Dynamically Growing Arrays in C++
Embedded Design: Enabling Robotics
EECE2160

Michael BRODSKIY
Brodskiy.M@Northeastern.edu

March 23, 2023

Date Performed:	March 16, 2023
Partner:	Dylan POWERS
Instructor:	Professor SHAZLI

Abstract

This laboratory experiment served as an introduction to pointers and array manipulation in `C++`. By having us generate logic to manipulate a value pointing to an array, we became more familiarized with pointer and array concepts.

KEYWORDS: C++, array, pointer

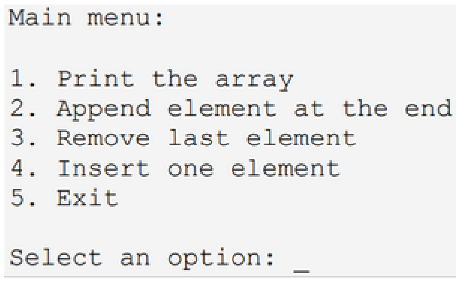
1 Equipment

Available equipment included:

- DE1-SoC board
- DE1-SoC Power Cable
- USB-A to USB-B Cable
- Computer
- MobaXTerm SSH Terminal
- USB-to-ethernet Adapter

2 Introduction

When creating a dynamically growing array, an initial region of memory is assigned to it. Then, as elements begin to be inserted into the array, the initial memory region is occupied, and, if necessary, expanded. In this lab, the goal was to serve as an introduction to implementing and modifying dynamically growing arrays into a program. To begin, a program was created to prompt a user to select an option from a menu of array modifications, shown below in Figure 1:



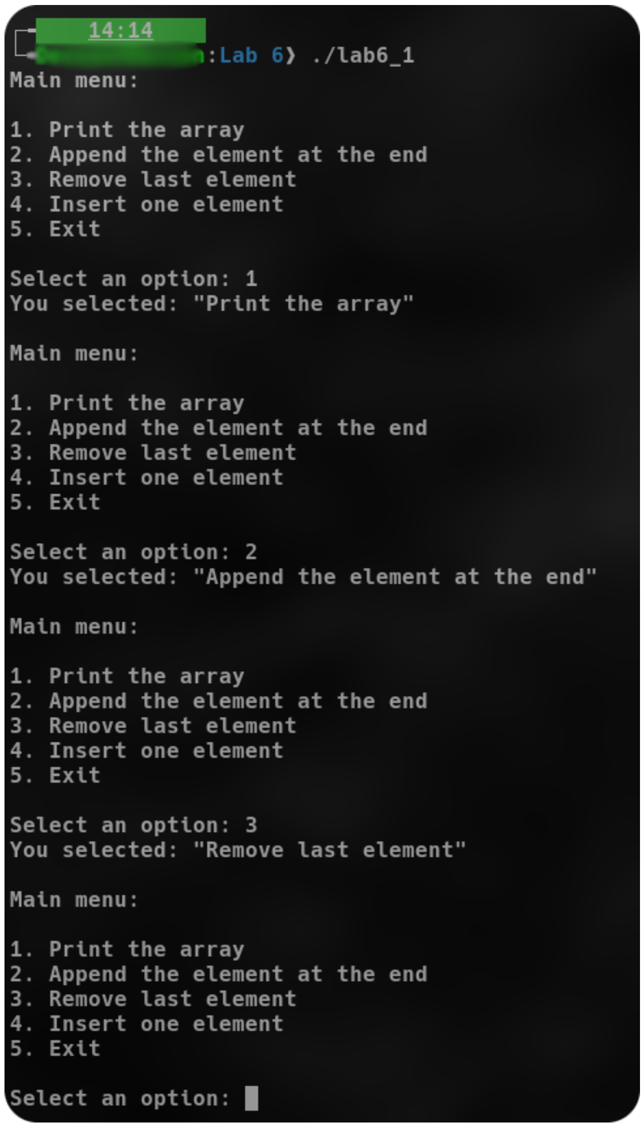
```
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: _
```

