Chapter 10 Vectors

Vectors are containers that can store multiple elements like arrays. However, unlike arrays, the size of a vector can shrink or grow automatically. As a result, there is no memory wastage and no need to specify the vector's initial size.

The vector makes insertion and deletion far simpler. It allows insertion and deletion operation anywhere within a sequence. After an element is inserted or deleted, there is no need to do any shifting.

1. Vector Declaration

To use vector, we need to include the `<vector>` header file in our program.

Here is how we declare a vector:

```
vector<type> vector_name;
```

For example, create an empty vector of integers:

```
vector<int> num;
```

2. Vector Initialization

If you want to create a vector of integers with initialized values, you can initialize the vector in the following two ways:

```
vector<int> vector1 = {1, 2, 3, 4, 5};
```

or

```
vector<int> vector1 {1, 2, 3, 4, 5};
```

Here, we are initializing the vector by providing values directly to the vector. Now, the vector1 is initialized with values 1, 2, 3, 4, 5.

If you want to create a vector containing five elements and their values are `0`, do it as below:

```
vector<int> vector3(5);
```

is equivalent to

```
vector<int> vector3 = {0, 0, 0, 0, 0};
```

This code creates an `int` vector with size of 5 and initializes the vector with the value of `0` for all of elements in the vector.

In case you want to create a vector containing a number of elements with a specific value, do it as below:

Let's look at another example below:

```
vector<int> vector2(5, 12);
```

is equivalent to

```
vector<int> vector2 = {12, 12, 12, 12, 12};
```

This code creates an `int` vector with size of 5 and initializes the vector with the value of 12 for all of elements in the vector.

You can copy all elements from a vector to another vector as below:

```
vector<string> vector1 = {"first", "second", "third", "fourth"};
vector<string> vector2(vector1);
```

or

```
vector<string> vector2 = vector1;
```

3. Vector Methods

Here are some common vector methods:

Method	**Description**
`size`()	returns the number of elements present in a vector
`begin`()	returns an iterator that points to the first element of a vector.
`end`()	returns an iterator that points to the past-the-end element of a vector. i.e., vector.`end`() - 1 is the iterator that points to the last element of the vector.
`at`()	returns reference to the element present at a specified location in a vector.
`push_back`()	inserts an element at the back of the vector.
`insert`()	inserts a new element to a specified location once or more than once in a vector.
`pop_back`()	removes elements from the back of a vector.
`erase`()	`erase`(i) removes an element from vector based on index position i. `erase`(i, j) removes a range of elements from a specified location [i, j[in a vector.
`clear`()	removes all elements from a vector.
`front`()	returns the first element of a vector.
`back`()	returns the last element of a vector.
`empty`()	returns 1 (true) if the vector is empty.

A vector iterator is an object that points to the memory address of a vector element.

4. Basic Vector Operations

The vector class provides various methods to perform different operations on vectors. We will look at some commonly used vector operations such as:

- Access elements
- Iterate through a vector
- Update elements
- Add elements
- Delete elements
- …

4.1 Accessing Vector Elements

You access vector elements by index number using square brackets [ ] or method `at()` method.

Example 01: Display elements in a vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vec = {1, 3, 5, 7, 9};

    cout << vec[0] << endl;
    cout << vec.at(1);

    return 0;
}
```

Output

```
1
3
```

Note that the `at()` method is preferred over [ ] because the `at()` method throws an exception whenever the vector is out of bound, while [ ] gives a garbage value.

4.2 Iterating Through a Vector

You iterate through vector elements by index number or via iterators using loops.

Example 02: Display elements in a vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    // Display the vector elements
    for(int i = 0; i < vector1.size(); i++)
        cout << vector1.at(i) << "\t";


    return 0;
}
```

Output

```
1	3	5	7	9
```

You can also use the **range-based `for` loop**, also known as **for each loop**, (introduced in C++11) to iterate through a vector or array.

```
for(type variable_name : vector_name){
   // loop body
}
```

Example 03: Iterate through a vector using the range-based `for` loop.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    // Display the vector elements
    for(int i: vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
1	3	5	7	9
```

4.3 Updating Vector Elements

We can access to a vector element using square brackets [ ] or at() method and update its value.

Example 04: Update elements in a vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    vector1[0] = 10;
    vector1.at(1) = 20;

    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
10      20      5       7       9
```

Example 05: Update elements in a vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    // Update each element by multiply it with 2
    for(int i = 0; i < vector1.size(); i++)
        vector1.at(i) *= 2;

    // Display the vector elements
    for(int i = 0; i < vector1.size(); i++)
        cout << vector1.at(i) << "\t";

    return 0;
}
```

Output

```
2       6       10      14      18
```

If you want to update the elements in the vector using range-based `for` loop, then pass the address of the element to the variable `i` as below:

Example 06: Update the elements in the vector using range-based `for` loop

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    for(int &i: vector1) {
        i *= 2;
    }

    // Display the vector elements
    for(int i: vector1) {
        cout << i << "\t";
    }

    return 0;
}
```

