Lab Assignment 7

Dynamically Growing Arrays in C++

Zhenming Yang, Connor McEleney

[yang.zhenm@northeastern.edu](mailto:yang.zhenm@northeastern.edu), [mceleney.c@northeastern.edu](mailto:mceleney.c@northeastern.edu)

Instructor: Prof. Zagieboylo

[d.zagieboylo@northeastern.edu](mailto:d.zagieboylo@northeastern.edu)

Due Date: November 2nd, 2023

Submit Date: November 2nd, 2023

**Abstract**

In this lab, we created a dynamically sized array in c++. The program allows user to add, insert, delete, and print the array through the terminal.

# Introduction

A dynamically expanding array, often called a vector, serves as a widely used data structure for accommodating a collection of elements that can change in size during a program's execution. When a vector is initially created, a chunk of memory is allocated for it. As the first few elements are inserted into the vector, this memory segment is gradually occupied until it reaches its limit. At this point, when a new element is inserted, the vector's capacity must be increased to provide additional space. Expanding the vector's capacity involves reallocating its memory, copying the contents from the original vector to the new one, and freeing the old vector. This process can be resource-intensive, and, therefore, it's advisable to minimize the frequency of such operations when inserting elements. Instead, we opt to allocate space for multiple additional elements each time the vector's capacity is extended. Specifically, our approach will be to double the vector's capacity whenever we surpass the current storage limit.

# Lab Setup

## Pre-Lab

## Equipment

DE1-SoC:

* The DE1-SoC is a hardware design platform built around the Altera System-on-Chip (SoC) FPGA. The DE1-SoC is designed for experiments on computer organization and embedded systems. It includes embedded processors, memory, audio and video devices, and some simple I/O peripherals.

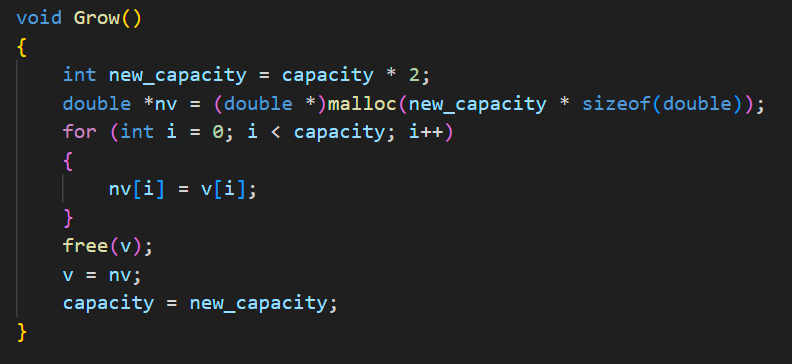
# Results and Analysis

**Results**

## Part 1: Growing the Vector

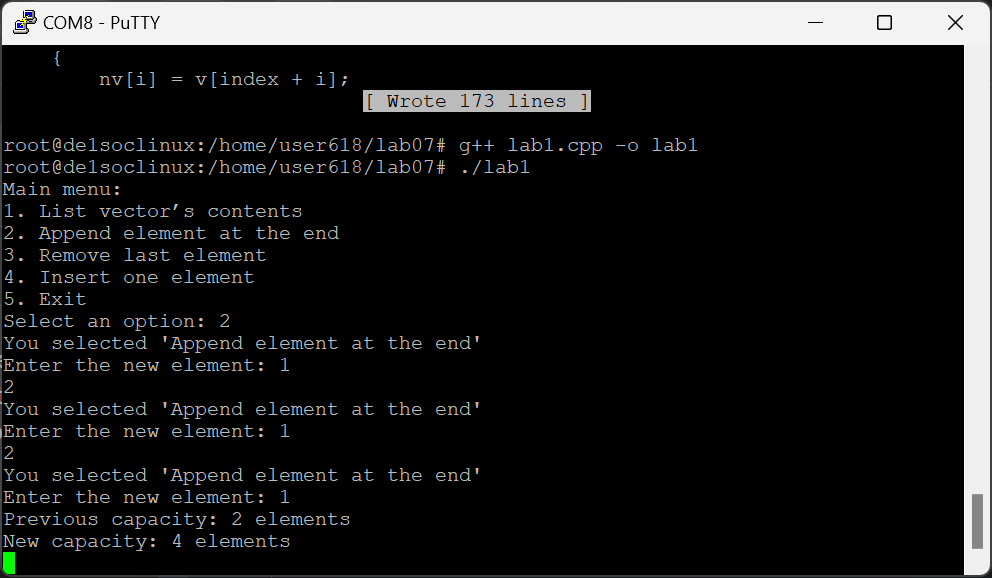
Based on the program we created in the pre-lab, we created two classes, Initialize() and Free(), and three global variables, v, size, and capacity, to initialize the array and free the memory when it array is finished using. We defined the variables to be “v” which is the array, “size” which is the current array size with value 0, and “capacity” which is the maximum capacity that the array can store with value 2. The Initialize() function allocates memory for the array and the Free() function frees the allocated memory.