Figure 1: Menu of Array Modifications

3 Discussion & Analysis

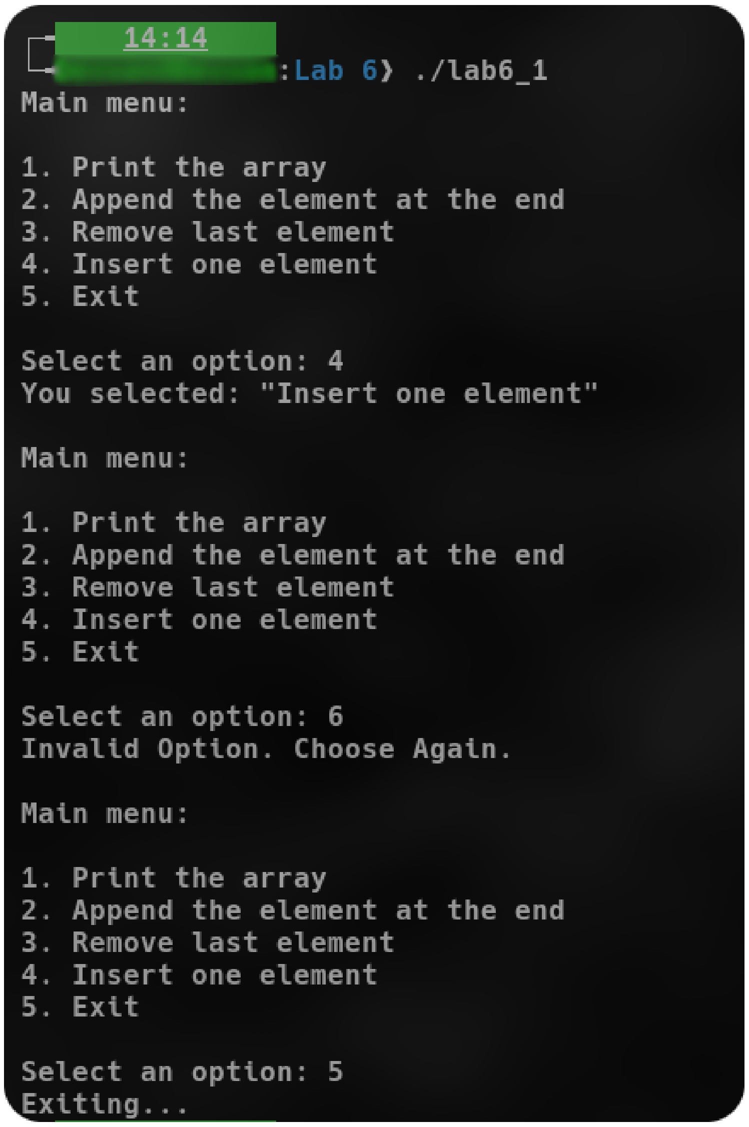
3.1 Assignment 1

The goal of Assignment 1 was to write a program that displays the menu shown in Figure 1, and waits for a user to enter a selection. At this point, if a valid selection is made, the menu repeats except for the selection of integer 5 where the program exits. However, If any user input was invalid, an error message displayed and the main menu repeated. Below is the output of an execution example where several different options were selected (Figures 2-3).



```
14:14
[redacted]:Lab 6❯ ./lab6_1
Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1
You selected: "Print the array"

Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
You selected: "Append the element at the end"

Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 3
You selected: "Remove last element"

Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 
```

Figure 2: Menu Selection Output



```
   14:14
[redacted]:Lab 6❯ ./lab6_1
Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 4
You selected: "Insert one element"

Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 6
Invalid Option. Choose Again.

Main menu:

1. Print the array
2. Append the element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 5
Exiting...
```

Figure 3: Menu Selection Output 2

3.2 Assignment 2

The goal of Assignment 2 was to write a function `Grow()` that grows the capacity of the vector. The designed function increased the vector's allocated storage while keeping the same set of elements in the vector. The new code for the `Grow()` function is shown below in Listing 1.

