Homework 6: Dynamically Growing Arrays in C++

Assigned Thursday, November 7, 2024. Due Sun. Nov. 17 at 11:59pm on Canvas

Introduction

A dynamically growing array, also referred to as a **vector**, is a common data structure used to store a set of elements that can vary in number throughout the execution of a program. When a vector is created, an initial region of memory is assigned to it. As the first few elements are inserted into the vector, this memory region is progressively occupied, until eventually it fills up. When a new element is inserted at this moment, the capacity of the vector must be increased in order to make additional room. Increasing the capacity of the vector involves reallocating its memory, copying the content of the original vector into the new vector, and freeing the old vector. This is a costly process, and thus, we should avoid repeating it with every element insertion. Instead, we will add room for multiple additional elements every time the vector capacity grows. Our choice will be doubling the vector capacity every time we exceed the current storage limit.

1. First Step

To be able to test your dynamically growing array program right away, we will start by writing code for an interactive main program that asks the user to select an element from a menu containing a set of possible actions. Name your source file `hw6.cpp`

The program should display the following message:

```
Main menu:
    1. List vector's contents
    2. Append element at the end
    3. Remove last element
    4. Insert one element
    5. Exit
Select an option: _
```

Write a C++ program that displays the menu shown above and waits for the user to enter an option. If the option is invalid, an error message should be displayed, and the main menu should be shown again. When the user selects option 5, the program should finish. When the user enters any other valid option,

your pre-lab program should just print the name of the option on the screen. To develop the full program you will replace the code for each option with its actual functionality. An example of selecting an option is shown below:

Select an option: 4
You selected “Insert one element”

After a valid option is selected and the proper message is displayed, the main menu should be displayed again. Use a switch control flow statement to manage the menu.

2. Growing the Vector

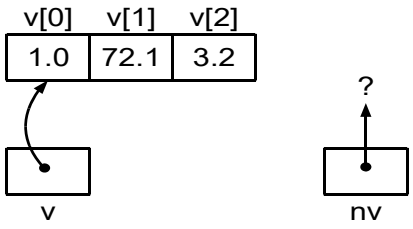
Edit the hw6.cpp program you implemented to define a vector of double-precision floating-point numbers. This vector can be represented with the following three global variables:

- double *v. This is a pointer to the first element in a sequence of elements appearing consecutively in memory.
- int size. This variable represents the number of elements inserted in the vector so far by the user. Its initial value is 0.
- int capacity. This variable represents the space currently allocated for the vector, given in number of elements. We will initialize its value to 2.

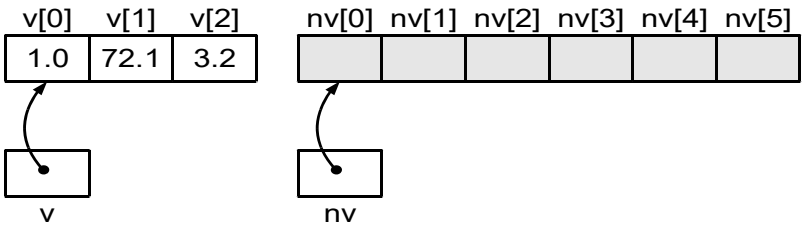
Add two functions to your program: Initialize() and Finalize(). The main() function in your program should invoke functions Initialize() when the program begins, and Finalize() right before it ends. The former will initialize the three global variables associated with the vector with valid values and allocate memory for the vector with an initial capacity of just 2 elements. The latter will free the memory associated with the vector.

Now add a function that grows the capacity of the vector. This function should increase the vector's allocated storage while keeping the same set of elements in it, using the following steps:

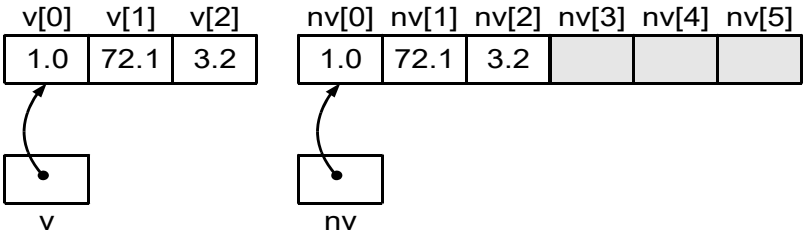
- Initially, we have a vector v that has reached its full capacity, and an uninitialized pointer nv to what will be the new vector.



