Lab Assignment 1
Dynamically Growing Arrays in C++

Levi Kaplan
kaplan.l@northeastern.edu

Submit Date: 9/21/20
Due Date: 9/21/20

Abstract
In this lab, we were introduced to working with vectors in C++. We created a basic menu and added functionality for printing, appending values, removing values, adding values at an index, and quitting the program. We worked with memory initialization and deletion and growing and shrinking the size of vectors.

Introduction

In this lab, we expanded our understanding of C++ by exploring the functionality of vectors in depth. We implemented growing and shrinking vector memory size, appending elements, printing the vector, removing elements, inserting elements at a particular index, and exiting the program. Our goal was to understand the workings and requirements when working with vector data types in C++. This lab assumed working knowledge on the C++ programming language, although certain details such as how to work with vector data types were up to us to figure out as a part of the learning objectives of this lab.

Software and Hardware Used

Hardware

- DE1-SoC board
- Ethernet cable

Software

- Atom text editor
- Online C++ compiler (https://www.onlinegdb.com/online_c++_compiler)
- ssh and scp

Lab Steps

Prelab

1. We implemented the switch statement for the menu
2. We tested the functionality of the switch statement

Lab 1.1

1. We duplicated the content of the Prelab
2. We first read through the details on growing a vector in C++
3. We next implemented the Grow() function
4. We tested the Grow() function by inputting test data

Lab 1.2

1. We duplicated the content of Lab 1.1
2. We added the AddElement() method stub to our program

3. We added the AddElement() method call in the menu switch statement
4. We wrote a preliminary implementation for AddElement() and PrintVector()
5. We tested this implementation and found numerous issues
6. We changed the method to have different logic if the size was 0
7. We implemented the PrintVector method
8. We tested the method
9. We changed the logic to better display smaller arrays
10. We debugged these two methods

Lab 1.3

1. We duplicated the content of Lab 1.2
2. We added the RemoveElement() method stub to our program
3. We added the RemoveElement() method call in the menu switch statement
4. We wrote a preliminary implementation for RemoveElement()
5. We tested the RemoveElement() implementation
6. We bug fixed the RemoveElement() method

Lab 1.4

1. We duplicated the content of Lab 1.3
2. We added the InsertElement() method stub to our program
3. We added the InsertElement method call in the menu switch statement
4. We wrote a preliminary implementation for RemoveElement()
5. We tested the functionality of this program
6. We debugged the implementation

Lab 1.5

1. We duplicated the content of Lab 1.4
2. We added the Shrink() method stub to our program
3. We added the Shrink() method call in the RemoveElement() method
4. We wrote a preliminary implementation for RemoveElement()
5. We tested the functionality of this program

Lab Discussion

Prelab

1. We implemented the switch statement for the menu
    a. We used a recursive method to continually provide the menu after an operation was concluded
    b. We forgot to break the cases after recurring which introduced a bug when exiting (see Analysis)
2. We tested the functionality of the switch statement
    a. Entering 5 (quitting) worked properly as the bug was only introduced when the other options were functional, so it slipped past the Prelab portion

Lab 1.1

1. We duplicated the content of the Prelab
2. We first read through the details on growing a vector in C++
    a. We analyzed how the logic of duplication and altering the pointer works
3. We next implemented the Grow() function
    a. We used a for loop to copy the values over to a new array that is double the size of the original
    b. We deleted the original array and set its pointer as the new array
    c. We printed debug data
4. We tested the Grow() function by inputting test data
    a. We initialized a vector and made sure the initialization worked
    b. We didn't have the ability to add elements yet and couldn't properly test, but we had one of the cases call the function and observed the debug data

Lab 1.2

1. We duplicated the content of Lab 1.1
2. We added the AddElement() method stub to our program
3. We added the AddElement() method call in the menu switch statement
4. We wrote a preliminary implementation for AddElement() and PrintVector()
    a. Our preliminary implementation did not properly handle when the size of the array was 0 (as we initialized it to be)

   b. This implementation did recursively call the function after Grow() was called, which made it a very clean method
   c. We used 'size' in places where we should have had 'count', which caused many unexpected 0's to be found when we printed the array
5. We tested this implementation and found numerous issues
   a. Both AddElement() and PrintVector() did not behave as expected
   b. AddElement() wouldn't properly add, and PrintVector() was ugly and didn't behave as expected with limited values
6. We changed the method to have different logic if the size was 0
   a. We added logic that increased the size properly if the size was 0, and would otherwise call Grow() if there wasn't any size left
7. We tested the method
   a. We ran numerous queries where we would repeatedly add and then print to make sure all the behavior was expected
8. We changed the logic to better display smaller arrays
   a. The PrintVector() method would add an unexpected 0 in the array when it had no values, so the logic for when to display v[0], as well as the initial size of the method, was tweaked
   b. Again, there were places where 'size' was used instead of 'count', now in the PrintVector() method, that had to be changed
9. We debugged these two methods
   a. We changed the logic and display as we found new edge cases that did not conform to our expectations