Output

```
2	6	10	14	18
```

You can use the auto keyword which is a part of the “Type Inference” feature. Type Inference refers to the automatic deduction of data-type of variables and functions at compile time. Therefore, the programmer is not obliged to specify the exact data-types and leave it to the compiler. Since the compiler has to do this, it increases the compilation time to a few seconds

Example 07: Re-write the example above using auto keyword.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    // Display the vector elements
    for(auto i: vector1)
        cout << i << " ";

    return 0;
}
```

Output

```
1     3     5     7     9
```

4.4 Adding Vector Elements

To add a single element to the end of the vector, we use the push_back() method.

Example 08: Add elements to a vector using the push_back() method.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    vector1.push_back(11); // Add 11 to the vector
    vector1.push_back(13); // Add 13 to the vector

    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
1     3     5     7     9     11     13
```

We can also use the `insert()` method to add elements to a vector with following syntax:

```
vector_name.insert(position, value);
```

Here, `position` is the iterator which specifies the position where you want to insert the element and `value` is element you want to insert.

Example 09: Add elements to a vector using the `insert()` method.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    vector1.insert(vector1.begin(), 20);     // Add 20 to the vector at index 0
    vector1.insert(vector1.begin() + 2, 30); // Add 30 to the vector at index 2
    vector1.insert(vector1.end() - 1, 40); // Add 40 to the end of the vector


    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
20      1       30      3       5       7       40      9
```

You add element more than once at specified position using `insert()` method with following syntax:

```
vector_name.insert(position, size, value);
```

Here, `size` is the parameter in the `insert()` method which specifies the number of times a specified value is inserted.

Example 10: Add an element more than once to a vector at specified position.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    // Add an element 10 at the front of vector 3 times
    vector1.insert(vector1.begin(), 3, 10);

    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output:

```
10	10	10	1	3	5	7	9
```

4.5 Deleting Vector Elements

To delete a single element, which is the last element, from a vector, we use the pop_back() method.

Example 11: Delete elements in a vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    vector1.pop_back(); // removes the last element
    vector1.pop_back(); // removes the last element

    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
1    3    5
```

You can use the erase() method to remove an element of a specific position or range of elements from the vector.

```
// Remove single element
vector_name.erase(position);

// Erase range of elements
vector_name.erase(first, last);
```

- `position`: Iterator to the element to be deleted.
- `first`: Iterator to the first element of the range.
- `last`: Iterator to the element one after the last element of the range.

Example 12: Delete elements in a vector at a specific index.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    vector1.erase(vector1.begin() + 3); // removes the element at index 3

    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
1   3   5   9
```

Example 13: Delete elements in a vector at a specific range.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vector1 = {1, 3, 5, 7, 9};

    // removes the elements at index 1 to 3
    vector1.erase(vector1.begin() + 1, vector1.begin() + 4);

    // Display the vector elements
    for(int i : vector1)
        cout << i << "\t";

    return 0;
}
```

Output

```
1     9
```

4.6 Sorting a Vector

We can use the sort() function defined under the header <algorithm> to sort a vector.

Example 14: Sort a vector in ascending order.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main(){
    vector<int> vec = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};

    //Sort the vector
    sort(vec.begin(), vec.end());  // sort(vec.begin(), vec.end(), less<int>());

    //Display the vector
    for(int i = 0; i < vec.size(); i++)
        cout << vec.at(i) << " ";

    return 0;
}
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

By default the sort() functions sorts the vector in ascending order. We can pass greater<type>() function to sort in descending order.

Example 15: Sort a vector in a descending order.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main(){
    vector<string> vec = {"Green", "Black", "Yellow", "Blue"};

    //Sort the vector
    sort(vec.begin(), vec.end(), greater<string>());

    //Display the vector
    for(int i = 0; i < vec.size(); i++)
        cout << vec.at(i) << " ";

    return 0;
}
```

Output
```
Yellow Green Blue Black
```

4.7 Sum Elements in a Vector

Sum up of all elements of a C++ vector can be very easily done by `accumulate`() method defined in `<numeric>` header.

