Fundamentals of Programing

Lab Manual # 10

**Lab Instructor:** Muhammad Affan

## Student Name:

## CMS ID:

**DATE:**

# Lab Manual # 10

**Vectors and Object Oriented Programming**

# Objectives

The objectives of this lab are:

* Learn about Vectors
* Learn about Object Oriented Programming

# Description

A Dynamic array automatically grows when we try to make an insertion and there is no more space left for the new item. Usually the area doubles in size. A simple dynamic array can be constructed by allocating an array of fixed-size, typically larger than the number of elements immediately required. The elements of the dynamic array are stored contiguously at the start of the underlying array, and the remaining positions towards the end of the underlying array are reserved, or unused. Elements can be added at the end of a dynamic array in constant time by using the reserved space until this space is completely consumed. When all space is consumed, and an additional element is to be added, the underlying fixed-sized array needs to be increased in size. Typically resizing is expensive because you have to allocate a bigger array and copy over all of the elements from the array you have overgrow before we can finally append our item.

**Approach:** When we enter an element in array but array is full then you create a function, this function creates a new array double size or as you wish and copy all element from the previous array to a new array and return this new array. Also, we can reduce the size of the array. and add an element at a given position, remove the element at the end default and at the position also.

**Key Features of Dynamic Array**

**Add Element:** Add element at the end if the array size is not enough then extend the size of the array and add an element at the end of the original array as well as given index. In dynamic array you can create fixed-size array when required added some more element in array then use this approach:

A diagram of a number

Description automatically generated

**Delete Element:** Delete an element from array, default remove() method delete an element from end, simply store zero at last index and you also delete element at specific index by calling removeAt(i) method where i is index. removeAt(i) method shift all right element in the left side from the given index.A screenshot of a graph

Description automatically generated

**Resize of Array Size:** When the array has null/zero data (aside from an element added by you) at the right side of the array, meaning it has unused memory, the method shrinkSize() can free up the extra memory. When all space is consumed, and an additional element is to be added, then the underlying fixed-size array needs to increase in size. Typically resizing is expensive because you have to allocate a bigger array and copy over all of the elements from the array you have outgrown before we can finally append our item.A screenshot of a computer

Description automatically generated

Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

**What is vector in C++?**

**Vector**in C++ is the class template that contains the vector container and its member functions. It is defined inside the **<vector>** header file. The member functions of std::vector class provide various functionalities to vector containers. Some commonly used member functions are written below:

**Iterators**

1. [begin()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/) – Returns an iterator pointing to the first element in the vector
2. [end()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/) – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3. [rbegin()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/) – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4. [rend()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/) – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
5. [cbegin()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/) – Returns a constant iterator pointing to the first element in the vector.
6. [cend()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/) – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
7. [crbegin()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/) – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
8. [crend()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/) – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

// C++ program to illustrate the

// iterators in vector

#include <iostream>

#include <vector>

**using** **namespace** std;

**int** main()

{

    vector<**int**> g1;

**for** (**int** i = 1; i <= 5; i++)

        g1.push\_back(i);

    cout << "Output of begin and end: ";

**for** (**auto** i = g1.begin(); i != g1.end(); ++i)

        cout << \*i << " ";

    cout << "\nOutput of cbegin and cend: ";

**for** (**auto** i = g1.cbegin(); i != g1.cend(); ++i)

        cout << \*i << " ";

    cout << "\nOutput of rbegin and rend: ";

**for** (**auto** ir = g1.rbegin(); ir != g1.rend(); ++ir)

        cout << \*ir << " ";

    cout << "\nOutput of crbegin and crend : ";

**for** (**auto** ir = g1.crbegin(); ir != g1.crend(); ++ir)

        cout << \*ir << " ";

