**DTS Lab 1  
*dynamic array***

**Objective**

Implement a template class, *DynArray,* that will encapsulate the dynamic array functionality discussed in the Day 1 lecture materials. The default expansion rate should be double the current capacity with a minimum of 1.

Place all your code in a file named *DynArray.h*

**Prototypes**

The DynArray class will have the following public interface:

/////////////////////////////////////////////////////////////////////////////  
// Function : Constructor  
// Notes : Constructs an empty array (Size 0 Capacity 0)  
/////////////////////////////////////////////////////////////////////////////  
DynArray()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Destructor  
// Notes : cleans up any dynamic memory  
/////////////////////////////////////////////////////////////////////////////  
~DynArray()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Copy Constructor  
/////////////////////////////////////////////////////////////////////////////  
DynArray(const DynArray<Type>& that)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Assignment Operator  
/////////////////////////////////////////////////////////////////////////////  
DynArray<Type>& operator=(const DynArray<Type>& that)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : operator[]  
// Parameters : index - the index to access  
// Return : Type & - returns the array element at [index]  
// Notes : performs no error checking. user should ensure index is   
// valid with the size() method  
/////////////////////////////////////////////////////////////////////////////  
Type& operator[](const unsigned int index)

/////////////////////////////////////////////////////////////////////////////  
// Function : operator[]  
// Parameters : index - the index to access  
// Return : const Type & - returns the array element at [index]  
// Notes : performs no error checking. user should ensure index is   
// valid with the size() method  
/////////////////////////////////////////////////////////////////////////////  
const Type& operator[](const unsigned int index) const

/////////////////////////////////////////////////////////////////////////////  
// Function : size  
// Returns : int - returns the number of items being stored  
// Notes : this function returns the number of items being stored,   
/////////////////////////////////////////////////////////////////////////////  
unsigned int size() const

/////////////////////////////////////////////////////////////////////////////  
// Function : capacity  
// Returns : int - returns the number of items the array can store before   
// the next resize  
// Notes : this function returns the number of items the array can store,   
// not the number of bytes  
/////////////////////////////////////////////////////////////////////////////  
unsigned int capacity() const  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : clear  
// Notes : cleans up any dynamic memory and resets size and capacity to 0  
/////////////////////////////////////////////////////////////////////////////  
void clear()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : append  
// Parameters : item - the item to be appended to the next open spot  
// Notes : this function will append the item to the next open spot. if   
// no room exists, the array's capacity will be doubled and then   
// the item will be added  
/////////////////////////////////////////////////////////////////////////////  
void append(const Type& item)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : reserve  
// Parameters : newCap - the new capacity  
// Notes : - default parameter - reserve more space in the array, based on // the expansion rate (100%, 1 minimum).  
// - non-default parameter, expand to the specified capacity  
// - if newCap is LESS than the current capacity, do nothing.   
// This function should NOT make the array smaller.  
/////////////////////////////////////////////////////////////////////////////  
void reserve(const unsigned int & newCap = 0)

**Desired Output**

Compile and run your code with the DTSLab1.cpp file provided via FSO. Your console output should match the following block identically:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\* LAB 1: \*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
\*\*\* TEST 1 \*\*\*  
{ } Size 0 Capacity 0  
{ 0 } Size 1 Capacity 1  
{ 0 4 } Size 2 Capacity 2  
{ 0 4 8 } Size 3 Capacity 4  
{ 0 4 8 12 } Size 4 Capacity 4  
{ 0 4 8 12 16 } Size 5 Capacity 8  
{ 0 4 8 12 16 20 } Size 6 Capacity 8  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 8  
  
\*\*\* TEST 2 \*\*\*  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 16  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 16  
{ } Size 0 Capacity 4  
{ 0 } Size 1 Capacity 4  
{ 0 4 } Size 2 Capacity 4  
{ 0 4 8 } Size 3 Capacity 4  
{ 0 4 8 12 } Size 4 Capacity 4  
{ 0 4 8 12 16 } Size 5 Capacity 8  
{ 0 4 8 12 16 20 } Size 6 Capacity 8  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 8  
  
\*\*\* TEST 3 \*\*\*  
{ } Size 0 Capacity 10  
  
\*\* TEST 4 \*\*\*|  
{ 1 2 3 5 } Size 4 Capacity 4  
  
\*\*\* TEST 5 \*\*\*  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 8  
{ } Size 0 Capacity 0  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 8  
{ } Size 0 Capacity 0  
  
\*\*\* TEST 6 \*\*\*  
{ } Size 0 Capacity 0  
{ } Size 0 Capacity 1  
{ } Size 0 Capacity 2  
{ } Size 0 Capacity 4  
{ 0 1 } Size 2 Capacity 4