We then created the Grow() function, which basically double the capacity of the array v. The function basically creates a new array nv and copy every entry in the original array v to the new array. After copying everything over, the old v array is freed and v is redefined to point at the nv array (Figure 1).



**Figure 1**: The Grow() function doubles the capacity of the original array and copy paste the items from the original array to the new array.

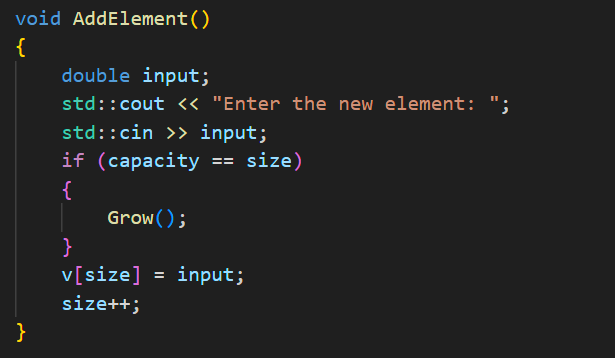
We tested the growth function, and we got the correct debug information on the terminal (Figure 2).



**Figure 2**: The terminal output of the Grow() function debug information.

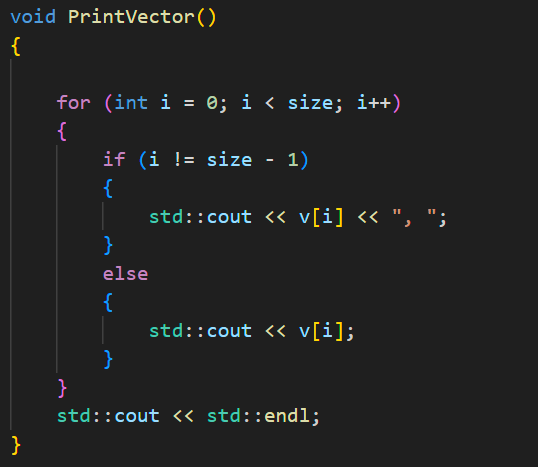
## Part 2: Adding an element at the end

We then created a new function AddElement() to add element to our empty initialized array. The function first prompt the user to input the element they want to add and record the input in an int variable. it then check if the current size variable is equal to the capacity or not. If they are equal the Grow() function is called and the capacity is doubled. And then the new element is added at the end of the array and the size is incremented by 1 (Figure 3).