Lab 1.3

1. We duplicated the content of Lab 1.2
2. We added the RemoveElement() method stub to our program
3. We added the RemoveElement() method call in the menu switch statement
4. We wrote a preliminary implementation for RemoveElement()
   a. We wrote similar logic to Grow() that creates a new empty vector pointer and vector and then copies all but the last element of the original vector into the new vector, and adjusts the original pointer to point to the new vector
5. We tested the RemoveElement() implementation

   a. We tested that RemoveElement() works in conjunction with AddElement() and PrintVector()
   b. We tested that the method works properly when there are no elements in the array
6. We bug fixed the RemoveElement() method
   a. Testing RemoveElement() with AddElement() revealed some bugs in AddElement() that we fixed

Lab 1.4

1. We duplicated the content of Lab 1.3
2. We added the InsertElement() method stub to our program
3. We added the InsertElement method call in the menu switch statement
4. We wrote a preliminary implementation for RemoveElement()
   a. We used a technique of copying the array to the right from the last element until the index we are inserting the element in
5. We tested the functionality of this program
6. We debugged the implementation
   a. We initially had it indexing from 1 to the count plus one but realized that the assignment specified the indexing
   b. We changed the indexing but had some difficulties getting it to work again
   c. We looked over the logic again and was able to get it working

Lab 1.5

1. We duplicated the content of Lab 1.4
2. We added the Shrink() method stub to our program
3. We added the Shrink() method call in the RemoveElement() method
   a. We included the logic for when to shrink the array
4. We wrote a preliminary implementation for RemoveElement()
   a. We used a very similar method to that of Grow()
   b. We were able to use the Grow() logic and simply reduce instead of increase the size
5. We tested the functionality of this program

Results

Lab 1.1

The screenshot below demonstrates that the vector is grown due to two elements being preemptively added to the array. The capacity is grown and the debug statements indicating the previous and new capacity are shown to the user.

```
root@de1soclinux:~/Labs/Lab1/Lab1# g++ -o lab11 main.cpp
root@de1soclinux:~/Labs/Lab1/Lab1# ./lab11
Vector grown
Previous capacity: 2 elements
New capacity: 4 elements
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: _
```

Lab 1.2

The screenshots below show the functionality of adding items to the vector and printing the vector. If double values are used, the double will be inserted.

```
root@de1soclinux:~/Labs/Lab1/Lab1# ./lab1-2
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
Print the array
[]
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
Append element at the end
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: _2
Append element at the end
Enter the new element: _2
Vector grown
Previous capacity: 1 elements
New capacity: 2 elements
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
Append element at the end
Enter the new element: _3
Vector grown
Previous capacity: 2 elements
New capacity: 4 elements
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
Print the array
[1, 2, 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: _
```

```
Select an option: _2
Append element at the end
Enter the new element: _1.0
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
Print the array
[1, 2, 3, 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: _2
Append element at the end
Enter the new element: _1.5
Vector grown
Previous capacity: 4 elements
New capacity: 8 elements
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
Print the array
[1, 2, 3, 1, 1.5]
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: _
```

Lab 1.3

Below is an example of running ‘3’ in the menu, and deleting the final item in the array. As you can see, when the final item is deleted, running delete again throws an error and says there aren’t more elements left to remove.

```
[1, 2.3, 4.5]
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
Remove last element
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
Print the array
[1, 2.3]
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: _3
Remove last element
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
Print the array
[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
Remove last element
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
Print the array
[]
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
Remove last element
Could not perform: there are no elements left to remove
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: _
```

Lab 1.4

The two screenshots side by side show three different examples of inserting values into the array. The first example appends a 1 to the beginning of the array. The second example appends a 3 between 2 and 4. The third example appends a 5 at the end of the array.

```
root@de1soclinux:~/Labs/Lab1/Lab1# g++ -o lab1-4 lab1-4.cpp
root@de1soclinux:~/Labs/Lab1/Lab1# ./lab1-4
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
Append element at the end
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: _2
Append element at the end
Enter the new element: _4
Vector grown
Previous capacity: 1 elements
New capacity: 2 elements
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
Insert one element
Enter the index of new element: _0

Enter the new element: _1
Vector grown
Previous capacity: 2 elements
New capacity: 4 elements
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
Print the array
[1, 2, 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: _4
Insert one element
Enter the index of new element: _2

Enter the new element: _3
Vector grown
Previous capacity: 4 elements
New capacity: 8 elements
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
Print the array
[1, 2, 3, 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: _4
Insert one element
Enter the index of new element: _4

Enter the new element: _5
Vector grown
Previous capacity: 8 elements
New capacity: 16 elements
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
Print the array
[1, 2, 3, 4, 5]
Main menu:
```

This final example demonstrates the errors that are shown when the user tries to pick a negative index or an index outside of the range. In both cases, the console instructs the user on how to pick an appropriate 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: _4
Insert one element
Enter the index of new element: _-1

