Pairs 🡪 are the storing type of data types as a part of utility class

Declare as;

Pair<int,int> arr[] or name or any other data types;

Ans can be access as;

Pair\_name.first or second and to store multiple can be uses **neseted concept**.

VECTORS

Vector is dynamic in Nature.

Vector<data\_type> name; 🡪 how to Decalare?

Vector<int> v1 (5 🡪 5 instances or memory predicted ans can be increase afterward, 20 🡪 store and convert every stored garbage value in it change to given)

Vector\_name **.push\_back()**  for input in vector;

Vector\_name.emplace\_back() also for input in vector \_\_-> it is faster than push\_back().

It is also have 0 based Indexing.

Also access as v.at() , in a Loop via int I or with a Iterator;

.begin() 🡪 for first address pointing i.e 0;

.end() 🡪 for pointing address after last element 🡪 i.e **v.size()+1**.

.size() 🡪 for finding length or size of vector.

There are reverse end , reverse begin also present but never use due to its nature;

Iterator

It is declare as –> vector<datatype> :: iterator **(iterator\_name)**  = v.begin();

This point to the memory address of the vector. (Call by refrence can change affects)

Can use ++it or it++ change increase the pointing memory address.

To get value stored at memory we use 🡪 cout<<\*(it\_name)<<endl;

STL give helps in shortening of code so it provide simpler version to print the vector;

1. For(vector<int >:: iterator it = v.begin(); it != v.end(); it++){

cout<<\*(it)<<” ”;

}

2.

For(auto it = v.begin();it!=v.end();it++){

Cout<<\*(it)<<” “;

}

3.

For(auto it: v) {

Cout<<it<<” ”;  **🡪 Simplest Version**

} auto is automatic assign the data type at iterator;

4.

For(int I =0; i<v.size();i++){

Cout<<v[i]<<” ”; **🡪 Also a type for printing the element and accessing the vector**

}

**.erase(**give iterator or v.begin() +1,+2,+3……… or basically address of the vector stored**)**

**v.erase ( v.begin()+2 , v.begin()+4) 🡪 means**  .begin point 0 ans 0+2 = point at 2nd index,

v.begin()+4 = 0+4 🡪 4th index;

here v.erase (first is Starting point **,** aftere comma we have End point and deleting the element before the end index)

V.insert ( address to start inserting , number of element to be inserted , the element itself)

Ex 🡪 vector = {300,100,200};

Now

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

* Vector = {10,10,300,100,200};

More Functions of Vector;

.pop\_back(); 🡪 Remove the top Element;

.swap(); 🡪 for swapping the elements;

.clear() ; 🡪 for erasing the Complete vector;

.empty(); 🡪 Boolean function to check is vector empty or not.

LIST

List<int> name; 🡪 Intialise

Similar as Vector ;

Here is **.push\_front()** 🡪 available to pushing element in the front of the list;

Ex, List = {5,2,20};

Now, list.push\_front(8);

List = {8,5,2,20};

Same goes as **.emplace\_front();**

**DEQUE ;**

Dequeue<int> name ; 🡪 initialise.

**STACK ;**

Stack<int> name; 🡪 initialise.

Follows LIFO 🡪 Last IN, first Out

Stack.top() 🡪 pointing to the top of the stack i.e the last element to be store

Stack.push() or emplace() 🡪 for insert. Top i.e 5

Stack.pop() 🡪 For Removing the Element 3

.empty() 🡪 to check the Stack empty or not 2

.size() 🡪 for getting the size of stack 2

1

Here is to kind of indexing present

Queue

Queue<int> name; 🡪 intialise

Here its follows FIFO 🡪 First in, First Out

Also have;

5

.push() or emplace();

6

.back() 🡪 Pointing on the Last Element;

3

.front() 🡪 Pointing on the First Element;

Second i.e 2

.pop() 🡪 Use In Removing But from First Element

First element i.e 1

And assign the first into next first element

Priority Queue

Priorty\_queue<int> name; 🡪 initialise.

