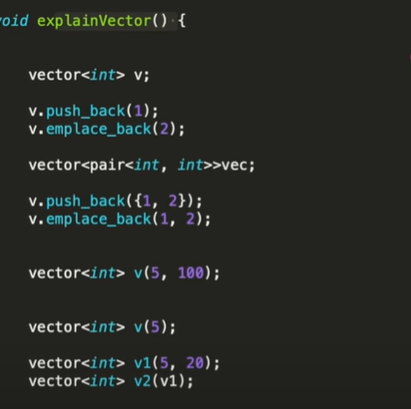
**VECTORS:**

These are dynamic arrays in which we can add, delete and do various functions on an array. It is similar to pairs.But don’t consider array and vector to be same. Only the functionality is same.

**SYNTAX:**

vectors<data\_type> variabel={ };



THE vector<int> v; is done to create an empty container { }

Now using v.push\_back(1) and v.emplace\_back(2) basically adds the element 1,2 from the back side.

Difference between push\_back and emplace\_back is that emplace is faster.also emplace\_back inserts elements directly into the vector but in case of push\_back() it creates a new vector which copies the parameter value as well as old value and places them together.

**Similarly**

vector<pair<int,int>> vec;

says that the vector has data type pair i.e it can store values in type of pair

You can also see that for the emplace\_back() function we can directly put the pairi values but in the push\_back ( { } ) we have to use the braces.

**NOW**

vector<int> v(5,100);

This means that we have created a vector of type int whose size is “”5”” AND each place has “”100”” as the default value stored in it.

( i know we have used v before but it’s just for reference don't get confused)

vector<int> v(5);

This means every value in the vector v is 5 now.

**NOW VECTORS ALSO HAS A TYPE OF INHERITANCE PROPERTY**

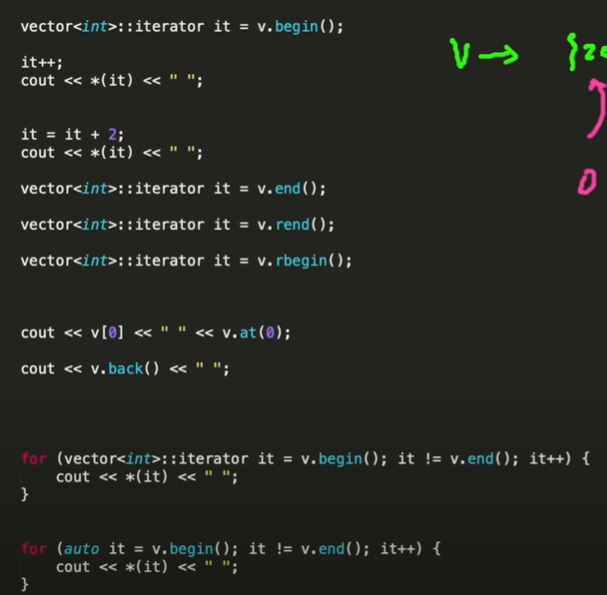
vector<int>v1(5,200);

vector<int> v2(v1);

What this does is the v1 created a vector with size 5 where all values are 200

In v2 it copies the whole v1 container.

Now remember although we declared v(5,100) size to be 5 but the vector is always dynamic so size will increase if we want it.



**ACCESSING ELEMENTS IN AN VECTOR:**

This can be done similarly to the way of array a[i]; (more preferred v[i] )

Otherwise,

We can use the built in function v.at(i);

USING ITERATOR:

This is helpful in pointing out elements in the container by using their memory address as a reference

SYNTAX:

vector<data\_type>**::** iterator variable= v.begin()/end()/rend()/rbegin();

In this the iterator points at the memory address of what is required i.e v.begin() means memory address of the first element. Since it points at the memory address to access the element we can use a pointer to get the value.

We can use **auto it=v.begin() auto is same as vector<>:: iterator**

vector<int> v;

v.push\_back(2);

v.emplace\_back(4);

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

cout<<\*it (gives 2);

It++; (we increase the size by 1)

cout<<\*it; (gives 4);

To do it for a longer range we can use for loop

for(v.begin;v.end()) as we see in the image.

begin() start point. end() last. rend() reverse end and vice versa.

Note: end() points at the last elements memory+1 location;

**INSTEAD OF SUCH HASSLE JUST USE THE FOR LOOP FOR ACCESSING**

size= v.size();

for(int i=0;i<size;i++){

cout<<v[i];

}

Does the same work as iterator without the larger syntax and no pointing at memory location directly.

**LET’S SEE HOW TO DELETE AN ELEMENT IN VECTOR:**

1. **erase():**

****

Syntax: v.erase(address of element);

If you want to delete an element at a specific position we can use the erase() operator

Example:

vector<int> v;

v.push\_back(10);

v.push\_back(20);

v.push\_back(30);

v.push\_back(40);

v.erase(v[3]);

v.erase(v.begin()+2); {10,20,40}

v.erase(v.begin()+2,v.begin()+4); {10,40}

Here we have first created an array/vector {10,20,30,40}

When we use v.erase() we have to pass a parameter inside the erase() function.

v.erase(v[2]); is invalid and will give error as erase() needs an iterator as parameter not v[ ]

v.erase(v.begin()+2); is valid as parameter is iterator and will delete element 30

v.erase(v.begin()+1,v.begin()+3); is similar to range(0,4) i.e it will star from 20 and go to 30 will include last element.

Note: After the removal of any element please resize the size of the vector if you print it with the same original size it will give an unexpected action either an error or confusion.

1. **pop\_back():**

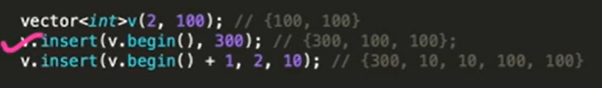
Syntax: **v.pop\_back();**

This operation removes the last element from the vector.

**NOW LET’S SEE FOR INSERTING AN ELEMENT IN THE VECTOR:**

Insert here means to add element and scient vector is a dynamic array we can do so

**Syntax: v.insert();**



As we can see we have created a vector with the size of 2 with both element 100; {100,100}

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

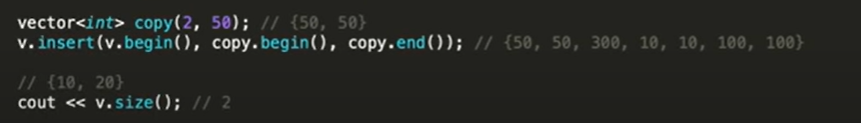
It inserts 300 at first position and pushes the rest accordingly. {300,100,100}

v.insert(v.begin()+1,2,10);

This inserts 10 from second position or index 1 where 10 is inserted 2 times.{300,10,10,100,100}

**Note:** Replace and insert are not the same.

**NOW LET’S SEE THE COPY STRUCTURE USE IN THE INSERT OPERATION.**

****

**NOW LET’S SEE FUNCTIONS LIKE SIZE,CLEAR,EMPTY:**

1. **LENGTH OF A VECTOR:**

**SYNTAX: v.size()**

**gives size of the vector**

1. **MAKE THE VECTOR TO LOOSE ALL THE VALUES IN IT:**

**SYNTAX: v.clear();**

**if the vector has certain values in it so to empty the container we can use this to do by using this way.**

1. **Empty is an boolean function to check whether the vector is empty or not**

**SYNTAX: v.empty();**