Listing 1: `Grow()` Function Code

```
int Grow() {

    double *nv = new double[2 * size]; // Allocate double
        the memory for new vector

    for (int i = 0; i < size; i++) { // Copy the values
        from v to nv

        nv[i] = v[i];

    }

    Finalize(); // Free memory consumed by vector v

    v = nv; // Set v to nv

    cout << endl << "Vector grown" << endl; // Print
        statements
    cout << "Previous capacity: " << size << " elements" <<
         endl;

    size = 2 * size; // Double the size
    cout << "New capacity: " << size << " elements" << endl
         << endl;

    return 0; // Return success

}
```

3.3 Assignment 3

The goal of Assignment 3 was to write an `AddElement()` function capable of adding an element at the end of the vector even if the vector was full. This required invoking `Grow()` when the current number of present elements was equal to the capacity of the vector. Once it was ensured that there was enough storage capacity for a new element, the new element was safely added at the end of the vector. Along with `AddElement()`, a `PrintVector()` function was written to print the current elements contained within the vector. The code for both functions are shown below in Listings 2 and 3.

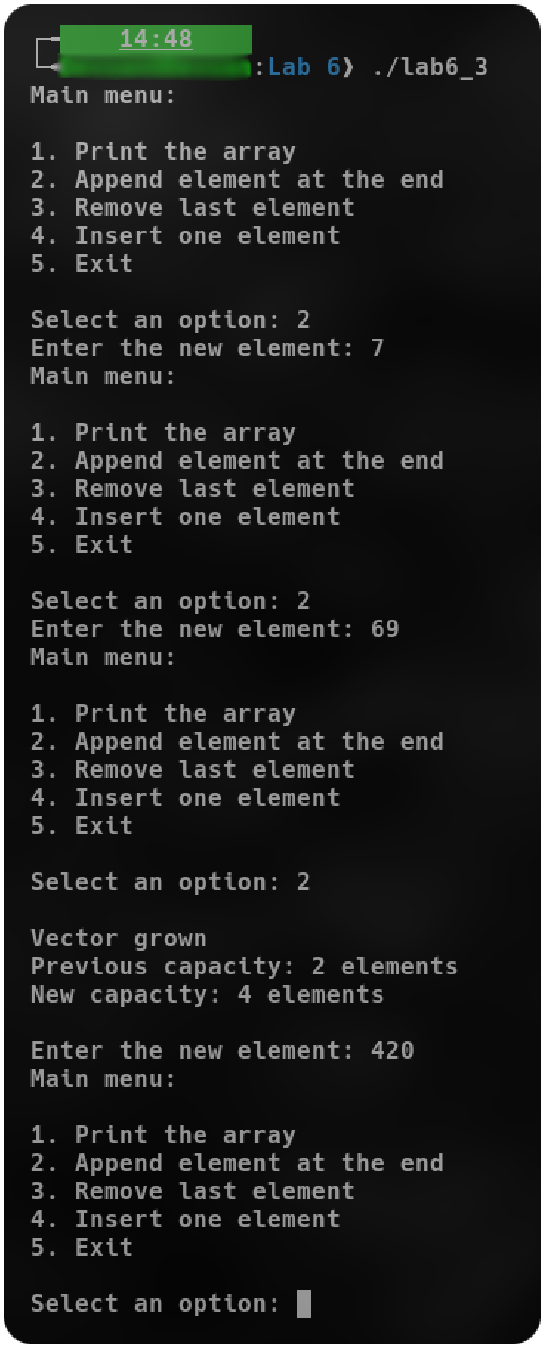
Listing 2: `AddElement()` Function Code

```
int AddElement() {

    if (count == size) {

        Grow(); // If vector at capacity, increase size

    }

    cout << "Enter the new element: ";
    cin >> v[count]; // Set input value to empty vector
        slot
    count++; // Increase count value

    return 0; // Return success

}
```

