STL C++

December 27, 2023 11:41 PM

```
#include <bits/stdc++.h>  // Includes everything
using namespace std;      //



// STRUCTURE

struct node {
   int data;
   double doub;
   string st;

   // Constructor definition
   node(int data_, double doub_, string st_) : data(data_), doub(doub_), st(st_) {}
};

int main() {
   node *a = new node(1, 1.2, "asdf"); // dynamic allocation
   node s = node(2, 3.2, "adf"); // static allocation

   // Output the values of 'a' object
   cout << "a->data: " << a->data << ", a->doub: " << a->doub << ", a->st: " << a->st
<< endl;

   // Output the values of 's' object
   cout << "s.data: " << s.data << ", s.doub: " << s.doub << ", s.st: " << s.st << endl;

   // Remember to free dynamically allocated memory
   delete a;

   return 0;
}
```

```
namespace ss {
        int val = 80;
}

int main() {
        double val = 1.2;
        cout << val << endl; // 1.2
        cout << ss::val << endl; // 80
}
```

```
// ARRAYS
int arr[100];
or
array<int, 100> arr; // if the array is declared inside main then it will have garbage values, if it is declared
globally then all the values will be 0;

arr.fill(3); // fills the entire array with 3, works only with array<int, 34> arr;

arr.at(index);

// ITERATORS
// begin(), end() , rbegin(), rend()

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

for(auto it = arr.rbegin() ; it != arr.rend() ; it++)  // it++ because iterator will automatically increment
        cout << *it << " ";
for(auto it : arr)
        cout << it << " "; // no star because it iterates the element itself

cout << arr.size();
cout << arr.front();
cout << arr.back();
cout << arr.at(index);
maximum size of array is 10 ^ 6 inside main if it is int, double, or char
mazimum size of array declared globally will be 10 ^ 7 if it is int, double or char
if bool then inside main - 10 ^ 7
if bool then globally - 10 ^ 8


// VECTOR
vector <int> v;
v.push_back(0); // pushes 0
v.pop_back() // deletes last element
v.clear(); // erases all elements
```

```
v.erase(iterator) // erase an element
v.erase(v.begin(), v.begin() + 1); // erase a range of elements
vector <int> v1(4, 2); // {2, 2, 2, 2}

// copy entire vector
vector <int> v2(v1.begin(), v1.end()) // makes a copy of v1
vector <int> v3(v1) // copies v1 to v3

vector <int> v4(v1.begin(), v1.begin() + 1); // [ )

v.emplace_back(1); // push_back and emplace_back are identical but emplace_back takes lesser time
than push_back;


swap(v1, v2)

// 2D VECTOR
vector <vector <int> > vec; // a vector vec that contains vector datatype

for(auto it : vec) {  // for each loop
        for(auto itt : it)
                cout << itt << " ";
        cout << endl;
}


// DEFINE A 10 X 20 SIZE VECTOR

vector <vector <int>> v(10, vector <int>(20, 0));
vec.push_back(vector <int> (20, 0));

// Define an array of vectors
vector <int> arr[4]; // this will create an array of size of with empty vectors;
// arr[0].emplace_back(2);
// in this case the array is not dynamic but the indices are

// 3D VECTOR OF 10 X 20 X 30
vector <vector <vector <int>>> vec(10, vector <vector <int>> vec(20, vector <int> (30, 0)));



// Given n elements, count the number of unique elements
// use set;
set<int> st;

for(int i = 0 ; i < n ; i++) {
        int x;
        cin >> x;
        st.insert(x);
}

// sets will have unique elements
// sets will store elements in ascending order
// cannot access the elements of a set by using st[i];
// to access st.begin() will give a pointer pointing to the address
// insert function takes log n time complexity, n is the size

//log n
st.erase(st.begin());
st.erase(st.begin(), st.begin() + 2); // erase the first 2 elements [ )
st.erase(3) // deletes the ELEMENT 5 from the set
auto it = st.find(7) // returns an iterator that points to the position 7

st.emplace(3) // equivalent to insert but faster

// printing is same as vector

st.erase(st.begin(), st.end()) // erase all elements

// UNORDERED SET

unordered_set<int>st;
same operations;
// the average time complexity in UN set is constant where as in ordered set it is
log
// but the worst case TC is linear in nature O(N)

try to use unordered set


// MULTISET
```

```
multiset <int> ms;
ms.insert(1);
ms.insert(1); // log n
ms.insert(2);
// all functions same as set
// it stores all the elements given to it in ordered format

there is no unordered multiset

ms.erase(ms.find(2), ms.find(2) + 2);
ms.count(3) // gives the frequency of 3


// MAP DATASET

map <string, int> mp;
mp["a"] = 1;
mp["s"] = 2;
// stores sorted according to the keys
// stores only unique keys
mp.emplace("a", 3);
mp.erase("keyname")
mp.erase(mp.begin())
mp.erase(mp.begin(), mp.end());
mp.clear();
mp.find("a") // gives a pointer to where a lies
mp.empty() // returns bool value

mp.count("a") // returns 1 if a exists

for(auto it : mp)
        cout << mp.first << mp.second; // dot because it is a pair

for (auto it = mp.begin() ; it != mp.end() ; it++)
        cout << it -> first << " " << it -> second; << endl; // -> beacuse it is a pointer

unordered_map<string, int> mp;
// O(1) in almost all cases
// O(n) in the worst case;
// a map is a container which stores pairs
// unordered_map cannot store pairs as a key or value

// MULTI MAP

multimap <string, int> mp;
// can store multiple keys with same name in ascending order


// PAIR
pair <int, int> pr = {1, 2};

pr.first = 1;
pr.second = 10;
pair <pair <int, int>, int> prr = {{1, 2}, 4};
prr.first.first
pr.first.second


// STACK

stack <int> st;
LIFO
pop
size
top // gives the top element
empty
push and emplace


no clear function
while (st.empty())
        st.pop()

if the stack is empty and if you try to access the
top, it will throw an error // runtime error


// QUEUE

queue <int> q;
FIFO
push
size
pop
empty
emplace
front

while(q.front())
        q.pop();

all are constant time operations for stack and queue
except the clear // linear since manually


// PRIORITY QUEUE
// stores all in sorted order and does the operations in log n tc

priority_queue <int> pq;

push
size
top
pop
empty
```

