7.21 Coding lab #3: use separate files to manage a dynamic array via pointers.

Honor Code

- Your answers to this homework must be your own work.
- You are not allowed to share your solutions.
- You may not engage in any other activities that will dishonestly improve your results or dishonestly improve or damage the results of others.

Plagiarism

Plagiarism is when you copy words, ideas, or any other materials from another source without giving credit. Plagiarism is unacceptable in any academic environment.

Important, please read!

- Make sure you use the specified function prototypes, as they will be referred to as such in the unit tests.
- Please feel free to introduce additional subroutines (i.e., functions) to help you implement the required functions, although they will not be tested.
- zyBook's compilation environment is not the same as your IDE! zyBook's C++ compilation environment seems to be "bare-bone". It's much closer to programming without help from a sophisticated IDE such as NetBeans or Visual Studio. I would recommend you to test your programs using a less-embellished IDE such as Dev C++ or the online site cpp.sh as recommended by Scott on iLearn.
- Submit three files as described below. Please name your files as specified to avoid unnecessary complications (even though I don't think it actually matters.) You can submit your solution for grading up to 50 times.
- Your attitude matters! Believe or not, if you hold the belief that your codes are perfect and they should pass the unit tests on zyBook, this will effectively set up a mental barrier that prevents you from fixing the issues faster!

The main objectives of this lab include:

- Use pointers to manage a dynamic array of integers, including
  - memory allocation & value initialization
  - resizing
  - changing and reordering the contents of an array
  - memory deallocation
- Learn the difference between passing a pointer to a function by value vs. by reference
- Learn to use *pointers to functions* to make a function more powerful and template-like.
- Learn to include multiple files in a C++ project by distributing the source codes accordingly to different files.

1. Your project will include the following three files:

- A header file: dynamicArray.h that includes a list of function prototypes as enumerated in the next section.
- An implementation file: dynamicArray.cpp that implements the functions declared in the header file.
- A test driver file: dynamicArray-main.cpp that includes the *main()* function so that you can test all the functions you've implemented above.

2. The header file **dynamicArray.h** will include the following list of functions:

- constructing a dynamic array of the specified *size* and initializing the *i-th* array element to *i*i*

```
int * array_constructor(int *  &intPtr, int &size );
```

Specifically, the above function allocates space to *intPtr* and uses it to manage a dynamic array of *size* integers. Use proper exception handling to make sure that the allocation is successful. Then initialize the value of the i-th element to *i*i*. The function returns a pointer pointing to the new array. To avoid memory leakage, you should check whether *intPtr* already has a valid pointee when being passed into this function. If the answer is yes, you would want to first deallocate the space occupied by its pointee.

As an example, after invoking this function `array_constructor( myArray, size=5);` in a different function, the content of myArray will be

```
myArray[0]=0
myArray[1]=1
myArray[2]=4
myArray[3]=9
myArray[4]=16
```

- resizing a dynamic array pointed to by *intPtr*, where the new size can be smaller or larger than the array's current size

```
int * array_resize(int * &intPtr, int& currSize, int& newSize);
```

You will need to first make sure both *curSize* and *newSize* have valid values, i.e., positive integers. Then consider the following three scenarios:

- currSize==newSize or newSize<0: do nothing
- currSize>newSize: the array's size is reduced to *newSize*. Furthermore, its content is reduced to its first *newSize* elements.
- currSize<newSize: the array's size is increased to *newSize*. The content of the array will be expanded by inserting at the end as many elements as needed to reach *newSize*. Furthermore, initialize each of the new elements to *i*i*, where *i* is the index of the element.

As an example, after invoking this function **`array_resize( myArray, currSize, newSize);`** (where currSize=5, newSize=9), the content of myArray will be changed to

```
myArray[0]=0
myArray[1]=1
myArray[2]=4
myArray[3]=9
myArray[4]=16
myArray[5]=25
myArray[6]=36
myArray[7]=49
myArray[8]=64
```

Later, another invocation **`array_resize( myArray, currSize, newSize);`**(where currSize=9, newSize=2) will change the content of myArray to:

```
myArray[0]=0
myArray[1]=1
```

- deallocating the memory space occupied by the dynamic array *intPtr*. Please make sure you check whether this array actually exists. After you finish deallocating the array space, make sure to assign *nullptr* to the pointer.

```
void array_destructor(int * &intPtr);
```

- Randomizing the content of the dynamic array *intPtr* by calling the **`srand()`** and **`rand()`** functions (see zyBook section 2.19).

```
void array_set(int* &intPtr, int &size);
```

Specifically, after having set a seed value using the **`srand( time(0) )`** function. Then invoke the **`rand()`** to assign each element in the array a random value.

As an example, after invoking **`array_set( myArray, currSize);`** (where currSize=9) in a different function, *myArray* will look like (yours will be different, of course):

```
myArray[0]=415960052
myArray[1]=981322979
myArray[2]=420899093
myArray[3]=239922833
myArray[4]=1556248812
myArray[5]=1670446471
myArray[6]=1140120866
myArray[7]=14812681
myArray[8]=1996110162
```

- Passing a *pointer to functions* to a sorting function **`mysort()`** so that this function can either sort an array in ascending or descending order. Please modify the *insertionSort()* you implemented in the last coding lab to sort an integer array. (Please feel free to use the implementation in the sample solution posted on iLearn if yours didn't pass the test.)

```
void mysort( int* &intPtr, int size, bool (* comp)(int&, int&) );
```

To do this, please include the following two boolean functions for integer comparison in your header file: `bool my_less_equal(int& x, int & y); //return true if x<=y, false otherwise. bool my_greater_equal(int& x, int & y ); //return true if x>=y, false otherwise.` Now, if one calls `mysort( myArray, size, my_less_equal);` in another function, the content in *myArray* will be sorted in ascending order; calling `mysort( myArray, size, my_greater_equal);` will sort *myArray* in descending order.

3. The implementation file **dynamicArray.cpp** will implement all the functions declared in the above **dynamicArray.h** header file.

4. A test driver file: **dynamicArray-main.cpp** that includes the *main()* function so that you can test all the functions you've declared and implemented in the above two files.

LAB ACTIVITY 7.21.1: Coding lab #3: use separate files to manage a dynamic array via pointers. 0 / 36

Submission Instructions

Deliverables

`dynamicArray-main.cpp` , `dynamicArray.h` and `dynamicArray.cpp` *You must submit these file(s)*

Compile command

`g++ dynamicArray-main.cpp dynamicArray.cpp -Wall -o a.out` *We will use this command to compile your code*

Submit your files below by dragging and dropping into the area or choosing a file on your hard drive.

dynami...n.cpp	**dynami...ray.h**	**dynami...y.cpp**
Drag file here or **Choose on hard drive.**	Drag file here or **Choose on hard drive.**	Drag file here or **Choose on hard drive.**

Submit for grading 50 submissions left This lab can only be submitted once every 1 minute

Latest submission

No submissions yet