Exercise – Templates

In these exercises we will practice using template functions and template classes.

Exercise 1:

Create the following template functions:

1. **Min**

   DESCRIPTION: Returns the lower of two values.
   INPUT: Two template arguments *(a, b)*.
   OUTPUT: The lower of the two values *a* and *b*.

2. **Max**

   DESCRIPTION: Returns the higher of two values.
   INPUT: Two template arguments *(a, b)*.
   OUTPUT: The higher of the two values *a* and *b*.

3. **Clamp**

   DESCRIPTION: Constrains a value within the range of two other values.
   INPUT: Three template arguments *(min, max, val)*.
   OUTPUT: The value of the *val* constrained between *min* and *max*.

4. **Min** (specialised for char)

   DESCRIPTION: As above but first checks if *a* and *b* represent alphabetical characters. If so, the function will return the value that is alphabetically lower.
   INPUT: Two template arguments *(a, b)*.
   OUTPUT: The alphabetically lower of the two values *a* and *b*.

5. **Max** (specialised for char)

   DESCRIPTION: As above but first checks if *a* and *b* represent alphabetical characters. If so, the function will return the value that is alphabetically higher.
   INPUT: Two template arguments *(a, b)*.
   OUTPUT: The alphabetically higher of the two values *a* and *b*.

Exercise 2:

Create a template class for storing any data-type. The data itself should be stored as a private pointer to an array on the heap.

Your class should have the following public template functions:

1. **Constructor**

   DESCRIPTION: Creates an array on the heap of the specified capacity. The new heap memory should be zeroed using *memset*. You should store the value of capacity and size for later use (size should start at 0).
   INPUT: A single integer specifying the capacity for the data array.
   OUTPUT: N/A.

2. **Deconstructor**

   DESCRIPTION: Deletes the data pointed to on the heap.
   INPUT: N/A.
   OUTPUT: N/A.

3. **Add**

   DESCRIPTION: Adds an item to the next empty array location (you should be able to use the size value as an index to the next free location). Before adding the new item to the array, we should check if there is room and call *Expand* if not.
   INPUT: One template arguments *(item)*.
   OUTPUT: None.

4. **Expand**

   DESCRIPTION: Doubles the capacity of the data array by creating a new array on the heap with twice the current capacity, and then copying the data from the current array into the new array using *memcpy*. (Be sure to delete the old memory!)
   INPUT: None.
   OUTPUT: None.

5. **Operator []**

   DESCRIPTION: Returns an item from the array at a specified index (like a regular subscript).
   INPUT: A single integer specifying the array-index of the item to return.
   OUTPUT: A copy/reference (your choice) of an item from the data array.