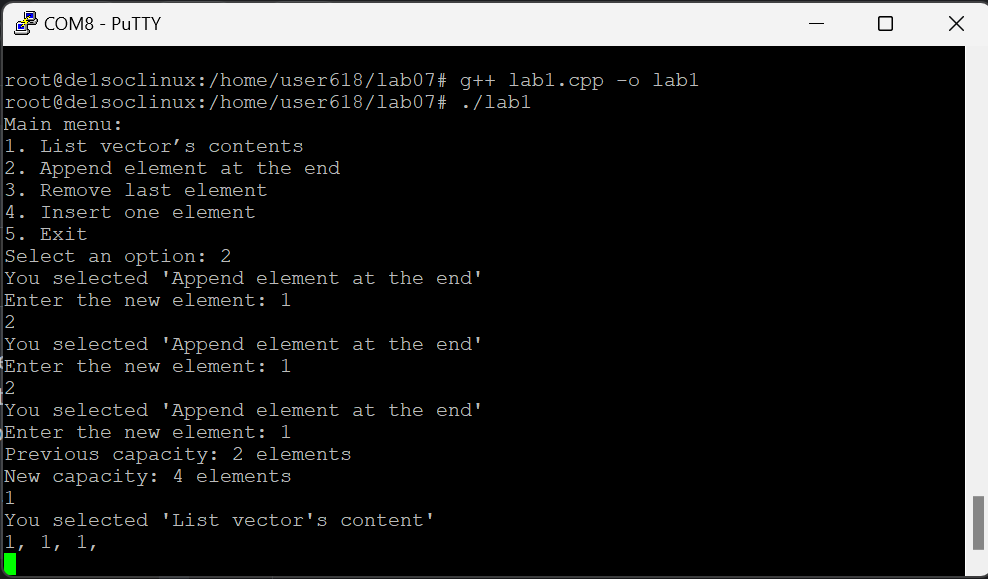
**Figure 3**: The AddElement() function add a user input number at the end of end of the array and grow the array when the size exceed capacity.

We also created the PrintVector() function to test our functions to see if the new elements are correctly added to the array. The PrintVector() function basically loop through the array and print the element separated by a comma (Figure 4).



**Figure 4**: The PrintVector() function loop through the array and print the element on the terminal.

These two functions are added to the user control menu and now the user adds element/display array using the control. We tested the add and print functionalities and they work as intended (Figure 5).



**Figure 5**: The array dynamically increases the capacity based on the how many elements are added into the array. The program also correctly outputs the array that the user had stored into the system.

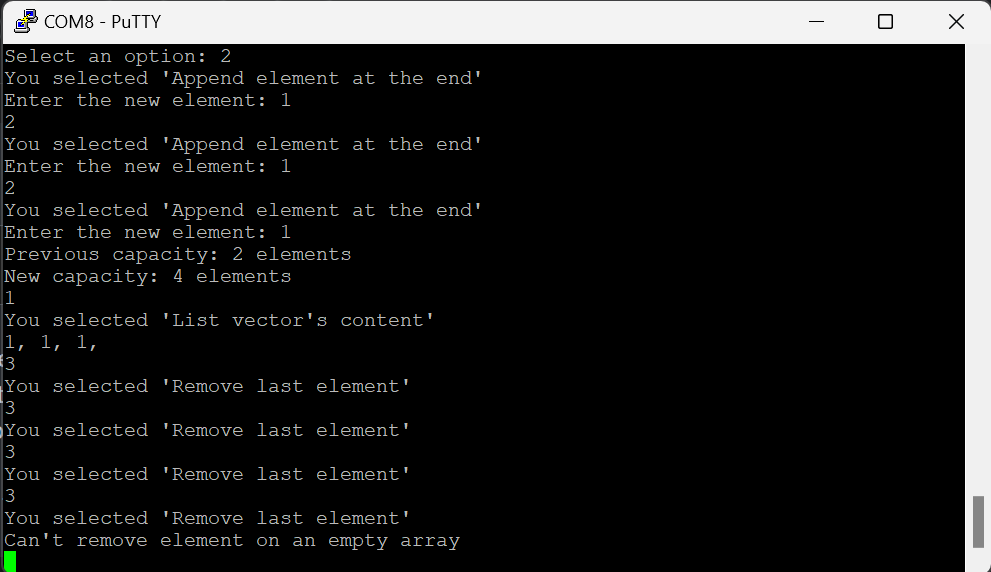
## Part 3: Removing an element from the end

We then created the RemoveElement() function to delete the last element of the array. The function first checks if the size is 0. Then it minus 1 for the size (Figure 6).