\*\*\* TEST 7 \*\*\*  
{ 0 1 2 3 } Size 4 Capacity 4  
  
\*\*\* TEST 8 \*\*\*  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 8  
{ 28 } Size 1 Capacity 1  
{ 28 } Size 1 Capacity 1  
  
\*\*\* TEST 9 \*\*\*  
{ 1.1 2.2 3.3 4.4 5.5 } Size 5 Capacity 8  
{ 1.1 2.2 3.3 4.4 5.5 } Size 5 Capacity 8  
  
\*\*\* TEST 10 \*\*\*  
{ 0 4 8 12 16 20 24 } Size 7 Capacity 8

**Challenge**

Add the following functions to your DynArray class. These will allow the user to insert or remove single or multiple items from valid indices in the array.   
\*these functions should only insert or remove at valid indices.

/////////////////////////////////////////////////////////////////////////////  
// Function : insert  
// Parameters : val - the value to insert  
// index - the index to insert at  
// Notes : if the array is full, this function should expand the array at   
// the default expansion rate (double the capacity, 1 minimum)  
/////////////////////////////////////////////////////////////////////////////  
void insert(const Type val, const unsigned int index)

/////////////////////////////////////////////////////////////////////////////  
// Function : insert  
// Parameters : val - the items to insert  
// n - the number of items to insert  
// index - the index to insert at  
// Notes : if the array is full, this function should expand the array at   
// the default expansion rate (double the capacity, 1 minimum)   
// before inserting  
/////////////////////////////////////////////////////////////////////////////  
void insert(const Type \* val, const unsigned int n, const unsigned int index)

/////////////////////////////////////////////////////////////////////////////  
// Function : remove  
// Parameters : index - the index to remove from  
// Notes : this function removes one item from the specified index  
/////////////////////////////////////////////////////////////////////////////  
void remove(const unsigned int index)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : remove  
// Parameters : index - the index to remove from  
// n - the number of items to remove  
// Notes : this function removes multiple items from the specified index  
/////////////////////////////////////////////////////////////////////////////  
void remove(const unsigned int index, const unsigned int n)

To test these functions, we'll use the same .cpp file as before, but you will need to change the CHALLENGE\_ACCEPTED macro from 0 to 1.

Compile and run your code now and the first 9 tests should remain unchanged, but output for tests 10 and 11 will be added. if your code works correctly, the new output should match the output below

\*\*\* TEST 11 \*\*\*  
{ } Size 0 Capacity 0  
{ 99 } Size 1 Capacity 1  
{ 28 99 } Size 2 Capacity 2  
{ 24 28 99 } Size 3 Capacity 4  
{ 20 24 28 99 } Size 4 Capacity 4  
{ 16 20 24 28 99 } Size 5 Capacity 8  
{ 12 16 20 24 28 99 } Size 6 Capacity 8  
{ 8 12 16 20 24 28 99 } Size 7 Capacity 8  
{ 4 8 12 16 20 24 28 99 } Size 8 Capacity 8  
{ 0 4 8 12 16 20 24 28 99 } Size 9 Capacity 16  
{ 0 1 2 3 4 8 12 16 20 24 28 99 } Size 12 Capacity 16  
  
\*\*\* TEST 12 \*\*\*  
{ 0 1 2 12 16 20 24 28 99 } Size 9 Capacity 16  
{ 1 2 12 16 20 24 28 99 } Size 8 Capacity 16  
{ 2 12 16 20 24 28 99 } Size 7 Capacity 16  
{ 12 16 20 24 28 99 } Size 6 Capacity 16  
{ 16 20 24 28 99 } Size 5 Capacity 16  
{ 20 24 28 99 } Size 4 Capacity 16  
{ 24 28 99 } Size 3 Capacity 16  
{ 28 99 } Size 2 Capacity 16  
{ 99 } Size 1 Capacity 16  
{ } Size 0 Capacity 16

**Submission**

To submit the lab assignment:

* Clean, build, and run DTSLab1.cpp with your DynArray.h file in Visual Studio (debug mode).
* clear up any warnings you encounter.
* verify that your output is correct by comparing it to the lab document's Desired Output section, line-by-line.
* ensure there are no memory leaks.
* On your desktop, create a new folder with your name in the following format:
* your last name
* a comma
* a single space
* your first name  
  **\* Appropriate capitalization for proper names should be used.**  
  Suitable examples include : "Pollack, Joey"; "De La Paz, Christhian"; "Tjarks, Matthew".
* Copy your 'DynArray.h' file into the folder that you created in step 2. This is the only file I need and should therefore be the only file you submit.
* Right-click on the folder and select 'send to->compressed (zipped) folder'.
* Submit the compressed folder via FSO.