⇒ Vector :-

Vector is a dynamic size array, which don't need pre-fix size. It has all advantage of array like random access and store many element, but also eleminate disadvantage of array like, it can be resized as needed, also vector class provide many function, which makes it easier to use.

advantage of vectors :-

- Dynamic size.
- Rich library function.
- Easy to know size using size().
- No need to pass size.
- Can be returned from function while array can't.
- by default initialized with dault value like 0, false or NULL.
- we can copy one vector to other.

$$v_1 = v_2.$$

→ Different ways to initialize a vector :-

- vector <data_type> name ;

ex.-

vector <int> V ;

- vector <data_type> name { a1, a2, a3 ... } ;

ex. -

vector <char> v { 'c', 'd', 'z' } ;

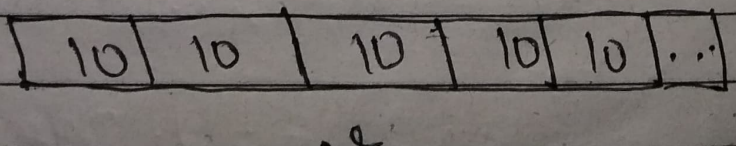
- This method is used to initialize a vector with all element with given value.

vector <data_type> v (size, value) ;

ex-

int n = 5 , x = 10 ;

vector <int> v (n, x) ;



v

- In this method we can initalize vector from other pre-initlized containers.

```
vector<data_type> v(c.begin(), c.end());
```

here, 'c' can be other container of vector or map etc.

ex.-

- ```
  int arr[] = {10, 5, 20};
  int n = sizeof(arr) / sizeof(arr[0]);

  vector<int> v(arr, arr+n);
  ```

- ```
  vector<int> v1{10, 20, 30};
  vector<int> v(v1.begin(), v1.end());
  ```

→ different ways to traverse through vector-

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

here, size() function gives size of vector v.

- since, v[i] this gives reference to that ith element we can change value at that index by
  v[i] = 10;

NOTE:- Using syntax of array to access vector element v[i] we can use v.at(i) this syntax.
at() is better than [], as at() does bound checking.

- ```
  for( int x : v)
      cout << x << " ";
  ```

here, x copies one by one element from and then print it. we can't change value in vector by doing x=10;

To change value in vector we have to write.

```
for(int &x : v){
    cout << x << " ";
        v= x+5;
}
```

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

here, it is iterator which points to the element of container. iterator can also be initalized as

vector<int>:: iterator it=v.begin()

but

auto it=v.begin() is same.

begin() function returns first (0th) element address, while end() return address of element beyond last element.

- for(auto it = v.rbegin(); it != v.rend(); it++)
  cout<<(*it)<<" ";

reverse begin()

rbegin() function returns iterator to the last elements [(n-1)th] element also, iterator is reverse which moves is opposite direction.

rend() function gives reverse iterator of element before first element.

also, iterator is reverse it++ decreases its value by 1.

→ Different function of vector class :-

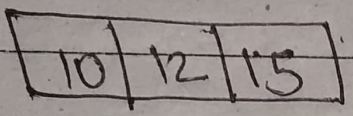
- size() :-

Gives size of vector as how many elements are in vector.

v.size();

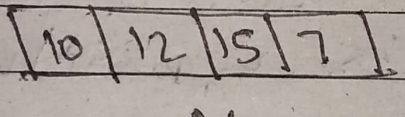
- Push_back()

Push_back() function inserts/push element from end of vector one by one.



v

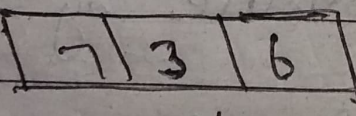
v.push_back(7);



v

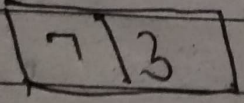
- pop_back()

opposite of push_back(). It removes last element from vector.



v

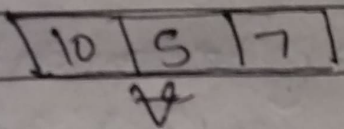
v.pop_back();



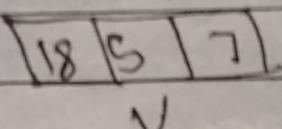
v

- front() :-

returns reference to the first element of vector. we can print, change or modify first value from it.



v.front = 18;



cout << v.front;

output :- 18

- back() :-

returns reference to the last element of the vector. Just like front().

- insert() :-

insert() is used to insert one or many element at specific position in vector.

Syntax!-

v.insert(iterator to insert position, value)

or,

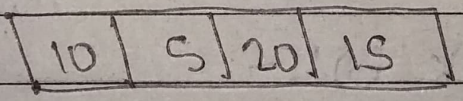
v.insert(iterator, no of times to insert, value)

another way to insert another vector to this vector at specific position.

v.insert(iterator, iterator to the starting position of 2nd vector, iterator to position after required index.)