**Figure 6**: The RemoveElement() function checks the size and also reduce the size to remove the last element.

We tested the function when there are elements in the array and when it is empty. The terminal displays the correct result (Figure 7).



**Figure 7**: The element in the array is being removed tile there are no more elements left. The terminal gives the correct output and indicates that the array is empty.

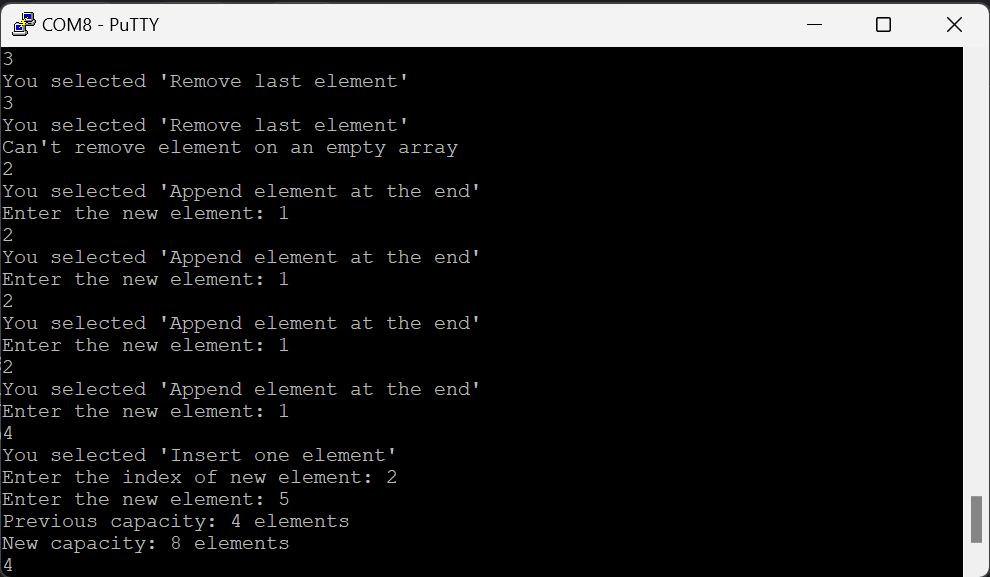
## Part 4: Inserting an element

We created a new function InsertElement() to insert element to a specific index. The function first as for the insert index and the element the user wants to insert into the array. Then, it copies every element after the insert index to a new array, changes the old array’s at the insert index while keeping everything the same, and iterates through the new array and puts all the elements back to the original array after the index (Figure 8). The size is also checked with the capacity to trigger the grow function just like the case in the AddElement() function. The size is then increased by 1 at the end of the function.



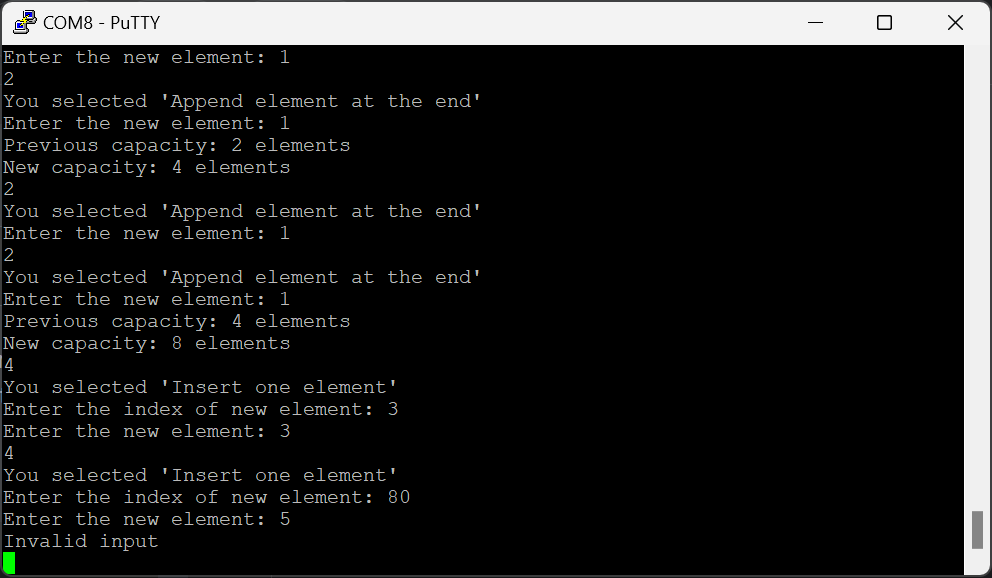
**Figure 8**: The InsertElement() function takes the user specified index and element and inserted into the array while shifting every elements after by 1. Noted the Grow() function will be triggered if the capacity is full just like the AddElement() function