Listing 3: `PrintVector()` Function Code

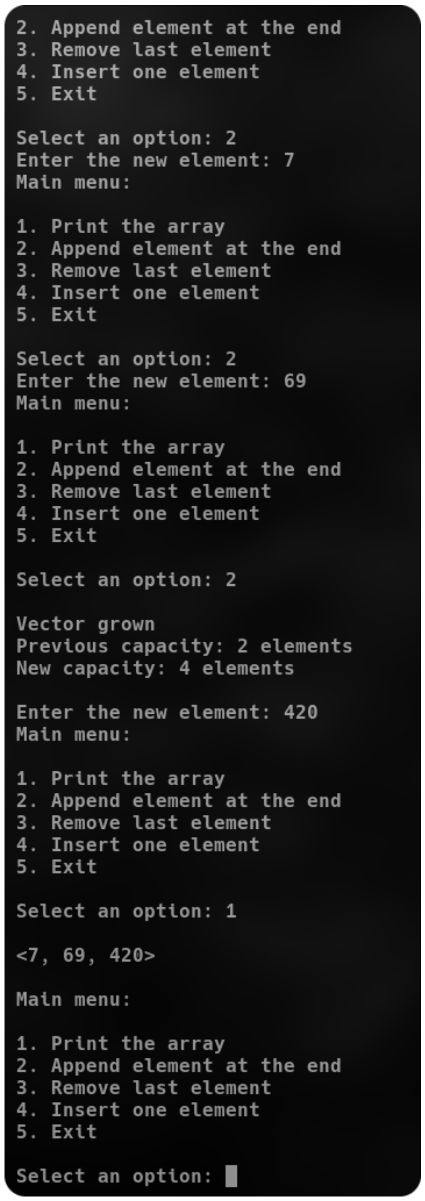
```
int PrintVector() {

    cout << endl << "<"; // Print left bracket

    for (int i = 0; i < count - 1; i++) {// Iterate through
         elements, printing each

        cout << v[i] << ", ";

    }

    if (count > 0) cout << v[count - 1]; // Print last
        element without comma
    cout << ">" << endl << endl; // Print right bracket

    return 0; // Return success

}
```

Additionally, the output of the program for an execution where several elements were added to a vector and the current content of the vector was printed is shown in Figures 4-5.



```
14:48
[redacted]:Lab 6⟩ ./lab6_3
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 7
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 69
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2

Vector grown
Previous capacity: 2 elements
New capacity: 4 elements

Enter the new element: 420
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 
```

Figure 4: Assignment 3 Execution



```
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 7
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 69
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2

Vector grown
Previous capacity: 2 elements
New capacity: 4 elements

Enter the new element: 420
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1

<7, 69, 420>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 
```