```
vector_name. accumulate(first, last, init);
```

- `first`: Iterator to the first element of the range.
- `last`: Iterator to the element one after the last element of the range.
- `init`: Initial value to start the accumulation with.

Example 16: Sum elements in a vector.
```
#include<iostream>
#include<vector>
#include<numeric>
using namespace std;

int main() {
   vector<int> vec = {2, 7, 6, 10};

   int sum = accumulate(vec.begin(),vec.end(), 0);

   cout<< "sum = " << sum;
}
```

Output:
```
sum = 25
```

5. Vectors and Functions

5.1 Passing Vectors to a Function

When a vector is passed to a function, a copy of the vector is created. This new copy of the vector is then used in the function and thus, any changes made to the vector in the function do not affect the original vector.

Example 17: Pass a vector to a function (Pass by value)

```
#include <iostream>
#include <vector>
using namespace std;

void func(vector<int> vect){
    vect.push_back(30);
}

int main() {
    vector<int> vector1 = {1, 2, 3, 4};

    func(vector1);

    //Display the elements
    for(int i = 0; i < vector1.size(); i++)
        cout << vector1[i] << " ";

    return 0;
}
```

Output

```
1 2 3 4
```

Example 18: Pass a vector to a function (Pass by reference using the address operator &)

```
#include <iostream>
#include <vector>
using namespace std;

void func(vector<int> &vect){
    vect.push_back(30);
}

int main() {
    vector<int> vector1 = {1, 2, 3, 4};

    func(vector1);

    //Display the elements
    for(int i = 0; i < vector1.size(); i++)
        cout << vector1[i] << " ";

    return 0;
}
```

Output

```
1 2 3 4 30
```

Practice

1. Create a function that sorts a vector in descending order.
2. Create a function that returns the sum the elements of a vector.

5.2 Returning a Vector from a Function

Example 19: Return a vector from a function

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> func(vector<int> vect){
    vect.push_back(30);

    return vect;
}

int main() {
    vector<int> vector1 = {1, 2, 3, 4};

    vector1 = func(vector1);

    //Display the elements
    for(int i = 0; i < vector1.size(); i++)
        cout << vector1[i] << " ";

    return 0;
}
```

Output:

```
1 2 3 4 30
```

6. Multi-dimensional Vectors

A vector can be placed inside another vector to create a multi-dimensional vector. Multidimensional Vectors refer to Vectors of Vectors. A multi-dimensional vector is almost similar to a multi-dimensional array but the difference is only the dynamic characteristic.

6.1 Two-dimensional Vectors

A 2D vector is a vector of the vector. Like 2D arrays, we can declare and assign values to a 2D vector. By placing a vector specifier within the `type` parameter of another vector, you may specify a Two-dimensional vector. For example:

```
vector<vector<int>> vector2D = {
    {1,  2,  3,  4},
    {5,  6,  7,  8},
    {9, 10, 11, 12}
};
```

The number of elements in the inner vectors of a vector can be different. For example:

```
vector<vector<int>> vector2D = {
    {1,  2},
    {5,  6,  7},
    {9, 10, 11, 12}
};
```

6.1.1 Iterate Through 2D Vectors

Example 20: Iterate through 2D vectors

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<vector<int>> vector2D = {
        {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9, 10, 11, 12}
    };

    //Display the elements
    for (int i = 0; i < vector2D.size(); i++)
        for (int j = 0; j < vector2D[i].size(); j++)
            cout << vector2D.at(i).at(j) << " ";
            // cout << vector2D[i][j] << " ";

    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

Example 21: Iterate through a 2D vector using the range-based for loop.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<vector<int>> vector2D = {
        {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9, 10, 11, 12}
    };

    //Display the elements
    for(vector<int> i : vector2D)
        for (int j : i)
            cout << j << " ";

    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

Example 22: Store elements in a 2D vector by user input.

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main(){
    vector<vector<int> > vector2D;

    int size;
    int number;

    cout<< "Enter the size for a square matrix: ";
    cin >> size;

    // Ask the user to enter the elements
    for (int i = 0; i < size; i++){
        vector<int> vector1D;

        for (int j = 0; j < size; j++){
            cout << "Enter element vector1[" << i << "][" << j << "]: ";
            cin >> number;

            vector1D.push_back(number);
        }
        vector2D.push_back(vector1D);
    }

    //Display the elements
    for (int i = 0; i < vector2D.size(); i++) {
        for (int j = 0; j < vector2D[i].size(); j++)
            cout << vector2D[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Output

```
Enter the size for a square matrix: 2
Enter element vector1[0][0]: 1
Enter element vector1[0][1]: 2
Enter element vector1[1][0]: 3
Enter element vector1[1][1]: 4
1 2
3 4
```

6.1.2 Adding Elements to a 2D Vector

Example 23: Add elements to a 2D vector

```cpp
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

int main() {
    vector<vector<int>> vector2D = {
        {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9, 10, 11, 12}
    };

    vector2D[1].push_back(100);
    vector2D.push_back({10, 20, 30});

    //Display the elements
    for (int i = 0; i < vector2D.size(); i++){
        for (int j = 0; j < vector2D[i].size(); j++)
            cout << left << setw(4) << vector2D[i][j];
        cout << endl;
    }

    return 0;
}
```