// concept of heap
// largest on the front max priority queue is default

min priority queue
to make a default pq to a minimum priority queue
insert as negative numbers and multiply -1 when accessing the values
cout << -1 * pq.top() << endl;

// MIN PQ
priority_queue<int, vector <int>, greater <int>> > pq;

// DEQUEUE
dequeue <int> dq;

push_front()
push_back()
pop_front()
pop_back()
begin, end, rbegin, rend
size
clear
empty
at

// LIST a doubly linked list
list <int> ls;
push_front() has emplace too
push_back()
pop_front()
pop_back()
begin, end, rbegin, rend
size
clear
empty
at
remove // list has a remove operation O(1), dq doesn't, element is given not the iterator, can also give iterator

ALWAYS USE UNORDERED MAP, BECAUSE THE TC IN ALMOST ALL CASES IS O(1), BUT IN WORST CASE O(N2), WHERE AS FOR MAP IT IS O(N LOG N)

IF TLE OCCURS FOR UNMAP SWITCH BACK TO MAP

BITSET
INT CAN TAKE UPTO 16 BITS
CHAR -> 8 BITS

bitset takes 1 bit

bitset <10> bt; // it will create a array kinda datatype that stores 0 or 1

functions
bt.all() // returns true if all the bits are set, else returns false. if any one is unset then it will return false

bt.any() // if any of the bit is set it will return true (set means 1) else false

bt.count() // print the number of set bits, or counts the number of 1s

bt.flip() // flips all the indices
bt.flip(2) // toggles the bit on position 2
bt.none() // return true if none is set else false
bt.set() // sets the entire set as 1
bt.set(3) // sets the 4th element as 1
bt.set(3, 0) // sets the 4th element as 0
bt.reset() // opposite of set
size // returns the size of bt
bt.test(3) // checks if index 3 is set or not set

ALGORITHMS
int arr[n]

```
sort(arr, arr + n); // for array
sort(arr + 1, arr + 4);
sort(v.begin(), v.end()) // for vector
sort(v.begin() + 1,  v.begin() + 4) // sort a range
```

REVERSE

```
reverse(arr, arr + n)
reverse(v.begin(), v.end())
```

MAXIMUM ELEMENT

```
*max_element(arr, arr + n);
*max_element(v.begin(), v.end());
*min_element(arr, arr + n);
```

sum of element in a range

```
accumulate(v.begin(), v.end(), 0); // 0 is the initial sum
```

// find the frequency of an element in an array	

```
count(v.begin(), v.end(), x) // counts x O(N)
```

```
// find the first occurrence of an element
auto it = find(v.begin(), v.end(), element); // it will return an iterator pointing to element
cout << "index = " << it - v.begin();
```

```
// BINARY SEARCH
WORKS ONLY ON SORTED ARRAYS
binary_search(v.begin(), v.end(), element); // returns true or false
```

```
// LOWER BOUND FUNCTION
returns an iterator pointing to the first element which is not less than x, array has to be sorted
auto it = lower_bound(v.begin(), v.end() , 4);
if there is no value then null
```

Consider the sorted array {1, 2, 2, 3, 4}:

- lower_bound(arr, arr+5, 2) returns an iterator pointing to the first 2.
- upper_bound(arr, arr+5, 2) returns an iterator pointing to the 3.

```
// NEXT PERMUTATION
bool res = next_permutation(st.begin(), st.end());
eg = "cba" :
if there is no permutation after a string then it will return false

prev permutation

bool res = prev_permutation(st.begin(), st.end());
```

```
// COMPARATOR
// are boolean functions

   q.  sort the array in descending arder

bool comp(int ele1, int ele2) {
        if(ele1 <= ele2)
                return true;
        return false;
}
sort(v.begin(), v.end(), comp());
//easy way to write a comparator
// - always consider 2 elements
// -
```

greater <int> is an inbuilt comparator
which works only if you wanna do this in descending order