Figure 5: Assignment 3 Execution Part 2

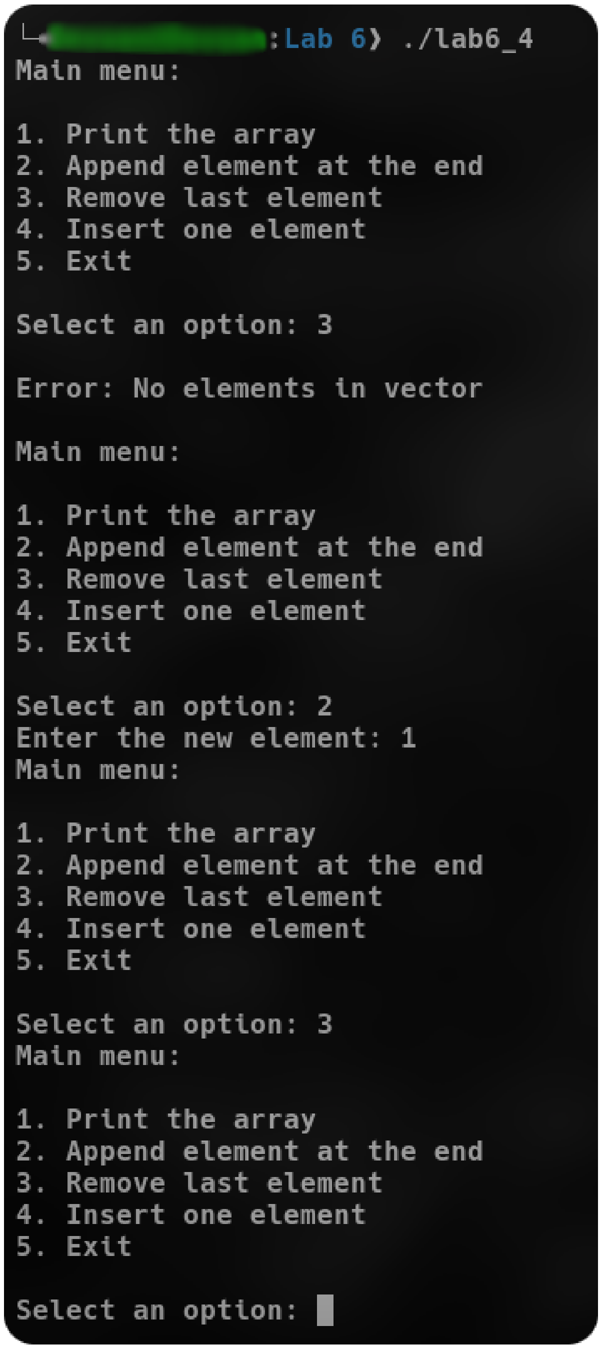
3.4 Assignment 4

The goal of Assignment 4 was to write a `RemoveElement()` function accessible through option 3 in the main menu that removed the last element contained in the vector. Additionally, the program displayed a proper error message indicating that there are no elements in the vector to remove when the vector was empty and the user selected option 3. The code for the `RemoveElement()` function is shown in Listing 4.

Listing 4: `RemoveElement()` Function Code

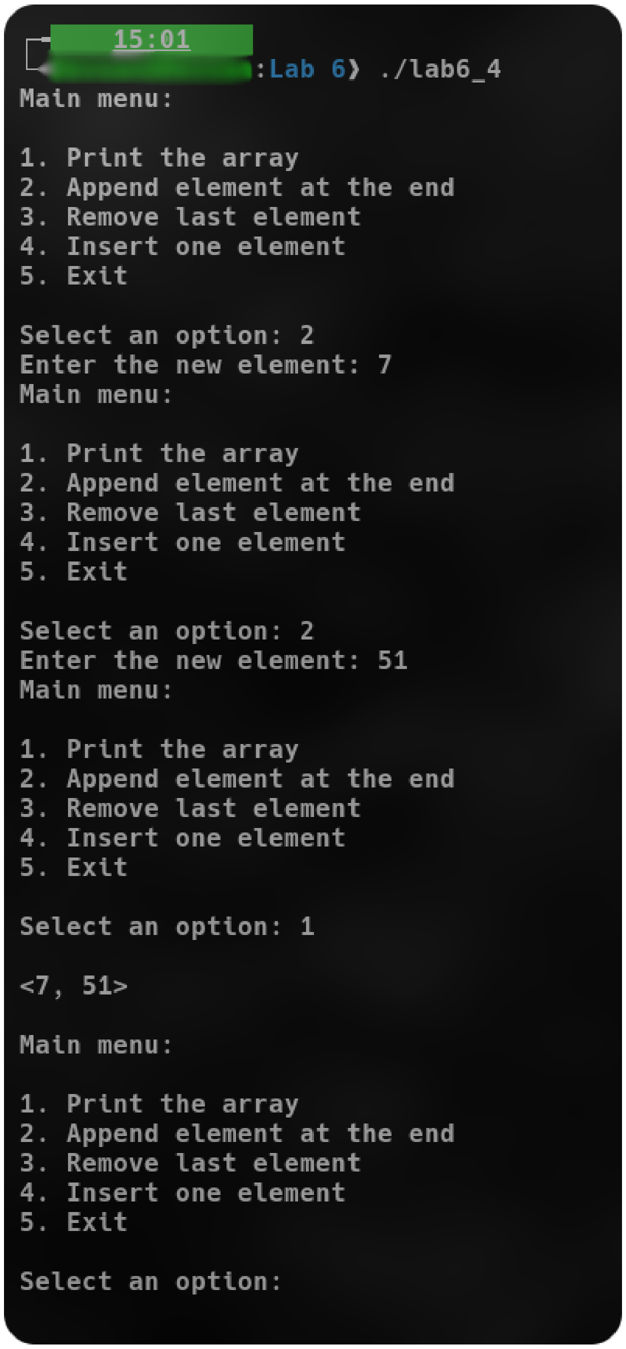
```
int RemoveElement() {

    if (count == 0) { // If no elements, print error message

        cout << endl << "Error: No elements in vector" << endl << endl;
        return 1; // Return error code

    }

    else {

        v[count - 1] = 0; // Free last element
        count = count - 1; // Subtract one from count
        cout << endl << "Successfully removed value" << endl << endl;

    }

    return 0; // Return Success

}
```

An output of the program removing the last element successfully and an output where the function is invoked on an empty vector are shown in Figures 6-8.