Enter the new element: _1
Index out of range. Please pick an index between 0 and the vector's count
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
Insert one element
Enter the index of new element: _6

Enter the new element: _6
Index out of range. Please pick an index between 0 and the vector's count
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: _
```

Lab 1.5

The screenshots below demonstrate the Shrink() functionality. As elements are removed from the large array at the start of the screenshot to the left, the Shrink() function is triggered and the vector gets shrunk, as is clear from the debug statements. As the vector gets smaller and smaller, so too does the memory the vector takes up.

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
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
Remove last element
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
Remove last element
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
Remove last element
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
Remove last element
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
Remove last element
Shrunk the vector
Previous capacity: 16
New capacity: 8
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, 2, 3, 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
Remove last element
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
Remove last element
Shrunk the vector
Previous capacity: 8
New capacity: 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
Remove last element
Shrunk the vector
Previous capacity: 4
New capacity: 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: _3
Remove last element
Shrunk the vector
Previous capacity: 2
New capacity: 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: _
```

Analysis

Prelab

The prelab warmed us up to the assignment and gave us experience with switch statements and testing them through the console. We inadvertently introduced a bug when trying to quit the program that we didn't catch until working on 1.5, wherein we forgot to break the statements and this introduced a 'double free or corruption (fasttop)' bug. The solution to this will be discussed in Analysis section 1.5.

Lab 1.1

Lab 1.1 required us to implement logic regarding creating a new vector, copying values over, and adjusting the pointer values for the original vector. We learned a lot about the expectations and requirements for vectors in C++, and how vectors are able to grow.

Lab 1.2

We had the most trouble with Lab 1.2, but we didn't realize the code was broken until after doing part of Lab 1.3. We implemented a solution but got some weird errors we were confused by, namely that there were 0's in unexpected places, but decided to move on. It was while testing the removal functionality that we realized a large part of the code was broken and had to be refactored. This iterative design and creating tools to better test our code was very successful and we were soon able to move past it. Some uncertainties were cleared up in Lab 1.3. We were very happy with the elegance of our solution to AddElement(), making use of a recursive call within the body after the vector had been expanded that made the code very clean.

Lab 1.3

Similarly to Lab 1.2, we found difficulty with the way C++ represented data and some ignorance on our part on how it works. We were getting large values like 4.88e419112 when removing values sometimes that confused us, and only after talking to the TA were we able to figure out that it had to do with the way we were accessing elements that hadn't been set yet. This was due to a confusion between 'size' and 'count', that once cleared up, had us no longer accessing elements of the vector that were not set yet.

Lab 1.4

This section of the lab had us initially making an incorrect implementation, 1-indexing the array instead of 0-indexing it. In trying to fix the mistake, we got

confused and had to walk through some examples to better understand why the code was no longer working and clear up the mistake. We were able to figure out the logic of going backward through the array and copying it to the next element, and then adding the new element at the specified index.

Lab 1.5

The final part of the lab was very similar to the code for Grow(), and we were able to implement the same logic. The logic was already tested and worked well for Grow(), so after testing Shrink() the first time, it immediately worked. We added logic in RemoveElement() to handle when the vector had decreased in size enough to have Shrink() be called on it.

We discovered a bug toward the end of development that once some data was added to the vector, when trying to quit the program it would try to run the append method and then throw an error. The reason was because we forgot to break the cases after the recursive method was called running the menu again. This caused the case to fall through when trying to quit the program, leading to this strange and previously unexplainable behavior. Adding break statements at the end of each case fixed the bug.

Conclusion

The discoveries made in this lab facilitated a deeper and more comprehensive understanding of many of the core features and language traits of C++. We were able to work through many unexpected bugs caused by language misunderstandings and in the process gain a deeper and more nuanced understanding of the underpinnings of C++. Additionally, we learned that C++ requires a more careful and diligent approach to programming due to the nature of memory management and vectors. We made great strides in complex program design, array adding, indexing, and copying, and memory management.