**return** 0;

}

**Output:**

Output of begin and end: 1 2 3 4 5

Output of cbegin and cend: 1 2 3 4 5

Output of rbegin and rend: 5 4 3 2 1

Output of crbegin and crend : 5 4 3 2 1

### Capacity

1. [size()](https://www.geeksforgeeks.org/vectorempty-vectorsize-c-stl/) – Returns the number of elements in the vector.
2. [max\_size()](https://www.geeksforgeeks.org/vector-max_size-function-in-c-stl/) – Returns the maximum number of elements that the vector can hold.
3. [capacity()](https://www.geeksforgeeks.org/vector-capacity-function-in-c-stl/) – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
4. [resize(n)](https://www.geeksforgeeks.org/vector-resize-c-stl/) – Resizes the container so that it contains ‘n’ elements.
5. [empty()](https://www.geeksforgeeks.org/vectorempty-vectorsize-c-stl/) – Returns whether the container is empty.
6. [shrink\_to\_fit()](https://www.geeksforgeeks.org/vector-shrink_to_fit-function-in-c-stl/) – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
7. [reserve()](https://www.geeksforgeeks.org/using-stdvectorreserve-whenever-possible/)– Requests that the vector capacity be at least enough to contain n elements.

|  |
| --- |
| // C++ program to illustrate the  // capacity function in vector  #include <iostream>  #include <vector>    **using** **namespace** std;    **int** main()  {      vector<**int**> g1;    **for** (**int** i = 1; i <= 5; i++)          g1.push\_back(i);        cout << "Size : " << g1.size();      cout << "\nCapacity : " << g1.capacity();      cout << "\nMax\_Size : " << g1.max\_size();        // resizes the vector size to 4      g1.resize(4);        // prints the vector size after resize()      cout << "\nSize : " << g1.size();        // checks if the vector is empty or not  **if** (g1.empty() == **false**)          cout << "\nVector is not empty";  **else**          cout << "\nVector is empty";        // Shrinks the vector      g1.shrink\_to\_fit();      cout << "\nVector elements are: ";  **for** (**auto** it = g1.begin(); it != g1.end(); it++)          cout << \*it << " ";    **return** 0;  } |

**Output:**

Size : 5

Capacity : 8

Max\_Size : 4611686018427387903

Size : 4

Vector is not empty

Vector elements are: 1 2 3 4

### Element access

1. [reference operator [g]](https://www.geeksforgeeks.org/vectoroperator-vectoroperator-c-stl/) – Returns a reference to the element at position ‘g’ in the vector
2. [at(g)](https://www.geeksforgeeks.org/vectorat-vectorswap-c-stl/) – Returns a reference to the element at position ‘g’ in the vector
3. [front()](https://www.geeksforgeeks.org/vectorfront-vectorback-c-stl/) – Returns a reference to the first element in the vector
4. [back()](https://www.geeksforgeeks.org/vectorfront-vectorback-c-stl/) – Returns a reference to the last element in the vector
5. [data()](https://www.geeksforgeeks.org/vector-data-function-in-c-stl/) – Returns a direct pointer to the memory array used internally by the vector to store its owned elements.

|  |
| --- |
| // C++ program to illustrate the  // element access in vector  #include <bits/stdc++.h>  **using** **namespace** std;    **int** main()  {      vector<**int**> g1;    **for** (**int** i = 1; i <= 10; i++)          g1.push\_back(i \* 10);        cout << "\nReference operator [g] : g1[2] = " << g1[2];        cout << "\nat : g1.at(4) = " << g1.at(4);        cout << "\nfront() : g1.front() = " << g1.front();        cout << "\nback() : g1.back() = " << g1.back();        // pointer to the first element  **int**\* pos = g1.data();        cout << "\nThe first element is " << \*pos;  **return** 0;  } |

**Output:**

Reference operator [g] : g1[2] = 30

at : g1.at(4) = 50

front() : g1.front() = 10

back() : g1.back() = 100

The first element is 10

### ****Modifiers****

1. [assign()](https://www.geeksforgeeks.org/vector-assign-in-c-stl/)– It assigns new value to the vector elements by replacing old ones
2. [push\_back()](https://www.geeksforgeeks.org/vectorpush_back-vectorpop_back-c-stl/) – It push the elements into a vector from the back
3. [pop\_back()](https://www.geeksforgeeks.org/vectorpush_back-vectorpop_back-c-stl/) – It is used to pop or remove elements from a vector from the back.
4. [insert()](https://www.geeksforgeeks.org/vector-insert-function-in-c-stl/) – It inserts new elements before the element at the specified position
5. [erase()](https://www.geeksforgeeks.org/vectorclear-vectorerase-c-stl/) – It is used to remove elements from a container from the specified position or range.
6. [swap()](https://www.geeksforgeeks.org/vectorat-vectorswap-c-stl/) – It is used to swap the contents of one vector with another vector of same type. Sizes may differ.
7. [clear()](https://www.geeksforgeeks.org/vectorclear-vectorerase-c-stl/) – It is used to remove all the elements of the vector container
8. [emplace()](https://www.geeksforgeeks.org/vector-emplace-function-in-c-stl/) – It extends the container by inserting new element at position
9. [emplace\_back()](https://www.geeksforgeeks.org/vectoremplace_back-c-stl/) – It is used to insert a new element into the vector container, the new element is added to the end of the vector