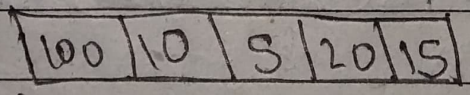
ex.-

vector<int> v {10, 5, 20, 15}



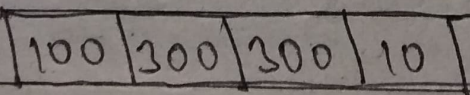
v

v.insert(v.begin(), 100);



v

v.insert(v.begin()+2, 300);



v

vector<int> v2{7,9};

v2.insert(v2.begin()+1, v.begin(), v.begin()+2);

7	100	300	300	9

v_2

upto 2nd element copied from v to v_2 at v_2.begin()+1 position.

- **erase() :-**

erase() function takes an iterator or starting and end iterator and erase all element from starting to end-1.

vector<int> v{10,5,20,15};

v.erase(v.begin());

5	20	15

v

v.insert(v.begin()+2, 2, 100);

5	20	100	100	15

v

v.erase(v.begin(), v.begin+3);

100	15

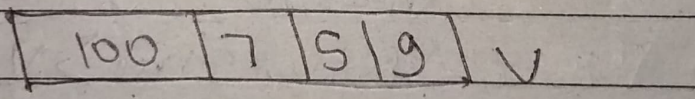
✓

It will erase all element from v.begin() to v.begin()+2 as we have given end point v.begin()+3, it will erase upto one element before that.

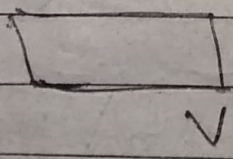
To erase all elements we can pass v.begin() and v.end(), then it will erase whole vectors.

- clear() :-

It do same work as erase to erase elements of vector but it erases all elements at once not one by one like erase selectively.



v.clear()



now, if we run v.size() we get 0 as answer.

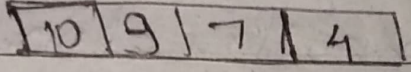
• empty():-

It checks if vector is empty or not. If empty returns true, otherwise false.

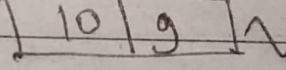
• resize():-

As name suggest, it is used to resize the array.

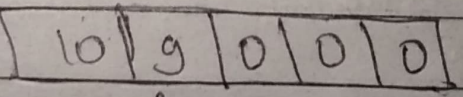
vector



v.resize(2);

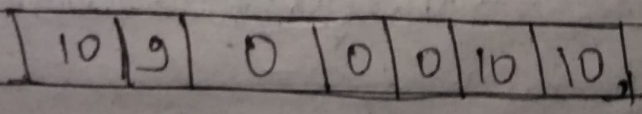


v.resize(5);



3 default values, if you want other values, you can do this.

v.resize(7,10);



→ Passing vector as function arguments:-

It is same as passing array but less more easy, we don't need to pass size. Also vectors not passed as pointers, as arrays do, they passed as whole.

So, when we pass a vector it's copy is made then function work on that copy, if any modification done on copy won't reflect on original one

To avoid making copy, we can pass vector as reference.

ex.-

normally passing - (copy is made) :-

```
void fun(vector<int> v)
{
    v.push_back(10);
    v.push_back(20);
}
int main()
{
    vector<int> v{5,7,8};
```

```
fun(v);
for(int x : v)
    cout << x << " ";

return 0;
}
```

Output :- 5 7 8

As vector is passed as copy.

reference passing (reference is pass) :-

```
void fun(vector<int> &v){
    v.push_back(10);
    v.push_back(20);
}
int main(){
    vector<int> v{5,7,8};
    fun(v);
    for(int x: v)
        cout << x << " ";
    return 0;
}
```

Output :- 5 7 8 10 20

```
struct student{
    string name;
    .
    .
    .
};

int main(){
    vector<student> v;
    student s1, s2;
    v.push_back(s1);
    v.push_back(s2);

    for(student s: v)
    {
        cout<<s.name;

    }
    return 0;
}
```

Problem with the programme is that in for loop, in s value is copied then printed or used, it is not refernce. So, we can't change it also if structure is big it will cause burden.

v.capacity()

So, it is better to use reference.

```
for(student &S:v)
{
    cout << S.name;
}
```

This problem will not occus if you are traversing with iterators.

```
for(auto it=v.begin(); it!=v.end(); it++)
{
    cout << it->name;
}
```

- reverse() :-

reverse the given vectors.

```
v.reverse();
```

- Shrink_to_fit() :-

Shrink the vectors capacity to fit the current element size.
Size becomes equal to capacity.

- capacity() :-

Returns size of storage allocate to the vector currently as number of elements.

V.capacity.