⇒ Way 2
vector<int> v1(5,20);
vector<int> v2;
v2 = v1;

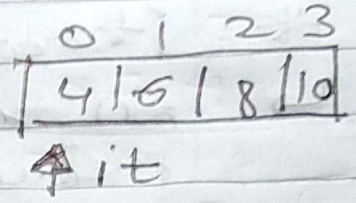
Iterator allows us to traverse the element of a container

26/07/24

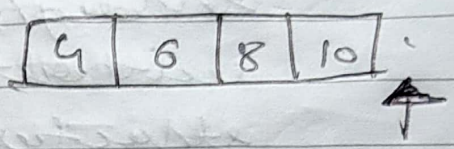
* Iterator is a vector.

⇒ v.begin() → Return the beginning position

~~for(auto it = v.begin();~~
it

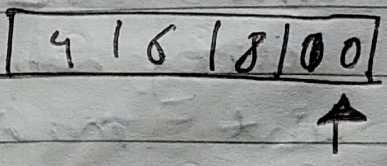


⇒ v.end()
↳ Return the after end position of the container.

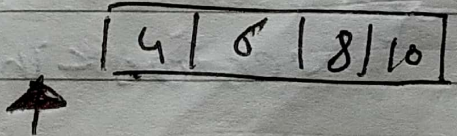



```
for (auto it = v.begin(); it != v.end(); it++)
{
    cout << *it << " ";
}
```

⇒ v.s begin():-
↳ return the end position of the container.

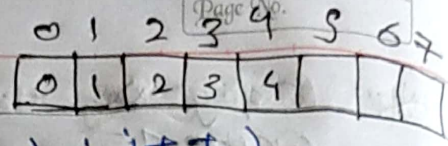


⇒ v.s end():-
↳ return the one before the beginning position.



Date :
Page No.

* Printing of vector.




```
for(i=0 ; i < v.size() ; i++)
{
    cout << v[i];
}
```

* reverse_iterator
↳ It is a iterator adaptor that reverses the direction, which allows us to iterate over a container in a opposite order.

* iterator
↳ An iterator is an object that allows you to ~~traverse~~ traverse through the ~~it~~ container sequentially without exposing the underlying structure of the container.

* prog 1 :- vector iterator.
↳ this program include iterator, reverse iterator, begin(), end(), v.begin() & v.end() operations.

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int> v={2,4,6,8,10};
```

```
vector<int>::iterator it = v.begin();
cout << "First Element of the
        vector" << *it << endl;

it = v.end();
it--;
cout << "Last Element of the
        vector" << *it << endl;

vector<int>::reverse_iterator rit =
        v.begin();
cout << "First Element of the
        vector" << *rit << endl;
rit = v.end();
rit--;
cout << "Last Element of the
        vector" << *rit << endl;

return 0;
}
```

⇒ output :-

```
First Element of the vector :- 2
Last Element of the vector :- 10
First Element of the vector :- 10
Last Element of the vector :- 2
```

Date :
Page No.

* Prog 2 :- Printing vector using iterator operation.

```
#include <iostream>
#include < vector >
using namespace std;
int main ()
{
  vector<int> V ={ 2, 4, 6, 8, 10};
  cout << "Printing Vector :- " << endl;
  for (auto it = v.begin (); it != v.end ();
                it++ )
  {
      cout << * it << "  ";
  }
  cout << endl;
  cout << "Printing vector in reverse
        direction :- " << endl;
  for ( auto jt = v.rbegin () ;
      jt != v.rend () ; jt ++)
  {
      cout << * jt << endl;
  }
return 0;
}
```

Printing Vector :-
2 4 6 8 10
Printing vector in reverse direction :-
10 8 6 4 2

Date :
Page No.

* auto keyword :-

↳ It is used to automatically deduce the type of variable at compile time. This can simplify code & improve readability especially in cases where the type is complex or lengthy.

```
auto x = 10;
// x is automatically deduced to be
   type of int.

auto y = 3.14;
// y is automatically deduced to be
   type of float.
```

---

* prog 3:- merging two vector

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
  vector<int> v={2,4,6,8,10};
  cout<<"Vector 1:-"<<endl;
  for(size_t i=0;i<v.size();i++)
  {
     cout<<v[i]<<" ";
  }
  cout<<"Vector 2:-"<<endl;
  vector<int> v2(2,50);
  for(size_t j=0;j<v2.size();j++)
  {
     cout<<v2[j]<<" ";
  }
```

```
cout<<"\n Merge Vector:-"<<endl;
v.insert(v.begin()+2,v2.begin(),
        v2.end());
for(size_t i=0;i<v.size();i++)
{
    cout<<v[i]<<" ";
}
return 0;
}
```

Vector 1:-
2 4 6 8 10
Vector 2:-
50 50
Merge Vector:-
2 4 50 50 6 8 10

Prog 4:-

Swap() function

⇒ It is used to exchange the contents of two vectors.