.push() or emplace() for input ;

In it Data is not stored Linearly;

.top() 🡪 Top always point to the Largest/ biggest element according to the lexical table.

Ex 🡪 Inputs = {5,2,8,6,3}

Priority queue will be 🡪 {2,3,5,6,**8(top)**};

If we want to store in decreasing order ;

We initialise like-

Priority\_queue<int,vector<int>,greater<int>> name ;

Ex 🡪 Inputs = {5,2,8,6,3}

Priority queue will be 🡪 {8,6,5,3, **2(top)**};

Known as Minimum Priority Queue or Minimum Heap

Some Basic Time Complexities;

Push() 🡪 O(log N)

Top() 🡪 O(1)

Pop() 🡪 O(1)

**SET**

It stored everything in Sorted Manner and Unique, It is also stored in Non Linear fashion,

.insert() or .emplace() 🡪 for input the values.

.find(Element) 🡪 Give the Index of element in Set

\*if the element is not in set and we use Find it will redirect to the Last Element of the Set.

.erase(element or Index via iterator , after comma we have start and end and delete till the end address) 🡪 delete the given element from set and maintain the Sorted ness of set

.count(element) 🡪 find the element is present in the set or not

.lower\_bound();

.upper\_bound();

\*in this Everything Happen in Log N time Complexity

MUTLI SET

Same as set , but it can store multiple occurances as well ,

i.e example 🡪

multiset<int> name;

name 🡺 {1,1,1}

.erase 🡪 work same if .erase(1) it will erase every 1 in the multi Set.

If .erase(start, end) 🡪 same as all above.

If .erase(name.find(element)); 🡪 find the specific element and remove that

UNORDERED SET

Initialise 🡪 unordered\_set <datatype> name;

\*it store only Unique Elements and It has **Randomise order(No Sort)**.

All operation are same but lowerbound and upperbound not work in this.

Wrost case it cross 🡪 O(N) but most of time the complexity of the any function is O(1).

MAP

Initialise 🡪 map <int, int > name;

Stored has **Key – Value Pair;**

**All key must be Unique**. And stored in Sorted form like SET (**onlu Key Sorted**).

.find() 🡪 finds the Key in the Map. If not available then point to the end of the map.

Time Complexity 🡪 O(log N)

In UNORDERED MAP 🡪 O(1) && Worstly 🡪 O(N)

­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Important Algorithms to be Known: -

**# SORT;**

.sort(start, end) 🡪 start include and end is not.

In other containers, .begin() to .end();

If we use in greater<int> then it sort in decreasing order

In pairs 🡪 it Sort according to the values or Second element but If the element are same in two then it **Sort as per the first element of these conflicting element in decreasing order.**

Ex 🡪 Set = {{1,2},{2,1},{4,1}}

After Sort 🡪 Set = {{4,1},{2,1},{1,2}}

For pairs initialise as 🡪 sort(name, names+size or point at end , comp);

Comp 🡺 it is a Boolean function have two input pairs, asper

**Sort as per the first element of these conflicting element in decreasing order.**

i.e == if(p1.second<p2.second) return true;

if(p1.second>p2.second) return false;

// else – they were same – the Second or values 😊

If(p1.first>p2.first) return true;

Else return false;

**#\_\_Builtin\_popcount();**

This algo use to count all setbits means all 1 in the binary form of the inputs,

If datatype id longlong then

It will be 🡪 \_\_builtin\_popcountll();

**#Max & MIN;**

Max(num1,num2); 🡪 to get Maximum of two.

Min (num1,num2) 🡪 to get Minimum of two.

**#Next\_permutation**

It will give all permutation to be happens in the given number or string

\*its input must be in sorted form otherwise it print some less permutations.

Ex 🡪 inp = “123” ;

The Permutations will be 🡪 1st is 123 🡪 2nd - 132,

3rd - 213,

4th - 231,

5th - 312,

6th – 321.