|  |
| --- |
| // C++ program to illustrate the  // Modifiers in vector  #include <bits/stdc++.h>  #include <vector>  **using** **namespace** std;    **int** main()  {      // Assign vector      vector<**int**> v;        // fill the vector with 10 five times      v.assign(5, 10);        cout << "The vector elements are: ";  **for** (**int** i = 0; i < v.size(); i++)          cout << v[i] << " ";        // inserts 15 to the last position      v.push\_back(15);  **int** n = v.size();      cout << "\nThe last element is: " << v[n - 1];        // removes last element      v.pop\_back();        // prints the vector      cout << "\nThe vector elements are: ";  **for** (**int** i = 0; i < v.size(); i++)          cout << v[i] << " ";        // inserts 5 at the beginning      v.insert(v.begin(), 5);        cout << "\nThe first element is: " << v[0];        // removes the first element      v.erase(v.begin());        cout << "\nThe first element is: " << v[0];        // inserts at the beginning      v.emplace(v.begin(), 5);      cout << "\nThe first element is: " << v[0];        // Inserts 20 at the end      v.emplace\_back(20);      n = v.size();      cout << "\nThe last element is: " << v[n - 1];        // erases the vector      v.clear();      cout << "\nVector size after clear(): " << v.size();        // two vector to perform swap      vector<**int**> v1, v2;      v1.push\_back(1);      v1.push\_back(2);      v2.push\_back(3);      v2.push\_back(4);        cout << "\n\nVector 1: ";  **for** (**int** i = 0; i < v1.size(); i++)          cout << v1[i] << " ";        cout << "\nVector 2: ";  **for** (**int** i = 0; i < v2.size(); i++)          cout << v2[i] << " ";        // Swaps v1 and v2      v1.swap(v2);        cout << "\nAfter Swap \nVector 1: ";  **for** (**int** i = 0; i < v1.size(); i++)          cout << v1[i] << " ";        cout << "\nVector 2: ";  **for** (**int** i = 0; i < v2.size(); i++)          cout << v2[i] << " ";  } |

**Output:**

The vector elements are: 10 10 10 10 10

The last element is: 15

The vector elements are: 10 10 10 10 10

The first element is: 5

The first element is: 10

The first element is: 5

The last element is: 20

Vector size after erase(): 0

Vector 1: 1 2

Vector 2: 3 4

After Swap

Vector 1: 3 4

Vector 2: 1 2

**Object-oriented programming**:

As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

There are some basic concepts that act as the building blocks of OOPs i.e.

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Passing

We will focus on Classes, Structures and Objects.

Link to Classes and Objects: <https://www.geeksforgeeks.org/c-classes-and-objects/>

Link to Structures: <https://www.geeksforgeeks.org/structures-in-cpp/?ref=lbp>

<https://www.geeksforgeeks.org/structures-unions-and-enumerations-in-cpp/?ref=lbp>

# Tasks (Due in last week of December):

1. Iterate Through Vector Using Iterators and print all pushed elements. Next you need to push integer 5 and remove element at that position.
2. Write a complete C++ program that uses 2 vectors, 1 for names (string) and 1 for grades (int)
   1. Ask the user for the number of name/grade pairs that will be entered.
   2. Display the mean of the grades.
   3. Display the median of the grades.
   4. Display the mode of the grades.
   5. Display the names of the students with the mode as their grade.
3. Write a program to print the area and perimeter of a triangle having sides of 3 m, 4 m and 5 m by creating a class named 'Triangle' with a function to print the area and perimeter.
4. Write a structure to store the names, salary, and hours of work per day of 10 employees in a company. Write a program to increase the salary depending on the number of hours of work per day as follows and then print the name of all the employees along with their final salaries.

|  |  |  |  |
| --- | --- | --- | --- |
| Hours of work per day | 8 | 10 | >=12 |
| Increase in Salary | $50 | $100 | $150 |

# 