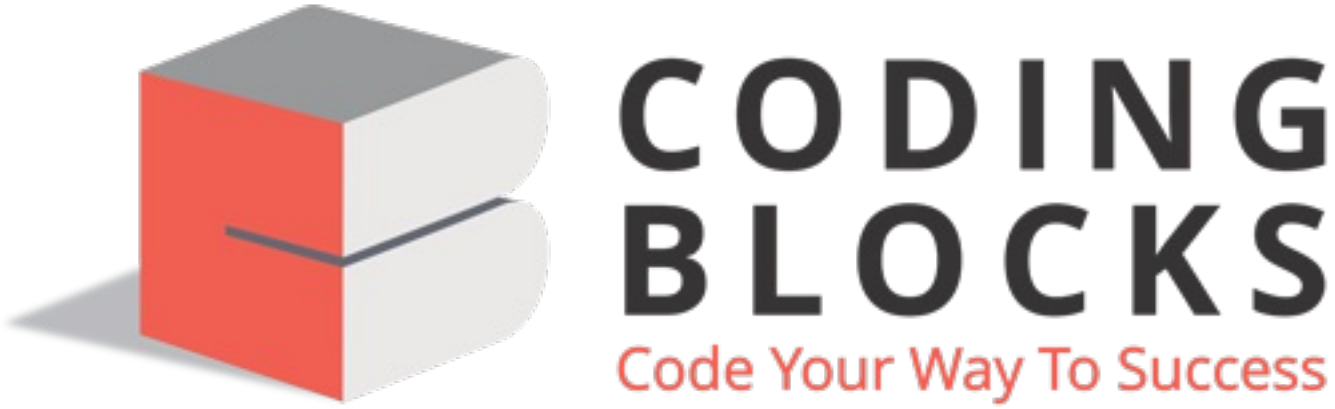

Vectors

Vectors are sequence containers representing arrays that can change in size. Container is a objects that hold data of same type. Sequence containers store elements strictly in linear sequence.

Vector stores elements in contiguous memory locations and enables direct access to any element using subscript operator []. Unlike array, vector can shrink or expand as needed at run time. The storage of the vector is handled automatically.

To support shrink and expand functionality at runtime, vector container may allocate some extra storage to accommodate for possible growth thus container have actual capacity greater than the size. Therefore, compared to array, vector consumes more memory in exchange for the ability to manage storage and grow dynamically in an efficient way.

Zero sized vectors are also valid. In that case vector.begin() and vector.end() points to same location.

Constructor from <vector>

- **Default Constructor**

  This default constructor `std::vector::vector()` constructs an empty container, with zero elements. Size of this container is always zero.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main( ) {
   vector<int> vect;

   cout << "size of vect = " << vect.size() << endl;

   return 0;
}
```

OUTPUT

```
size of vect = 0
```

- **Fill Constructor**

  The fill constructor `std::vector::vector()` constructs a

container of size n and assigns value val(if provided) to each element of the container.

Parameters

n − Size of container.

val − Value to be assigned to each element of container.

Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
   vector<int> vect(5, 0);

   for (int i = 0; i < vect.size(); ++i)
      cout << vect[i] << " ";

   return 0;
}
```

OUTPUT

```
0 0 0 0 0
```

Member Functions

- `size()` returns the number of elements present in the vector.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> vect;

    cout << "vector size = " << vect.size() << endl;

    vect.push_back(1);
    cout << "vector size = " << vect.size() << endl;

    return 0;
}
```

OUTPUT

```
vector size = 0
vector size = 1
```

Time Complexity Constant i.e. O(1)

- `push_back()` inserts new element at the end of vector and

increases size of vector by one.

Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
   vector<int> vect;

   /* Insert 5 elements */
   for (int i = 0; i < 5; ++i)
      vect.push_back(i + 1);

   for (int i = 0; i < vect.size(); ++i)
      cout << vect[i] << " ";

   return 0;
}
```

OUTPUT

```
1 2 3 4 5
```

Time Complexity Constant i.e. O(1)

