C++ Data Structures Detailed Guide

1. Array

- **Meaning:** Fixed-size collection of elements of the same type stored contiguously.
- **Syntax:**

```cpp
int arr[5];       // array of 5 integers
int arr2[] = {1,2,3,4,5}; // initialized array
```

- **Important terms:**
  - `arr[i]` → access element at index `i`
  - `sizeof(arr)/sizeof(arr[0])` → size of array
- **Example & Printing:**

```cpp
int arr[] = {10,20,30,40};
int n = sizeof(arr)/sizeof(arr[0]);
for(int i=0;i<n;i++)
    cout << arr[i] << " ";
```

2. Vector

- **Meaning:** Dynamic array; can grow or shrink automatically.
- **Syntax:**

```cpp
#include <vector>
vector<int> v;          // empty vector
vector<int> v2 = {1,2,3};    // initialized vector
```

- **Important functions:**

Function	Meaning
`push_back(x)`	Add element at the end
`pop_back()`	Remove last element
`size()`	Number of elements
`empty()`	Check if vector is empty
`insert(pos, val)`	Insert `val` at `pos` (iterator or index)
`erase(pos)`	Remove element at position
`clear()`	Remove all elements
`front()`	Access first element
`back()`	Access last element

- **Example:**

```cpp
vector<int> v;
v.push_back(10);
v.push_back(20);
v.push_back(30);
v.pop_back(); // removes 30
```

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

---

3. String

- **Meaning:** Sequence of characters; can be dynamically resized.
- **Syntax:**

```cpp
#include <string>
string s = "Hello";
```

- **Important functions:**

Function	Meaning
`length()` / `size()`	Get string length
`empty()`	Check if empty
`append(str)`	Add string at end
`substr(pos,len)`	Substring from `pos` with `len`
`erase(pos,len)`	Remove part of string
`find(str)`	Find substring
`at(i)`	Access character at index `i`

- **Example:**

```cpp
string s = "Hello";
s.append(" World"); // "Hello World"
for(char c : s) cout << c << " "; // prints each character
```

---

4. Linked List (STL list)

- **Meaning:** Doubly-linked list (nodes with prev & next).
- **Syntax:**

```cpp
#include <list>
list<int> l;
```

- **Functions:**

Function	Meaning
`push_back(x)`	Add at end
`push_front(x)`	Add at start
`pop_back()`	Remove last element
`pop_front()`	Remove first element
`insert(pos, val)`	Insert at iterator position
`erase(pos)`	Remove element at iterator position
`size()`	Number of elements
`empty()`	Check empty

- **Example & Printing:**

```cpp
list<int> l = {1,2,3};
l.push_back(4);
l.push_front(0);
for(int x : l) cout << x << " ";
```

---

5. Stack

- **Meaning:** LIFO (Last In First Out).

- **Syntax:**

```cpp
#include <stack>
stack<int> st;
```

- **Functions:**

Function	Meaning
`push(x)`	Add element at top
`pop()`	Remove top element
`top()`	Access top element
`size()`	Number of elements
`empty()`	Check empty

- **Example & Printing:**

```cpp
stack<int> st;
st.push(10); st.push(20);
cout << st.top() << endl; // 20
while(!st.empty()) {
    cout << st.top() << " ";
    st.pop();
}
```

---

6. Queue

- **Meaning:** FIFO (First In First Out).
- **Syntax:**

```cpp
#include <queue>
queue<int> q;
```

- **Functions:**

Function	Meaning
`push(x)`	Add element at back
`pop()`	Remove element from front
`front()`	Access first element
`back()`	Access last element
`size()`	Number of elements
`empty()`	Check empty

- **Example & Printing:**

```cpp
queue<int> q;
q.push(1); q.push(2);
cout << q.front() << endl; // 1
while(!q.empty()) {
    cout << q.front() << " ";
    q.pop();
}
```

---

7. Priority Queue

- **Meaning:** Queue with priority; largest (max-heap) or smallest (min-heap) element always on top.
- **Syntax:**

```cpp
#include <queue>
priority_queue<int> pq; // max-heap
priority_queue<int, vector<int>, greater<int>> pq_min; // min-heap
```

- **Functions:** Same as queue (`push` , `pop` , `top` , `size` , `empty`)
- **Example & Printing:**

```cpp
priority_queue<int> pq;
pq.push(30); pq.push(10); pq.push(20);
while(!pq.empty()) {
    cout << pq.top() << " "; // 30 20 10
    pq.pop();
}
```

---

8. Deque

- **Meaning:** Double-ended queue; can add/remove from both ends.
- **Syntax:**

```cpp
#include <deque>
deque<int> d;
```

- **Functions:**

Function	Meaning
`push_back(x)`	Add at end
`push_front(x)`	Add at front
`pop_back()`	Remove last
`pop_front()`	Remove first
`front()`, `back()`	Access first/last
`size()`, `empty()`	Standard

- **Example:**

```cpp
deque<int> d = {1,2,3};
d.push_front(0); d.push_back(4);
for(int x : d) cout << x << " ";
```

---

9. Set / Multiset

- **Meaning:** Unique elements (sorted).
- **Syntax:**

```cpp
#include <set>
set<int> s;
multiset<int> ms; // allows duplicates
```

- **Functions:** `insert` , `erase` , `find` , `count` , `size` , `empty`
- **Example:**

```cpp
set<int> s = {3,1,4};
s.insert(2);
for(int x : s) cout << x << " "; // prints 1 2 3 4
```

---

10. Map / Multimap

- **Meaning:** Key-value pairs; sorted by key.
- **Syntax:**

```cpp
#include <map>
map<int,string> m;
multimap<int,string> mm;
```

- **Functions:** `insert` , `erase` , `find` , `count` , `size` , `empty` , access by `m[key]`
- **Example:**

```cpp
map<int,string> m;
m[1] = "One"; m[2] = "Two";
for(auto p : m) cout << p.first << "->" << p.second << " ";
```

---

11. Unordered Set / Map

- **Meaning:** Like set/map but **not sorted**, uses hash table.
- **Syntax:**

```cpp
#include <unordered_map>
unordered_map<string,int> um;
unordered_set<int> us;
```

- **Example:**

```cpp
unordered_map<string,int> um;
um["apple"]=2; um["banana"]=3;
for(auto p : um) cout << p.first << ":" << p.second << " ";
```

---

12. Bitset

- **Meaning:** Fixed-size sequence of bits; efficient for bit operations.
- **Syntax:**

```cpp
#include <bitset>
bitset<8> b(13); // 00001101
```

- **Functions:** `set`, `reset`, `flip`, `test`, `count`, `size`
- **Example:**

```cpp
bitset<8> b(13);
cout << b << endl; // prints 00001101
```

---

13. Pair

- **Meaning:** Store two values together (can be different types).
- **Syntax:**

```cpp
```

```cpp
#include <utility>
pair<int,string> p = {1,"Hello"};
```

- **Access:** `p.first`, `p.second`
- **Example:**

```cpp
cout << p.first << " " << p.second << endl; // 1 Hello
```

---

14. `auto` keyword

- **Meaning:** Automatically deduces variable type from initializer.
- **Example:**

```cpp
map<int,string> m;
m[1]="One"; m[2]="Two";
for(auto p : m) cout << p.first << "->" << p.second << " ";
```

- Saves time instead of writing `pair<const int,string>`.