Appendix

Lab 1.1

```
#include <iostream>
#include <string>

using namespace std;

// vector values
double *v;
int count;
int size;
// string representing menu options
string menuString = "Main menu:\n1. Print the array\n2. Append element at the end\n3. Remove last
element\n4. Insert one element\n5. Exit \nSelect an option: _";

// Initialize method stub
void Initialize();
// Finalize method stub
void Finalize();
// runMenu method stub
int runMenu();
// selectStatement method stub
int selectStatement(char selection);
// Grow method stub
void Grow();

/*
 * Initializes the vector values and runs the menu.
 * @return : int representing the user's desire to quit the program
 */
int main()
{
  Initialize();
  Grow();
  return runMenu();
}

/*
 * Runs the menu, displaying menu options and getting user input.
 * @return : int if the program is exited, representing the user wanting to quit the program
 */
int runMenu()
{
```

```
  char selection;
  cout << menuString;
  cin >> selection;
  return selectStatement(selection);
}

/*
 * Represents logic for performing the desired action as specified by the user.
 * @param : selection the character selected by the user to be used for the switch
 * @return : 0 if the program is exited, otherwise recurs on runMenu()
 */
int selectStatement(char selection)
{
  switch (selection) {
    // print the array
    case '1':
      cout << "Print the array" << endl;
      runMenu();
    // append element at end
    case '2':
      cout << "Append element at the end" << endl;
      runMenu();
    // remove last element
    case '3':
      cout << "Remove last element" << endl;
      runMenu();
    // insert one element
    case '4':
      cout << "Insert one element" << endl;
      runMenu();
    // exit the program
    case '5':
      Finalize();
      return 0;
      break;
    // if invalid, display an error and run the menu again
    default:
      cout << "Error: please enter a valid option" << endl;
      runMenu();
  }
}

/*
```

```
 * Initializes vector values
 */
void Initialize()
{
  size = 2;
  count = 2;
  v = new double[size];
  // test values for testing purposes
  v[0] = 1;
  v[1] = 2;
}

/*
 * Deletes vector to free up memory space
 */
void Finalize()
{
  delete[] v;
}

/*
 * Doubles the size in memory of the array v .
 */
void Grow()
{
  // initializes pointer to nv
  double *nv;
  // sets nvSize equal to a doubling of the current size
  int nvSize = size * 2;
  // initializes nv
  nv = new double[nvSize];
  // for each element in v, set that same index in nv equal to its value
  for(int i = 0; i < count; i++)
  {
    double val = v[i];
    nv[i] = val;
  }
  // delete v
  delete[] v;
  // set to now point to nv
  v = nv;

  // print the details of the Grow() call
```

```
  cout << "Vector grown" << endl;
  cout << "Previous capacity: " << size << " elements" << endl;
  cout << "New capacity: " << nvSize << " elements" << endl;
  // set the new size equal to nvSize
  size = nvSize;
}
```

Lab 1.2

```
#include <iostream>
#include <string>

using namespace std;

// vector values
double *v;
int count;
int size;
// string representing menu options
string menuString = "Main menu:\n1. Print the array\n2. Append element at the end\n3. Remove last
element\n4. Insert one element\n5. Exit \nSelect an option: _";

// Initialize method stub
void Initialize();
// Finalize method stub
void Finalize();
// runMenu method stub
int RunMenu();
// selectStatement method stub
int SelectStatement(char selection);
// Grow method stub
void Grow();
// AddElement method stub
void AddElement(double element);
// PrintVector method stub
void PrintVector();

/*
 * Initializes the vector values and runs the menu.