We tested the function in the terminal, and we got the correct respond (Figure 9).



**Figure 9**: The InsertElement() function correctly triggers the Grow() function and insert the element to the correct index.

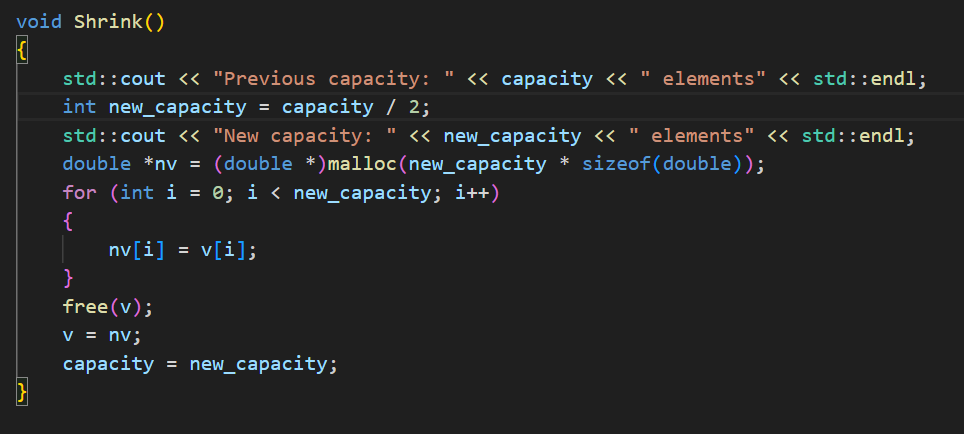
We also input an index outside the size of the array, it shows the correct error message (Figure 10).



**Figure 10**: The insert index is 80, which is outside the size of the array and an "Invalid input" message is displayed instead.

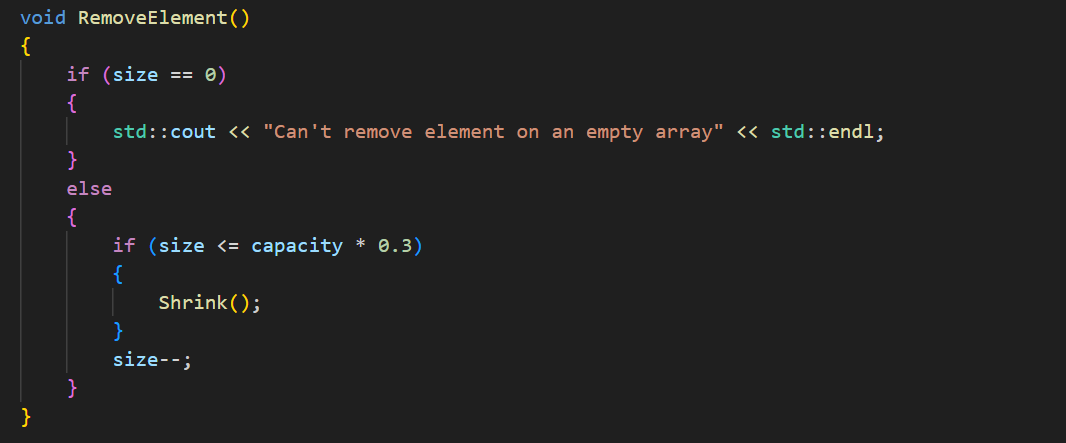
## Part 5: Shrinking the vector

At last, we created the Shrink() function that shrinks the size of the array when the user delete element to 30% of the capacity of the array. The Shrink() function works exactly like the Grow() function expect, the “nv” has a capacity of half of the original array than double (Figure 11).



