Copy and paste a screenshot of your tests (make test)* |
| --- |

# 