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int> v1={2,4,6,8,10};
    vector<int> v2={1,3,5,7,9};
    cout<<"Before Swapping:-"<<endl;
    cout<<"Vector 1:-"<<endl;
```

```
for ( size_t i= 0; i< v1.size(); i++)
{
    cout << v1[i] << " ";
}
cout << "\n Vector 2:- " << endl;
for ( size_t i =0 ; i < v2.size();i++)
{
    cout << v2[i] << " ";
}

v2.swap(v1);
cout << "\n After swapping:- " << endl;
cout << " Vector 1:- " << endl;
for( size_t i=0;i<v1.size();i++)
{
    cout << v2[i] << " ";
}
cout << "\n Vector 2:- " << endl;
for( size_t i= 0; i<v2.size(); i++)
{
    cout << v2[i] << " ";
}
return 0;
}
```

⇒ output :-

Before Swapping :-

Vector 1 :-

2 4 6 8 10

Vector 2 :-

1 3 5 7 9

After Swapping :-

Vector 1 :-	Vector 2 :-
1 3 5 7 9	2 4 6 8 10

Date :
Page No.

* Sorting in vectors

⇒ prog 5: sorting vectors in ascending order.

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
    vector<int> v={12,46,26,8,1};
    cout<<"Before Swapping :-"<<endl;
    for(size_t i=0;i<v.size();i++)
    {
        cout<<v[i]<<"  ";
    }
    sort(v.begin(),v.end());
    cout<<"After Swapping :->"<<endl;
    for(size_t i=0;i<v.size();i++)
    {
        cout<<v[i]<<"  ";
    }
return 0;
}
```

output :-

Before Sorting :-

12 4 6 26 8 1

After Sorting :-

1 8 12 26 46

vector:

⇒ Prog 6: Sorting is descending order:

```
sort (v.begin(), v.end(),
                greater<>());
```

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
  vector<int> v={12,46,26,8,1};
  cout<<"Before Swapping:-"<<
        endl;
  for(size_t i=0; i<v.size();i++)
  {
      cout<<v[i]<<"  ";
  }

  sort(v.begin(), v.end(), greater<>());
  cout<<"\nAfter Swapping:-"<<endl;
  for(size_t i=0; i<v.size(); i++)
  {
      cout<<v[i]<<"  ";
  }
return 0;
}
```

```
Before Swapping:-
12  46  26  8  1
After Swapping
46  26  12  8  1
```

* Searching in vector :-

binary_search(v.begin(), v.end(), 8);
↳ returns 1 if 8 is present
else
↳ returns 0 if 8 is not present.

* prog 7

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
    vector<int> v={2,4,6,8,10};
    cout<<"searching in :-"<<endl;
    for(size_t i=0; i<v.size(); i++)
    {
        cout<<v[i]<<" ";
    }
    cout<<endl;
    cout<<"Returns 1 if element found"<<binary_search(v.begin(),
        v.end(), 8)<<endl;
    cout<<"Returns 0 if element not found"<<binary_search(v.begin(),
        v.end(), 27);
return 0;
}
```

=> output :-
searching is :-
2 4 6 8 10
Returns 1 if element found : 1
Returns 0 if element not found : 0

---

* find() function
  ↳ returns index

find(v.begin(), v.end(), 8)
    - v.begin();

* prog 8 :-

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
    vector<int> v = {2, 4, 6, 8, 10};
    for(size_t i=0; i<v.size(); i++)
    {
        cout << v[i] << " ";
    }
    cout << endl;
// cout << "find() function → returns
    index of the element which is
    asked :- ";
    cout << "Index" << find(v.begin(),
        v.end(), 8) - v.begin();
}
```

=> output :-
=> Index :- 3

Date :
Page No.

* count() function
↳ count the occurence of a specific value in the container

count(v.begin(), v.end(), 8);

* prog 9

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()
{
  vector<int> v={2,4,6,8,10,8,9,
                 8,65,3,5,8};
  for(size_t i=0;i<v.size();i++)
  {
      cout<<v[i]<<" ";
  }
  cout<<endl;
  cout<<"count() function:"<<endl;
// cout<<"count the occurence of a
   specific value"<<endl;

  cout<<count(v.begin(),v.end(),8)
      <<endl;
  cout<<count(v.begin(),v.end(),27);
return 0;
}
```

=> output :-

2 4 6 8 10 8 9 8 65 3 5 8
count() function
4

↳ returns address
↳ need to do derifference.

Date :
Page No.

* max_element () function
↳ largest value within a given set of elements

```
auto maximum = max_element (v.begin(),
                            v.end());
cout << *maximum;
```

* lowerbound () function
↳ returns iterators pointing to the first element in the range [first, last] which has a value not less than val.

* upperbound () function
↳ returns iterators pointing to the first element in the range [first, last> which has value greater than val.

⇒ If no value is present then returns the end iterators.

```
auto lower = lower_bound (v.begin(),
                          v.end(), 6);
auto upper = upper_bound (v.begin(),
                          v.end(), 6);
cout << "lower_bound() of 6 : " <<
        (lower - v.begin());
cout << "upper_bound() of 6 : " <<
        (upper - v.begin());
```