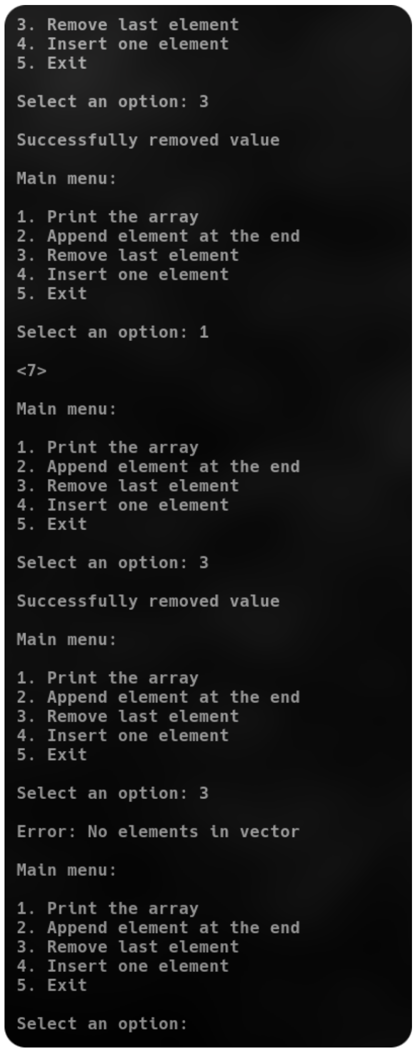
```
└─[redacted]:Lab 6❭ ./lab6_4
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 3

Error: No elements in vector

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 1
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 3
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 
```

Figure 6: Assignment 4 Execution



```
15:01
[redacted]:Lab 6❯ ./lab6_4
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 7
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 2
Enter the new element: 51
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1

<7, 51>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option:
```

Figure 7: Assignment 4 Execution Part 2



```
3. Remove last element
4. Insert one element
5. Exit

Select an option: 3

Successfully removed value

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1

<7>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 3

Successfully removed value

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 3

Error: No elements in vector

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option:
```

