A2Z DSA Course/Sheet

[Strivers A2Z DSA Course/Sheet - Crack Any FAANG or PBCs]

Pairs:-
    Pairs is a part of utility library

```
#include<bits/stdc++>
using namespace std;

//pairs
void expalinPair () {
          Pair<int , int> P = {2,3};
          cout<<P.first<<P.second<<endl;
          pair<int, pair<int, int>> Q = {1,{2,3}};
          cout<<Q.first<<Q.second,first<<Q.second.second<<endl;
          //as many you wanted go with nested pairs

//Know that you can implement pair array
          pair<int,int>arr[] = {{1,2},{3,4},{5,6},{7,8}};
//accessing array pair elements
          cout<<arr[1].second<<endl;
//output will be {{1,2},{3,4},{5,6},{7,8}}


}
Int main() {
}
```

Vectors:

This will be similar to all the containers
Vector is a container which is dynamic in size you can always increase the size Dynamically
Best place to use vector whenever you don't know the exact size of the array.

Syntax:
vector<int> v; //creates empty container { }
//add elements;
v.push_back(1); // {1}
v.emplace_back(2); // { 1,2 }

```
void explainVector() {
   vector<int> v;         //creates empty container { }
   v.push_back(1);        //adds element { 1 }
   v.emplace_back(2);         //faster than push_back  { 1,2 }
//Note vector can be of pair also  change data type declaration into pair
   vector<pair<int,int>>pvert;
//imp as like adding elements in vector your should use
push_back(inside curlybraces{1,2});
   v.push_back({1,2});
   v.emplace_back(3,4);  //curly braces is not needed it automatically considers and stores
the values
```

```
//container of elements with its size
// {100,100,100,100,100}
vector<int>v(5,100);
vector<int> v1 (5,20);  {20,20,20,20,20}
vector<int > v2(v1);  //{20,20,20,20,20} similar but different container not the same v1 cont
```

Access elements in vector

There are two types to access one is like normal array accessing another one is ITERATOR

Type 1:

vector<int> v = {5,200};   //200,200,200,200,200

cout<<v[0]<<" "<<v.at(0); //generally (.at) is not used

Type 2:

ITERATOR

Complete C++ STL in 1 Video | Time Complexity and Notes

v1 = | 20 | 10 | 15 | 6 | 7 |

vector<int>::iterator v1 = v.begin();

Begin points outs the initial address not the value…

O/P : 20

vector <int>::iterator vec = begin();
vec++;
cout<<*(vec)<<endl;

O/P : | 10 |

vec+=2;
cout<<*(vec)<<endl;
O/P : | 6 | //already 10 + two address next **6** occurs

Types of ITERATORS

begin();
end();
rend();
rbegin();
example:
| 20 | 10 | 15 | 6 | 7 |
vector<int>::iterator it = v.begin(); // 20
vector<int>::iterator it = v.end();   //
vector<int>::iterator it = v.rend();   //never ever used
vector<int>::iterator it = v.rbegin(); //never ever used reverse begin
i++implemented in reverse order
cout<<v[0]<<" "<<v.at(0)<<endl;
cout<<v.back()<<" "; // | 7 |
//Printing al elements;

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

//using foreach loop

```
for(auto it : v) {
   cout<< it << " ";
}
```

Erase in vector (deletion):

Swap in vector:

//v1 -> {10,20};
//v2 -> {30,40};
v1.swap(v2); //v1{30,40} v2{10,20};
v.clear();
cout<<v.empty();

LIST -container

List is similar to vector only thing differs is it provides front operations as well
Code:

```
void explainList() {
   list<int> ls;    // {}
   ls.push_back(2);   // {2}
   ls.emplace_back(4);  // {2,4}

   ls.push_front(5);   // { 5, 2, 4 }  notice here added at front id list
   ls.emplace_front(): //  {2, 4}
   // rest all other functions are same as vectors
   //begin, end, rbegin, rend, clear, insert, size, swap
}
```

Deque: container similar to list and vector

```
void explaindeque() {
   deque<int> dq;  // {}
   dq.push_back(1);  //  {1}
   dq.emplace_back(2):  // {1,2}
   dq.push_front(4);  // {4, 1, 2}
   dq.emplace_front(5);  // {5, 4, 1, 2}
   dq.pop_back();  // {5, 4, 1}
   dq.pop_front();  //{4, 1}
   dq.back();
   dq.front();
   // rest all other functions are same as vectors
   //begin, end, rbegin, rend, clear, insert, size, swap
}
```

Pattern printing

1) Function of outer loop, is to focus on no of lines

2) Function of inner loop is, focus on columns connect somehow with row
3) Print anything inside the inner for loop
4) Observe symmetry (optional)

Exercises

```
#include <iostream>

using namespace std;

int main()
{
   int n = 5;
   for(int i = 0 ; i < n ; i++) {
      for(int j = 0 ; j <= n ; j++) {
         cout<<"*";
      }cout<<"\n";
   }

   return 0;
}

O/P:
*****
*****
*****
*****
*****
```

```
#include <iostream>

using namespace std;

int main()
{
   int n = 5;
   for(int i = 0 ; i < n ; i++) {
      for(int j = 0 ; j <= i ; j++) {
         cout<<"*";
      }cout<<"\n";
   }

   return 0;
}
O/P:
*
**
***
****
*****
```

```
#include <iostream>

using namespace std;

int main()
{
   int n = 5;
   for(int i = n ; i >0 ; i--) {
      for(int j = i ; j >0 ; j--) {
         cout<<"*";
      }cout<<"\n";
   }

   return 0;
}        O/P:
         *****
         ****
         ***
         **
         *
```