- `begin()` returns a random access iterator pointing to the first element of the vector.
- `end()` returns an iterator which points to past-the-end element in the vector container. The past-the-end element is the theoretical element that would follow the last element in the vector.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main ()
{
  vector<int> vect;

  for (int i=1; i<=5; i++)
    vect.push_back(i);

  cout << "vect contains:";
  for (vector<int>::iterator it = vect.begin() ; it != vect.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';

  return 0;
}
```

OUTPUT

```
vect contains: 1 2 3 4 5
```

Time Complexity Constant i.e. O(1)

- `empty()` tests whether vector is empty or not. Vector of size zero is considered as empty vector. It returns true if vector is empty otherwise false.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
   vector<int> vect;

   if (vect.empty())
      cout << "Vector is empty" << endl;

   vect.push_back(1);
   vect.push_back(2);
   vect.push_back(3);

   if (!vect.empty())
```

```
        cout << "Vector is not empty" << endl;

    return 0;
}
```

OUTPUT

```
Vector is empty
Vector is not empty
```

Time Complexity Constant i.e. O(1)

- `clear()` destroys the vector by removing all elements from the vector and sets size of vector to zero.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main ()
{
  vector<int> vect;

  for (int i=1; i<=5; i++)
    vect.push_back(i);
```

```
  cout << "Initial size of vector     = " << vect.size() << e
ndl;
   /* destroy vector */
  vect.clear();

  cout << "Size of vector after clear = " << vect.size() << e
ndl;



  return 0;
}
```

OUTPUT

```
Initial size of vector     = 5
Size of vector after clear = 0
```

Time Complexity Linear i.e. O(n)

- `insert()` extends vector by inserting new element at position in container.

  Example:

```
#include <iostream>
#include <vector>
```

```
using namespace std;

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

   vect.insert(vect.begin(), 1);

   for (vector<int>::iterator it = vect.begin(); it != vect.e
nd(); ++it)
      cout << *it << " ";

   return 0;
}
```

OUTPUT

```
1 2 3 4
```

Time Complexity Linear i.e. O(n)

- `front()` returns a reference to the first element of the vector.

  Example:

```
#include <iostream>
#include <vector>
```

```
using namespace std;

int main() {
   vector<int> vect = {1, 2, 3, 4, 5};

   cout << "First element of vector = " << vect.front() << en
dl;

   return 0;
}
```

OUTPUT

```
First element of vector = 1
```

Time Complexity Constant i.e. O(1)

- `erase()` removes single element from the the vector.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
   vector<int> vect = {1, 2, 3, 4, 5};
```

```
    cout << "Original vector" << endl;
    for (vector<int>::iterator it = vect.begin(); it != vect.e
nd(); ++it)
       cout << *it << " ";
    cout<<endl;
    /* Remove first element */
    vect.erase(vect.begin());

    cout << "Modified vector" << endl;
    for (vector<int>::iterator it = vect.begin(); it != vect.e
nd(); ++it)
       cout << *it << " ";

    return 0;
}
```

OUTPUT

```
Original vector
1 2 3 4 5
Modified vector
2 3 4 5
```

Time Complexity Linear i.e. O(n)

Other Useful Functions

- `at()` returns reference to the element present at location n in the vector.

```
vector<int> vect = {1,2,3,4};
vect.at(0); //prints element at index 0 which is 1
```

- `back()` returns a reference to the last element of the vector.

```
vector<int> vect = {1,2,3,4};
vect.back(); //prints the last element which is 4
```

- `capacity()` returns the size of allocated storage, expressed in terms of elements.
- `swap()` exchanges the content of one vector v1 with contents of other vector v2.

  Example:

```
#include <iostream>
#include <vector>

using namespace std;

int main(void) {
   vector<int> v1;
   vector<int> v2 = {1, 2, 3, 4, 5};
```

```
    v1.swap(v2);

    cout << "Vector v1 contains" << endl;
    for (int i = 0; i < v1.size(); ++i)
        cout << v1[i] << " ";

    return 0;
}
```

OUTPUT

```
Vector v1 contains
1 2 3 4 5
```