Figure 8: Assignment 4 Execution Part 3

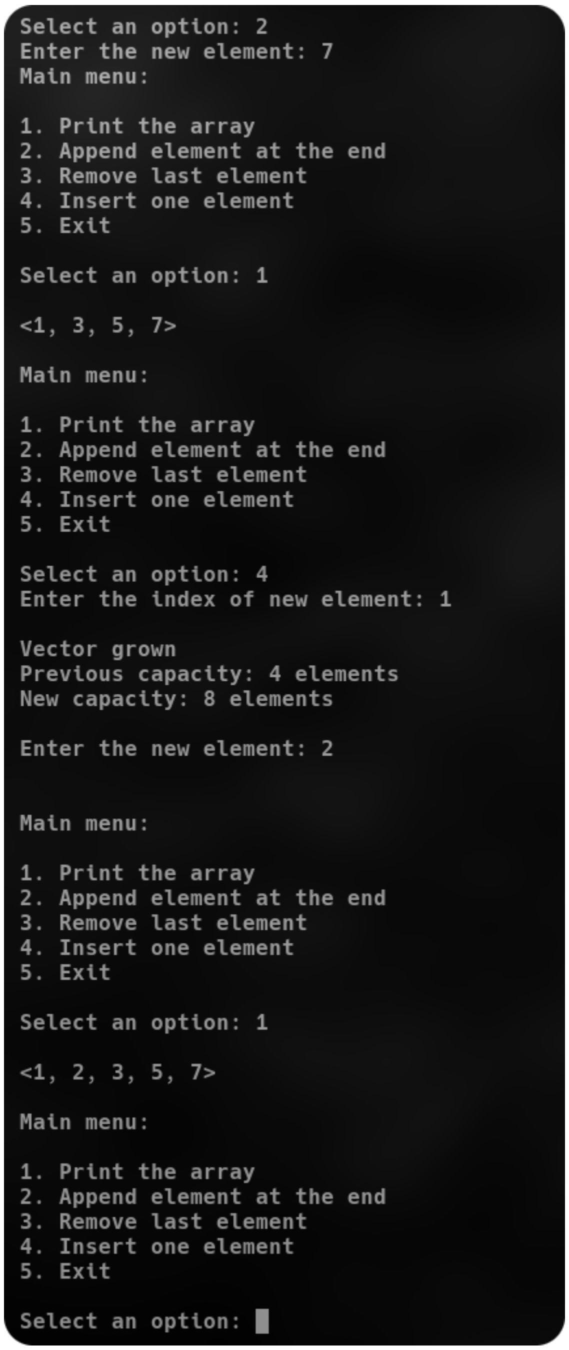
3.5 Assignment 5

The goal of Assignment 5 was to write an `InsertElement()` function and have it accessible to the user through option 4 in the menu. The function asks the user for an index and a value for the new element. The index is then checked for correct boundaries, and a proper error message is displayed if the entered value is invalid. The code for `InsertElement()` is shown below in Listing 5.

Listing 5: `InsertElement()` Function Code

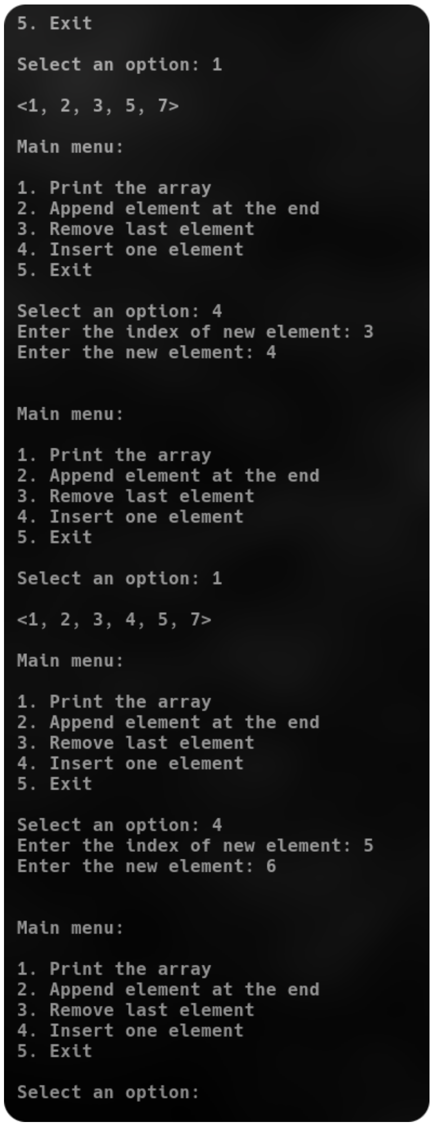
```
int InsertElement() {

    int index;
    double value;

    cout << "Enter the index of new element: ";
    cin >> index;

    while (index > count - 1|| index < 0) {

        cout << endl << "Error: Invalid index" << endl;
        cout << "Enter the index of new element: ";
        cin >> index;

    }

    if (count == size) {

        Grow();

    }

    cout << "Enter the new element: ";
    cin >> value;
    cout << endl << endl;

    int swap = v[index];
    int swap2 = 0;
    v[index] = value;

    for (int i = index; i < size; i++) {

        swap2 = v[i + 1];
        v[i + 1] = swap;
        swap = swap2;

    }

    count++;

    return 0; // Return success

}
```

Screenshots for testing the overall code are shown below in Figures 9-10.



```
Select an option: 2
Enter the new element: 7
Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1

<1, 3, 5, 7>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 4
Enter the index of new element: 1

Vector grown
Previous capacity: 4 elements
New capacity: 8 elements

Enter the new element: 2


Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1

<1, 2, 3, 5, 7>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 
```

Figure 9: Code Testing



```
5. Exit

Select an option: 1

<1, 2, 3, 5, 7>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 4
Enter the index of new element: 3
Enter the new element: 4


Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 1

<1, 2, 3, 4, 5, 7>

Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option: 4
Enter the index of new element: 5
Enter the new element: 6


Main menu:

1. Print the array
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit

Select an option:
```

Figure 10: Code Testing Part 2

4 Conclusion

Overall, this lab resulted in the creation of a menu-modified, dynamically-grown array. Through memory allocation and expansion, in tandem with element addition, insertion, and removal logic, the aforementioned dynamically-growing array was constructed. As such, this laboratory experiment demonstrated memory allocation concepts in `C++`.