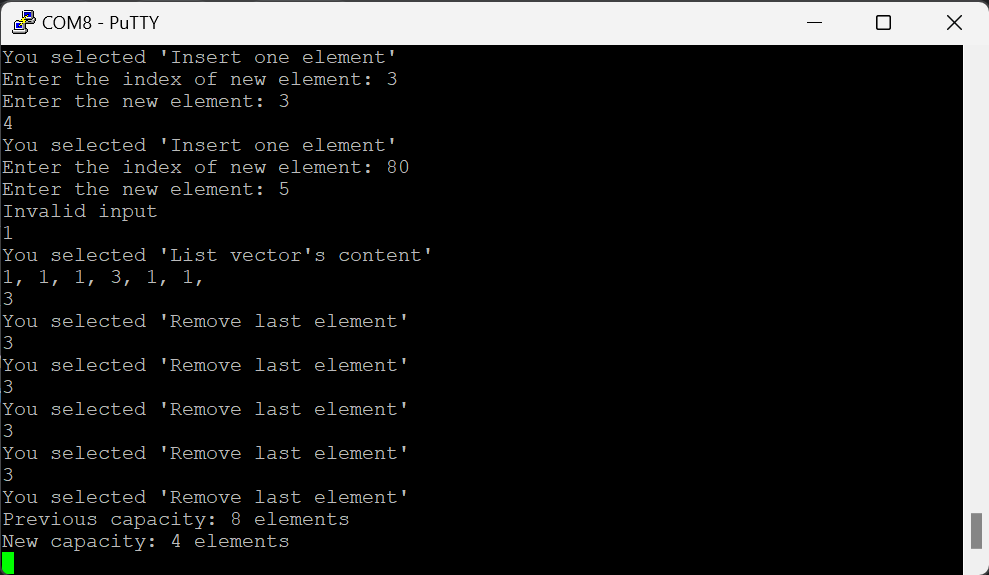
**Figure 11**: The Shrink() function works exactly like the Grow() function. Instead of doubling the capacity, the capacity is halfway.

We also modified the RemoveElement() function to incorporate the Shrink() function (Figure 12). The Shink() function is called every time the size is less or equal to the 30% of the capacity.



**Figure 12**: The updated RemoveElement() function triggers the Shrink() function when the size is too small.

We tested the new Shrink() function with the RemoveElement() function and it outputs the correct debug information (Figure 13).



**Figure 13**: The terminal output of the Shrink() function triggers correctly. The program reduces the size of the array when it is too small.

**Analysis**

In this lab we programmed a dynamic array (vector) in C. We created functions that allow you to list contents, add elements, remove elements, and manage memory. The array is initialized anf the memory allocated before manipulating elements. The functions that pertained to memory allocation were a function to double the size of the list when trying to append an element to a full list and a function that would shrink the array if below 30% capacity to save space. Concepts such as computational efficiency were considered, for example, the reason space is doubled every time the array is full is because to increase capacity by one every single rendition would be horribly inefficient since copying and relocating the values takes a relatively long time. Malloc was used to allocate memory for the array since it changes during runtime versus when compiled.

# Conclusion

This lab gave us a deeper understanding of how memory allocation works using vectors in C++ by writing a primitive library ourselves. We managed memory by programming functions to allocate more memory or to free up space when available. We were introduced to the concept of the stack and the heap, how variables initialized in the stack are temporary and to store their values past the function is to use ‘new’ or in the case of this lab ‘malloc’. Overall, the lab furthered our understanding of the hardware/software interaction and the process of compilation.