

# Lab Assignment 1

## Dynamic Growing Arrays in C++

Saul Reyna, Yuheng Dong

reyna.s@husky.neu.edu

dong.yuh@husky.neu.edu

Submit date: 1/28/2020

Due Date: 1/29/2020

### **Abstract**

The lab consisted of implementation of dynamic array in C++, also known as vectors. The program made for this lab was an interactive menu that allowed the user to perform different task on a vectors, which include the initialization, the increase in allocated memory for the vector, adding/remove elements to either the end of the list or at any given index, shrinking the size of the vector by freeing up unused memory and printing the current elements of the vector. This allowed an array to not have a set number of elements, allowing for the user to just keep adding elements without worry of the adding an element that is outside of the bounds of the static array.

## Introduction

Before this lab, any array that were used in any of the programs were static, meaning that the size of the array was set within the initialization of the array. This meant that the array wouldn't have much flexibility later on, when any changes to the program would inevitably come. With this lab, dynamic arrays, or vectors, were introduced and implements in C++. With this vector, it gave the user the ability to keep adding elements to an array without having to worry about adding an element to an index that was not in the initial array. With the implementation of vectors, previous knowledge about pointers to elements and how to reference them is necessary, since the pointers are what allow us to keep referencing and allocating memory to the initialized array.

## Lab Discussion

Equipment used in this lab include

- Zedboard (ZYNQ SoC)
  - o Dual ARM Cortex – A9 MPCore with CoreSight
  - o Zynq-7000 AP SoC XC7Z020-1CLG484
- Xilinx OS
- CoE lab desktop

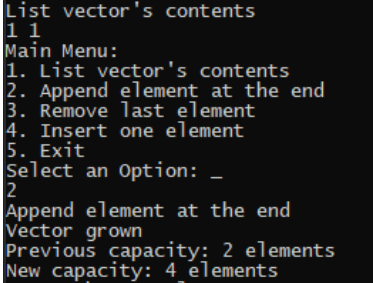
## Results and Analysis

### Assignment 1.1 – Growing the vector

```
void Grow(){
    int new_size = size * 2;
    double *new_array = new double[new_size];

    for(int i = 0; i < size; i++){
        new_array[i] = array[i];
    }

    Finalize();
    array = new_array;
    cout << "Vector grown" << endl;
    cout << "Previous capacity: " << size << " elements" << endl;
    cout << "New capacity: " << new_size << " elements" << endl;
    size = new_size;
}
```



```
List vector's contents
1 1
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: _
2
Append element at the end
Vector grown
Previous capacity: 2 elements
New capacity: 4 elements
```

Figure 1 - Output when growing a vector

One of the cornerstones of vectors are their ability to dynamically grow in size when needed. The function `Grow()` is what allows that by setting a new size equal to double of the previous size. Once the new size is calculated, a pointer to a new array is made, which has the new size. Once that array is initialized, all the elements of the old array are copied into the new one, leaving the rest with a bunch of zeroes. When all the elements are copied over, the pointer for the old array is changed to the new one, so that it can still be referenced without much trouble. Since we don't want to take up any unnecessary space, the old array is deleted.

## Assignment 1.2 – Adding an element to the end

```
// add an element to the end of the vector
```

```
void AddElement(){
    if(size == count){
        Grow();
    }
    double new_element;
    cout << "Enter the new element: _" << endl;
    cin >> new_element;
    array[count] = new_element;
    count++;
}
```

```
// Print the vector
```

```
void PrintVector(){
    for(int i = 0; i < size; i++){
        cout << array[i] << " ";
    }
    cout << endl;
}
```

```
List vector's contents
5 4
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: _
2
Append element at the end
Vector grown
Previous capacity: 2 elements
New capacity: 4 elements
Enter the new element: _
3
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: _
1
List vector's contents
5 4 3 0
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: _
```

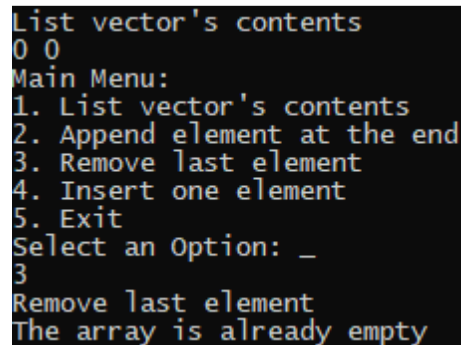
```
Main Menu:
1. List vector's contents
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit
Select an Option: _
1
List vector's contents
0 0
Main Menu:
1. List vector's contents
2. Append element at the end
3. Remove last element
4. Insert one element
5. Exit
Select an Option: _
2
Append element at the end
Enter the new element: _
5
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: _
1
List vector's contents
5 0
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: _
```

The `AddElement()` function is one of the main reasons for using a dynamic array. If the user wishes to add an element to an array that is not completely filled, it will just add the elements to the next empty index in the array. However, once `count` is equal to the `size` of the array, then the function knows to increase the size of the vector, since `count` is the total number of elements in the array, meaning that it will try to add the element to index `size`, an index that is out of bounds.

The `PrintVector()` goes to every single element in vector, assigned or not, and prints it. Since `size` is a global variable (something that is usually not recommended), it allows the program to reference it without the function needing an argument to know how large the array is.

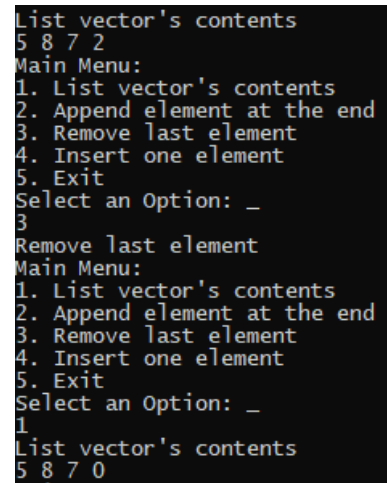
### Assignment 1.3 – Removing an element from the end

```
void RemoveElement(){
    if(count>0){
        array[count-1] = 0;
        count--;
    }
    else cout << "The array is already empty" << endl;
}
```



```
List vector's contents
0 0
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: _
3
Remove last element
The array is already empty
```

Figure 2. Removing an element from an empty vector



```
List vector's contents
5 8 7 2
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: _
3
Remove last element
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: _
1
List vector's contents
5 8 7 0
```

Figure 1. Removing an element from the end of the vector

The `RemoveElement()` removes the last element of the vector by accessing the array at the index of the last assigned index, which is one less than the total number of elements, and setting it to zero. If the array is already filled with zeroes (meaning the vector is completely empty and count is zero), an error message is printed, telling the user that the vector is already empty and any more elements cannot be removed.

## Conclusion

Explain what the results indicate from a larger or system-level perspective. Reconcile experimental results and account for any differences you observed. If appropriate, explain what work might be done in the future.

## References

If you use any additional texts, papers, websites, etc. and refer to them in the report, then you must include a reference. Note that copying text from other sources is typically considered plagiarism. If you verbatim copy text you will need to put “the copied text” in double quotes and cite the source. In case you are using main ideas from a different paper you need to cite. For example, most of this lab report guide is based by work of Michael Benjamin [1]. Also if you find useful sources, please tell me about them.

[1] Michael Benjamin, “*Lab Report Guide*”, Northeastern University, January 18 2006.

Saul Reyna, Yuheng Dong EECE2160	Embedded Design: Enabling Robotics Lab Assignment 1
-------------------------------------	--

### Appendix

Use an appendix to present additional information such source code snippets, code organization or test run outputs.