5 Appendix

Listing 6: Complete Source Code

```
/*
 * =====================================================================================
 *
 *       Filename:  lab6_5.cpp
 *
 *    Description:  Introduces many functions to work with
    vectors
 *
 *        Version:  1.0
 *        Created:  03/16/2023
 *       Revision:  none
 *       Compiler:  GCC
 *
 *         Author:  Michael Brodskiy, Dylan Powers
 *
 * =====================================================================================
 */

#include <iostream>
#include <string> // Include string and iostream

using namespace std; // Use std as default namespace

double *v; // Declare global variables
int count, size;

int Initialize() {

    size = 2;
    count = 0;
    v = new double[size]; // Initialize the global
        variables to default values

    return 0; // Return success

}

int Finalize() {

    free(v); // Free memory consumed by vector v

    return 0; // Return success

```

```
}

int Grow() {

    double *nv = new double[2 * size]; // Allocate double
        the memory for new vector

    for (int i = 0; i < size; i++) { // Copy the values
        from v to nv

        nv[i] = v[i];

    }

    Finalize(); // Free memory consumed by vector v

    v = nv; // Set v to nv

    cout << endl << "Vector grown" << endl; // Print
        statements
    cout << "Previous capacity: " << size << " elements" <<
         endl;

    size = 2 * size; // Double the size
    cout << "New capacity: " << size << " elements" << endl
         << endl;

    return 0; // Return success

}

int PrintVector() {

    cout << endl << "<"; // Print left bracket

    for (int i = 0; i < count - 1; i++) {// Iterate through
         elements, printing each

        cout << v[i] << ", ";

    }

    if (count > 0) cout << v[count - 1]; // Print last
        element without comma
    cout << ">" << endl << endl; // Print right bracket

    return 0; // Return success

}

int AddElement() {

    if (count == size) {

        Grow(); // If vector at capacity, increase size

    }

    cout << "Enter the new element: ";
    cin >> v[count]; // Set input value to empty vector
        slot
    count++; // Increase count value

    return 0; // Return success

}

int RemoveElement() {

    if (count == 0) { // If no elements, print error
        message

        cout << endl << "Error: No elements in vector" <<
            endl << endl;
        return 1; // Return error code

    }

    else {

        v[count - 1] = 0; // Free last element
        count = count - 1; // Subtract one from count
        cout << endl << "Successfully removed value" <<
            endl << endl;

    }

    return 0; // Return Success

}

int InsertElement() {

    int index;
    double value;

    cout << "Enter the index of new element: ";
    cin >> index;

    while (index > count - 1|| index < 0) {

        cout << endl << "Error: Invalid index" << endl;
        cout << "Enter the index of new element: ";
        cin >> index;

    }

    if (count == size) {

        Grow();

    }

    cout << "Enter the new element: ";
    cin >> value;
    cout << endl << endl;

    int swap = v[index];
    int swap2 = 0;
    v[index] = value;

    for (int i = index; i < size; i++) {

        swap2 = v[i + 1];
        v[i + 1] = swap;
        swap = swap2;

    }

    count++;

    return 0; // Return success

}

int main() {

    Initialize(); // Initialize values

    string options[] = {"Print the array", "Append element at the end", "Remove last element", "Insert one element", "Exit"}; // Store options in array for ease of access

    int input = 0; // Set input to 0 by default

    while (input != 5) { // Repeat main menu until exit key (5) is entered

        cout << "Main menu:" << endl << endl; // Print main menu header

        for (int i = 1; i <= 5; i++) { // Loop through array, printing it

            cout << i << ". " << options[i - 1] << endl;

        }

        cout << endl << "Select an option: "; // Allow user to select option
        cin >> input; // Set input to option selected by user

        switch (input) { // If cases 1-4, drop down to print statement; If exit case, return success; If invalid value, print and choose again

            case 1:

                PrintVector(); // Call PrintVector
                break;

            case 2:

                AddElement(); // Call AddElement
                break;

            case 3:

                RemoveElement(); // Call RemoveElement
                break;

            case 4:

                InsertElement(); // Call InsertElement
                break;

            case 5:

                cout << "Exiting..." << endl;
                return 0;
                break;

            default:

                cout << "Invalid Option. Choose Again." <<
                    endl << endl;

      }

 }

}
```