 * @return : int representing the user's desire to quit the program
 */
int main()
{
```

```
  Initialize();
  return RunMenu();
}

/*
 * Runs the menu, displaying menu options and getting user input.
 * @return : int if the program is exited, representing the user wanting to quit the program
 */
int RunMenu()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return SelectStatement(selection);
}

/*
 * Represents logic for performing the desired action as specified by the user.
 * @param : selection the character selected by the user to be used for the switch
 * @return : 0 if the program is exited, otherwise recurs on runMenu()
 */
int SelectStatement(char selection)
{
  double newElem;
  switch (selection) {
    // print the array
    case '1':
      cout << "Print the array" << endl;
      PrintVector();
      RunMenu();
    // append element at end
    case '2':
      cout << "Append element at the end" << endl;
      cout << "Enter the new element: _";
      cin >> newElem;
      AddElement(newElem);
      RunMenu();
    // remove last element
    case '3':
      cout << "Remove last element" << endl;
      RunMenu();
    // insert one element
    case '4':
```

```
      cout << "Insert one element" << endl;
      RunMenu();
    // exit the program
    case '5':
      Finalize();
      return 0;
      break;
    // if invalid, display an error and run the menu again
    default:
      cout << "Error: please enter a valid option" << endl;
      RunMenu();
  }
}

/*
 * Initializes vector values
 */
void Initialize()
{
  size = 0;
  count = 0;
  v = new double[size];
}

/*
 * Deletes vector to free up memory space
 */
void Finalize()
{
  delete[] v;
}

/*
 * Doubles the size in memory of the array v .
 */
void Grow()
{
  // initializes pointer to nv
  double *nv;
  // sets nvSize equal to a doubling of the current size
  int nvSize = size * 2;
  // initializes nv
  nv = new double[nvSize];
```

```
 // for each element in v, set that same index in nv equal to its value
 for(int i = 0; i < count; i++)
 {
  double val = v[i];
  nv[i] = val;
 }
 // delete v
 delete[] v;
 // set to now point to nv
 v = nv;

 // print the details of the Grow() call
 cout << "Vector grown" << endl;
 cout << "Previous capacity: " << size << " elements" << endl;
 cout << "New capacity: " << nvSize << " elements" << endl;

 // set the new size equal to nvSize
 size = nvSize;
}

/*
 * adds the passed-in element to the end of the vector.
 * @param double : the element to add to the end of the vector
 */
void AddElement(double element)
{
 // if the size is 0, we want to add the element at the first slot and increment count and size
 if(size == 0)
 {
  v[0] = element;
  count = count + 1;
  size = size + 1;
 }
 // otherwise, if size is greater than count, we don't need to Grow the vector, so just add it
 else if(size > count)
 {
  v[count] = element;
  count = count + 1;
 }
 // size is equal to count -- the vector needs to be grown  before more can be added
 else
 {
  // grow the vector
```

```
   Grow();
   // recursively call AddElement now that the vector has grown
   AddElement(element);
  }
}

/*
 * Prints the vector to the console.
 */
void PrintVector()
{
  // establish opening bracket
  cout << "[";
  // if there's an element present, add the elements
  if(count > 0)
  {
    // if there's just one element, we just add the first element
    if (count == 1)
    {
    cout << v[0];
    // there are multiple elements -- loop through them all
    } else {
      for(int i = 0; i < count - 1; i++) {
        cout << v[i] << ", ";
      }
      // print out the last element
      cout << v[count - 1];
    }
  }
  // close out the braces
  cout << "]" << endl;
}
```