Output

```
1   2   3   4
5   6   7   8   100
9   10  11  12
10  20  30
```

6.1.3 Deleting 2D Vector Elements

Example 24: Delete elements from a 2D vector using pop_back() method.

```
#include <iostream>
#include <vector>
using naespace std;

int main() {
    vector<vector<int>> vector2D = {
        {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9, 10, 11, 12}
    };

    vector2D[1].pop_back(); // delete the last element of row index 1
    vector2D.pop_back();    // delete the last row

    //Display the elements
    for (int i = 0; i < vector2D.size(); i++){
        for (int j = 0; j < vector2D[i].size(); j++)
            cout << vector2D[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Output

```
1 2 3 4
5 6 7
```

Example 25: Delete elements from a 2D vector using `erase()` method.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<vector<int>> vector2D = {
        {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9, 10, 11, 12}
    };

    vector2D[1].erase(vector2D[1].begin() + 2);    // delete number 7
    vector2D.erase(vector2D.begin() + 2);          // delete row {9, 10, 11, 12}

    //Display the elements
    for (int i = 0; i < vector2D.size(); i++){
        for (int j = 0; j < vector2D[i].size(); j++)
            cout << vector2D[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Output:

```
1 2 3 4
5 6 8
```

6.2 Three-dimensional Vectors

The 3D vector is a vector of vectors, like the 3D array. It stores elements in the three dimensions. It can be declared and assign values the same as a 3D array. Again, the 3D Vector is a dynamic 3D array which has the capability to resize itself automatically when an element is to be inserted or delete.

Let's declare and initialize a vector of size 2 x 3 x 4 as below:

```
vector<vector<vector<int>>> vector3D {
      {{ 1,  2,  3,  4}, { 5,  6,  7,  8}, { 9, 10, 11, 12}},
      {{13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}}
};
```

6.2.1 Iterating Through a 3D Vectors

Example 26: Iterate through a 3D vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    //A vector of size 2 x 3 x 4
    vector<vector<vector<int>>> vector3D
        {{{ 1,  2,  3,  4}, { 5,  6,  7,  8}, { 9, 10, 11, 12}},
         {{13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}}};

    //Display the elements
    for(int i = 0; i < vector3D.size(); i++)
        for(int j = 0; j < vector3D.at(i).size(); j++)
            for(int k = 0; k < vector3D.at(i).at(j).size(); k++)
                cout << vector3D.at(i).at(j).at(k) << " ";
    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

Example 27: Iterate through a 3D vector using the range-based for loop.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    //Declare and initialize a vector of size 2 x 3 x 4

    vector<vector<vector<int>>> vector3D
        {{{ 1,  2,  3,  4}, { 5,  6,  7,  8}, { 9, 10, 11, 12}},
        {{13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}}};

    //Display the elements
    for(vector<vector<int>> vec2D : vector3D)
        for(vector<int> vec1D : vec2D)
            for (int element : vec1D)
                cout << element << " ";

    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

Exercises

Write the programs below using vectors; you are NOT allowed to use arrays.

1. Write a program that first will read integers from the user into a vector until the user enters 0. Then, remove all duplicates in the vector, and display the vector.
2. Write a program that keeps reading an integer until the user input 0, then displays all the integers that have been read in ascending order.
3. Write a program that keeps reading the name of a country until the user enters "Done". Then, it alphabetically sorts the names of the countries that have been read, and displays those countries.
4. Write the function called `rotate_right` that takes a vector of integers and rotates the contents of the vector to the right by two slots. Numbers that fall off the right should cycle back to the left. For example:
   - If the input vector is `{1, 3, 5, 7}` then the rotated vector should be `{5, 7, 1, 3}`
   - If the input vector is `{1, 2, 3}` then the rotated vector should be `{2, 3, 1}`
5. Write a program to zip two vectors of the same size. For example:
   Original vectors: `{{1, 3}, {5, 7}, {9, 11}}` and `{{2, 4}, {6, 8}, {10, 12, 14}}`
   Zipped vector: `{{1, 3, 2, 4}, {5, 7, 6, 8}, {9, 11, 10, 12, 14}}`
6. Write a function called `remove_int` that will accept an integer and a vector, then remove all the occurrences of the integer from the vector.
7. (Locate the smallest element) Write a function called `locate_smallest` that returns the location of the smallest element in a 2D vector. The return value is a 1D vector that contains two elements. These two elements indicate the row and column indices of the smallest element in the 2D vector.

Reference

[1] Y. Daniel Liang. 'Introduction to Programming with C++', 3e – 2014

[2] Dey, P., & Ghosh, M. 'Computer Fundamentals and Programming in C', 2e – 2013

[3] https://cplusplus.com/doc/tutorial/

[4] https://www.programiz.com/cpp-programming

[5] https://www.studytonight.com/cpp/