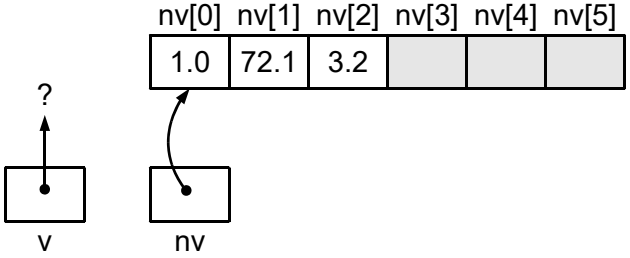
- We first allocate a new memory region that will serve as the new storage for the vector, with the desired new capacity.



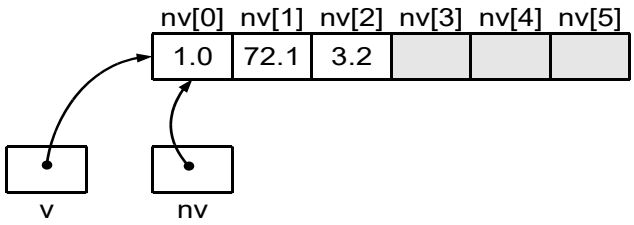
- All elements in the old vector are copied into the new vector. The additional elements in the new vector remain uninitialized.



- The memory originally allocated for old vector is now freed.



- We make the old vector pointer v point to the new memory region referenced by nv. We can now continue to use v normally for future insertion, deletion, search, or update operations.



Show the screen shot of running the new code for function Grow(). The function should print debug information on the screen, including the old and new capacities of the vector. For example:

Vector grown

Previous capacity: 2 elements

New capacity: 4 elements

3. Adding an Element at the End

Adding an element at the end of the vector may require growing its capacity. We will do so only when the current number of present elements is equal to the capacity of the vector, by invoking function Grow(). Once we make sure that there is enough storage capacity for the new element, a new element can be safely inserted at the end of the vector.

Write function AddElement(). The body of this function should ask the user to enter a floating-point number, which will be the new value added at the end of the vector.

Enter the new element: _

Write another function named PrintVector(), which should display the current content of the vector. The functionality to add an element and to print the vector should be now available through options 2 and 1 in the menu, respectively.

Include screenshots showing the output of your program for an execution where you add several elements (enough to make the vector grow at least once), and then display its current content.

4. Removing an Element from the End

This action is accessible through option 3 in the main menu. If the vector is empty and the user selects this option, a proper error message should be displayed, indicating that there are no elements left to remove.

Write function RemoveElement() with the described functionality.

Include screenshots showing the output of the program for two execution scenarios: one where removing the last element happens successfully, and one where the function is invoked on an empty vector.

5. 20 points Extra Credit: Inserting an Element

In general, inserting an element at a random position of the vector involves shifting all elements that appear on its right before the actual insertion to make room for the new element. In case the current number of elements in the vector is equal to the capacity of the vector, then you will need to grow the vector before inserting the new element.

A generic insertion operation requires two pieces of information: the index that will be occupied by the new element and the element to be inserted itself. These two values need to be requested from the user. Notice that the valid range for the index is between 0 and n, where n is the current size of the vector. *Note:* The value n for the insertion index is equivalent to appending the new element at the end of the vector as the index of the current last element is n-1

Write function InsertElement() and make it accessible through option 4 on the menu. The function should ask the user for an index and a value for the new element. The index should be checked for correct boundaries, and a proper error message should be displayed if the entered value is invalid.

Enter the index of new element: _

Enter the new element: _

Include screenshots showing the output of your program for an execution where you enter several elements at different intermediate positions. Cover the case where you enter an invalid value for the index.

Submitting Your Homework

1. (55 points total) Source code with comments explaining your program.
    a. (5 points) General comments explaining overall program
    b. (10 points each) comments explaining each section (1 – 5) of the code
2. (45 points total, 9 points each) Screen shots as asked for in sections 1 – 5.
3. Add up to 20 points extra credit depending on how well the student implemented the extra credit challenge.
    a. 10 points comments
    b. 10 points screen shot of working code