Lab 1.3

```
#include <iostream>
#include <string>

using namespace std;

// vector values
double *v;
int count;
```

```
int size;
// string representing menu options
string menuString = "Main menu:\n1. Print the array\n2. Append element at the end\n3. Remove last
element\n4. Insert one element\n5. Exit \nSelect an option: _";

// Initialize method stub
void Initialize();
// Finalize method stub
void Finalize();
// runMenu method stub
int RunMenu();
// selectStatement method stub
int SelectStatement(char selection);
// Grow method stub
void Grow();
// AddElement method stub
void AddElement(double element);
// PrintVector method stub
void PrintVector();
// RemoveElement method stub
void RemoveElement();

/*
 * Initializes the vector values and runs the menu.
 * @return : int representing the user's desire to quit the program
 */
int main()
{
  Initialize();
  return RunMenu();
}

/*
 * Runs the menu, displaying menu options and getting user input.
 * @return : int if the program is exited, representing the user wanting to quit the program
 */
int RunMenu()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return SelectStatement(selection);
}
```

```
/*
 * Represents logic for performing the desired action as specified by the user.
 * @param : selection the character selected by the user to be used for the switch
 * @return : 0 if the program is exited, otherwise recurs on runMenu()
 */
int SelectStatement(char selection)
{
  double newElem;
  switch (selection) {
    // print the array
    case '1':
      cout << "Print the array" << endl;
      PrintVector();
      RunMenu();
    // append element at end
    case '2':
      cout << "Append element at the end" << endl;
      cout << "Enter the new element: _";
      cin >> newElem;
      AddElement(newElem);
      RunMenu();
    // remove last element
    case '3':
      cout << "Remove last element" << endl;
      RemoveElement();
      RunMenu();
    // insert one element
    case '4':
      cout << "Insert one element" << endl;
      RunMenu();
    // exit the program
    case '5':
      Finalize();
      return 0;
      break;
    // if invalid, display an error and run the menu again
    default:
      cout << "Error: please enter a valid option" << endl;
      RunMenu();
  }
}
```

```
/*
 * Initializes vector values
 */
void Initialize()
{
  size = 0;
  count = 0;
  v = new double[size];
}

/*
 * Deletes vector to free up memory space
 */
void Finalize()
{
  delete[] v;
}

/*
 * Doubles the size in memory of the array v .
 */
void Grow()
{
  // initializes pointer to nv
  double *nv;
  // sets nvSize equal to a doubling of the current size
  int nvSize = size * 2;
  // initializes nv
  nv = new double[nvSize];
  // for each element in v, set that same index in nv equal to its value
  for(int i = 0; i < count; i++)
  {
    double val = v[i];
    nv[i] = val;
  }
  // delete v
  delete[] v;
  // set to now point to nv
  v = nv;

  // print the details of the Grow() call
  cout << "Vector grown" << endl;
  cout << "Previous capacity: " << size << " elements" << endl;
```

```
  cout << "New capacity: " << nvSize << " elements" << endl;
  cout << "Count: " << count << endl;
  cout << "Size: " << size << endl;

    // set the new size equal to nvSize
    size = nvSize;
}

/*
 * adds the passed-in element to the end of the vector.
 * @param double : the element to add to the end of the vector
 */
void AddElement(double element)
{
  // if the size is 0, we want to add the element at the first slot and increment count and size
  if(size == 0)
  {
    v[0] = element;
    count = count + 1;
    size = size + 1;
  }
  // otherwise, if size is greater than count, we don't need to Grow the vector, so just add it
  else if(size > count)
  {
    v[count] = element;
    count = count + 1;
  }
  // size is equal to count -- the vector needs to be grown  before more can be added
  else
  {
    // grow the vector
    Grow();
    // recursively call AddElement now that the vector has grown
    AddElement(element);
  }
}

/*
 * Prints the vector to the console.
 */
void PrintVector()
{
  // establish opening bracket
```

```
 cout << "[";
 // if there's an element present, add the elements
 if(count > 0)
 {
  // if there's just one element, we just add the first element
  if (count == 1)
  {
  cout << v[0];
  // there are multiple elements -- loop through them all
  } else {
    for(int i = 0; i < count - 1; i++) {
      cout << v[i] << ", ";
    }
    // print out the last element
    cout << v[count - 1];
  }
 }
 // close out the braces
 cout << "]" << endl;
}


/*
 * Removes an element from the end of the Vector.
 */
 void RemoveElement()
 {
  // initializes pointer to cv
  double *cv;
  // sets cvSize equal to the size of v
  int cvSize = size;
  // initializes cv
  cv = new double[cvSize];
  // if the size is zero, there's nothing to remove, so tell the user
  if(count == 0)
  {
   cout << "Could not perform: there are no elements left to remove" << endl;
  } else {
   // for each element in v up to but not including the last element,
    // set that same index in cv equal to its value
   for(int i = 0; i < count - 1; i++)
   {
    double val = v[i];
```

```
    cv[i] = val;
   }
   // reduce the count if the count is greater than 0
   if (count > 0) {
    count = count - 1;
   }
   // delete v
   delete[] v;
   // set to now point to cv
   v = cv;

  }
}
```

Lab 1.4

```
#include <iostream>
#include <string>

using namespace std;

// vector values
double *v;
int count;
int size;
// string representing menu options
string menuString = "Main menu:\n1. Print the array\n2. Append element at the end\n3. Remove last
element\n4. Insert one element\n5. Exit \nSelect an option: _";

// Initialize method stub
void Initialize();
// Finalize method stub
void Finalize();
// runMenu method stub
int RunMenu();
// selectStatement method stub
int SelectStatement(char selection);
// Grow method stub
void Grow();
// AddElement method stub
void AddElement(double element);
// PrintVector method stub
void PrintVector();
```

```
// RemoveElement method stub
void RemoveElement();
// InsertElement method stub
void InsertElement(int index, double element);

/*
 * Initializes the vector values and runs the menu.
 * @return : int representing the user's desire to quit the program
 */
int main()
{
  Initialize();
  return RunMenu();
}

/*
 * Runs the menu, displaying menu options and getting user input.
 * @return : int if the program is exited, representing the user wanting to quit the program
 */
int RunMenu()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return SelectStatement(selection);
}

/*
 * Represents logic for performing the desired action as specified by the user.
 * @param : selection the character selected by the user to be used for the switch
 * @return : 0 if the program is exited, otherwise recurs on runMenu()
 */
int SelectStatement(char selection)
{
  double newElem;
  int index;
  double elem;
  switch (selection) {
    // print the array
    case '1':
      cout << "Print the array" << endl;
      PrintVector();
      RunMenu();
```

```
    // append element at end
    case '2':
      cout << "Append element at the end" << endl;
      cout << "Enter the new element: _";
      cin >> newElem;
      AddElement(newElem);
      RunMenu();
    // remove last element
    case '3':
      cout << "Remove last element" << endl;
      RemoveElement();
      RunMenu();
    // insert one element
    case '4':
      cout << "Insert one element" << endl;
      cout << "Enter the index of new element: _";
      cin >> index;
      cout << endl;
      cout << "Enter the new element: _";
      cin >> elem;
      InsertElement(index, elem);
      RunMenu();
    // exit the program
    case '5':
      Finalize();
      return 0;
      break;
    // if invalid, display an error and run the menu again
    default:
      cout << "Error: please enter a valid option" << endl;
      RunMenu();
  }
}

/*
 * Initializes vector values
 */
void Initialize()
{
  size = 0;
  count = 0;
  v = new double[size];
}
```

```
/*
 * Deletes vector to free up memory space
 */
void Finalize()
{
  delete[] v;
}

/*
 * Doubles the size in memory of the array v .
 */
void Grow()
{
  // initializes pointer to nv
  double *nv;
  // sets nvSize equal to a doubling of the current size
  int nvSize = size * 2;
  // initializes nv
  nv = new double[nvSize];
  // for each element in v, set that same index in nv equal to its value
  for(int i = 0; i < count; i++)
  {
    double val = v[i];
    nv[i] = val;
  }
  // delete v
  delete[] v;
  // set to now point to nv
  v = nv;

  // print the details of the Grow() call
  cout << "Vector grown" << endl;
  cout << "Previous capacity: " << size << " elements" << endl;
  cout << "New capacity: " << nvSize << " elements" << endl;

  // set the new size equal to nvSize
  size = nvSize;
}
/*
 * adds the passed-in element to the end of the vector.
 * @param double : the element to add to the end of the vector
 */
```

```
void AddElement(double element)
{
 // if the size is 0, we want to add the element at the first slot and increment count and size
 if(size == 0)
 {
  v[0] = element;
  count = count + 1;
  size = size + 1;
 }
 // otherwise, if size is greater than count, we don't need to Grow the vector, so just add it
 else if(size > count)
 {
  v[count] = element;
  count = count + 1;
 }
 // size is equal to count -- the vector needs to be grown  before more can be added
 else
 {
  // grow the vector
  Grow();
  // recursively call AddElement now that the vector has grown
  AddElement(element);
 }
}

/*
 * Prints the vector to the console.
 */
void PrintVector()
{
 // establish opening bracket
 cout << "[";
 // if there's an element present, add the elements
 if(count > 0)
 {
  // if there's just one element, we just add the first element
  if (count == 1)
  {
  cout << v[0];
  // there are multiple elements -- loop through them all
  } else {
   for(int i = 0; i < count - 1; i++) {
    cout << v[i] << ", ";
```

```
    }
    // print out the last element
    cout << v[count - 1];
   }
  }
  // close out the braces
  cout << "]" << endl;
}

/*
 * Removes an element from the end of the Vector.
 */
 void RemoveElement()
 {
  // initializes pointer to cv
  double *cv;
  // sets cvSize equal to the size of v
  int cvSize = size;
  // initializes cv
  cv = new double[cvSize];
  // if the size is zero, there's nothing to remove, so tell the user
  if(count == 0)
  {
   cout << "Could not perform: there are no elements left to remove" << endl;
  } else {
   // for each element in v up to but not including the last element,
    // set that same index in cv equal to its value
   for(int i = 0; i < count - 1; i++)
   {
    double val = v[i];
    cv[i] = val;
   }
   // reduce the count if the count is greater than 0
   if (count > 0) {
    count = count - 1;
   }
   // delete v
   delete[] v;
   // set to now point to cv
   v = cv;

  }
 }
```

```
/*
 * Inserts the given element at the given index. The index should be between 0 and count.
 * @Param index : int - the index for which to insert the element
 * @ Param element : double - the element to insert at the given index
 */
 void InsertElement(int index, double element) {
   // check for invalid params - the index should be in the range [0, count]
   if (index > count || index < 0)
   {
     // print the error message
     cout << "Index out of range. Please pick an index between 0 and the vector's count" << endl;
   // the index is valid - insert the element at the appropriate index
   } else {
     // Grow the vector preemptively to account for the new element
     Grow();
   // shift the elements in the vector up to the index inserted at one to the right
    for(int i = count; i >= index + 1; i--)
    {
      v[i] = v[i - 1];
    }
    // insert the element at the index specified
    v[index] = element;
    // increase the count of the vector
    count = count + 1;
  }
 }
```

Lab 1.5

```
#include <iostream>
#include <string>

using namespace std;

// vector values
double *v;
int count;
int size;
// string representing menu options
string menuString = "Main menu:\n1. Print the array\n2. Append element at the end\n3. Remove last
element\n4. Insert one element\n5. Exit \nSelect an option: _";
```

```
// Initialize method stub
void Initialize();
// Finalize method stub
void Finalize();
// runMenu method stub
int RunMenu();
// selectStatement method stub
int SelectStatement(char selection);
// Grow method stub
void Grow();
// AddElement method stub
void AddElement(double element);
// PrintVector method stub
void PrintVector();
// RemoveElement method stub
void RemoveElement();
// InsertElement method stub
void InsertElement(int index, double element);
// Shrink method stub
void Shrink();

/*
 * Initializes the vector values and runs the menu.
 * @return : int representing the user's desire to quit the program
 */
int main()
{
  Initialize();
  return RunMenu();
}

/*
 * Runs the menu, displaying menu options and getting user input.
 * @return : int if the program is exited, representing the user wanting to quit the program
 */
int RunMenu()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return SelectStatement(selection);
}
```

```
/*
 * Represents logic for performing the desired action as specified by the user.
 * @param : selection the character selected by the user to be used for the switch
 * @return : 0 if the program is exited, otherwise recurs on runMenu()
 */
int SelectStatement(char selection)
{
  double newElem;
  int index;
  double elem;
  switch (selection) {
    // print the array
    case '1':
      cout << "Print the array" << endl;
      PrintVector();
      RunMenu();
      break;
    // append element at end
    case '2':
      cout << "Append element at the end" << endl;
      cout << "Enter the new element: _";
      cin >> newElem;
      AddElement(newElem);
      RunMenu();
      break;
    // remove last element
    case '3':
      cout << "Remove last element" << endl;
      RemoveElement();
      RunMenu();
      break;
    // insert one element
    case '4':
      cout << "Insert one element" << endl;
      cout << "Enter the index of new element: _";
      cin >> index;
      cout << endl;
      cout << "Enter the new element: _";
      cin >> elem;
      InsertElement(index, elem);
      RunMenu();
      break;
```

```
    // exit the program
    case '5':
      Finalize();
      return 0;
      break;
    // if invalid, display an error and run the menu again
    default:
      cout << "Error: please enter a valid option" << endl;
      RunMenu();
      break;
  }
}

/*
 * Initializes vector values
 */
void Initialize()
{
  size = 0;
  count = 0;
  v = new double[size];
}

/*
 * Deletes vector to free up memory space
 */
void Finalize()
{
  delete[] v;
}

/*
 * Doubles the size in memory of the array v .
 */
void Grow()
{
  // initializes pointer to nv
  double *nv;
  // sets nvSize equal to a doubling of the current size
  int nvSize = size * 2;
  // initializes nv
  nv = new double[nvSize];
  // for each element in v, set that same index in nv equal to its value
```

```
  for(int i = 0; i < count; i++)
  {
    double val = v[i];
    nv[i] = val;
  }
  // delete v
  delete[] v;
  // set to now point to nv
  v = nv;

  // print the details of the Grow() call
  cout << "Vector grown" << endl;
  cout << "Previous capacity: " << size << " elements" << endl;
  cout << "New capacity: " << nvSize << " elements" << endl;

  // set the new size equal to nvSize
  size = nvSize;
}
/*
 * adds the passed-in element to the end of the vector.
 * @param double : the element to add to the end of the vector
 */
void AddElement(double element)
{
  // if the size is 0, we want to add the element at the first slot and increment count and size
  if(size == 0)
  {
    v[0] = element;
    count = count + 1;
    size = size + 1;
  }
  // otherwise, if size is greater than count, we don't need to Grow the vector, so just add it
  else if(size > count)
  {
    v[count] = element;
    count = count + 1;
  }
  // size is equal to count -- the vector needs to be grown  before more can be added
  else
  {
    // grow the vector
    Grow();
    // recursively call AddElement now that the vector has grown
```

```
    AddElement(element);
  }
}

/*
 * Prints the vector to the console.
 */
void PrintVector()
{
  // establish opening bracket
  cout << "[";
  // if there's an element present, add the elements
  if(count > 0)
  {
    // if there's just one element, we just add the first element
    if (count == 1)
    {
    cout << v[0];
    // there are multiple elements -- loop through them all
    } else {
      for(int i = 0; i < count - 1; i++) {
        cout << v[i] << ", ";
      }
      // print out the last element
      cout << v[count - 1];
    }
  }
  // close out the braces
  cout << "]" << endl;
}

/*
 * Removes an element from the end of the Vector.
 */
 void RemoveElement()
 {
   // initializes pointer to cv
   double *cv;
   // sets cvSize equal to the size of v
   int cvSize = size;
   // initializes cv
   cv = new double[cvSize];
   // if the size is zero, there's nothing to remove, so tell the user
```

```
  if(count == 0)
  {
   cout << "Could not perform: there are no elements left to remove" << endl;
  } else {
   // for each element in v up to but not including the last element,
    // set that same index in cv equal to its value
   for(int i = 0; i < count - 1; i++)
   {
    double val = v[i];
    cv[i] = val;
   }
   // reduce the count if the count is greater than 0
   if (count > 0) {
    count = count - 1;
   }
   // delete v
   delete[] v;
   // set to now point to cv
   v = cv;

  if(count < size * 0.3) {
   Shrink();
  }
  }
}


/*
 * Inserts the given element at the given index. The index should be between 0 and count.
 * @Param index : int - the index for which to insert the element
 * @ Param element : double - the element to insert at the given index
 */
 void InsertElement(int index, double element) {
  // check for invalid params - the index should be in the range [0, count]
  if (index > count || index < 0)
  {
   // print the error message
   cout << "Index out of range. Please pick an index between 0 and the vector's count" << endl;
  // the index is valid - insert the element at the appropriate index
  } else {
   // Grow the vector preemptively to account for the new element
   Grow();
  // shift the elements in the vector up to the index inserted at one to the right
```

```
   for(int i = count; i >= index + 1; i--)
   {
     v[i] = v[i - 1];
   }
   // insert the element at the index specified
   v[index] = element;
   // increase the count of the vector
   count = count + 1;
 }
}

/*
 * Reallocates the Vector with 50% capacity.
 */
 void Shrink()
 {
  // initialize pointer to the small vector (sv)
  double *sv;
  // sets svSize equal to the size of v divided by 2
  int svSize = size / 2;
  // initializes sv
  sv = new double[svSize];
  // for each element in v, set that same index in sv equal to its value
  for(int i = 0; i < count; i++)
  {
    double val = v[i];
    sv[i] = val;
  }
  // delete v
  delete[] v;
  // set to now point to sv
  v = sv;

  // print debug info
  cout << "Shrunk the vector" << endl;
  cout << "Previous capacity: " << size << endl;
  cout << "New capacity: " << svSize << endl;
  // set the size equal to the new size
  size = svSize;
 }
```