

DATA STRUCTURES IN C++

{Quick Recap}

Data Structure Decision Diagram

- The following diagram gives you the direction to which data structure to use in C++ according to the problem you are trying to solve



Note: I don't have the source of this diagram. If you know it, please drop me a msg so I can add it here.

Arrays

- Fixed-size collection of elements of the same type
- Stored in **contiguous memory**
- Declared with syntax: **type arrayName[size]**

Example:

```
int numbers[5]
```

- Can also be initialized at declaration:

```
int arr[3] = {1 ,2, 3}
```

- Cannot resize after declaration
- Size can be calculated by **sizeof(arr) / sizeof(arr[0])**
- stdlib provides **std::array<type, size>**
- Includes header: **#include <array>**

Example:

```
std::array<int, 3> a = {1, 2, 3};
```

Arrays (vectors)

- `std::vector` is a sequence container that encapsulates dynamic sized arrays*
- Stored in **contiguous memory** (like arrays)
- Declared with syntax: **`std::vector<type> vectorName`**

Example:

```
std::vector<int> numbers
```

- Can be initialized at declaration:

```
std::vector<int> vec = {1, 2, 3}
```

- **Can resize dynamically** during runtime
- Size accessed with **`.size()`** method
- Includes header: **`#include <vector>`**

Example:

```
std::vector<int> v = {1, 2, 3};
```

```
v.push_back(4);
```

```
v.pop_back();
```

Linked List

- Dynamic collection of elements connected by pointers (implemented as doubly-linked list in C++)
- Stored in **non-contiguous memory** locations
- Declared with syntax: **std::list<type> listName**

Example:

```
std::list<int> numbers
```

- Can be initialized at declaration:

```
std::list<int> lst = {1, 2, 3}
```

- **Can resize dynamically** during runtime
- Size accessed with **.size()** method
- Includes header **#include <list>**

- **Example:**

```
std::list<int> l = {1, 2, 3};  
l.push_front(0);
```

- **LIFO** (Last In, First Out) data structure
- Elements added and removed from the **top** only
- Declared with syntax:

`std::stack<type> stackName`

Example:

```
std::stack<int> numbers
```

- Cannot be initialized with list at declaration
- **Can resize dynamically** during runtime
- Size accessed with **`.size()`** method
- Includes header **`#include <stack>`**
- **Example:**

```
std::stack<int> s;
```

```
s.push(1);
```

```
s.pop();
```

Queue

- **FIFO** (First In, First Out) data structure
- Elements added at **back** and removed from **front**
- Declared with syntax: **std::queue<type> queueName**

- **Example:**

```
std::queue<int> numbers
```

- Cannot be initialized with list at declaration
- **Can resize dynamically** during runtime
- Size accessed with **.size()** method
- Includes header **#include <queue>**

- **Example:**

```
std::queue<int> q;  
q.push(1);  
q.pop();
```

Heap

- **Priority queue** data structure (max-heap by default)
- Elements automatically ordered by **priority/value**
- Declared with syntax:

`std::priority_queue<type> heapName`

- **Example:**

```
std::priority_queue<int> numbers
```

- Can be initialized from container: `std::priority_queue<int> pq(vec.begin(), vec.end())`
- **Can resize dynamically** during runtime
- Size accessed with **`.size()`** method
- Includes header **`#include <queue>`**

- **Example:**

```
std::priority_queue<int> h;
```

```
h.push(3); h.push(1);
```

```
h.top(); // returns 3
```


Hash Table

- **Key-value pairs** with fast $O(1)$ average lookup
- Uses **hash function** to map keys to indices
- Declared with syntax: **`std::unordered_map<keyType, valueType> mapName`**
- **Example:**
`std::unordered_map<string, int> ages`
- Can be initialized at declaration: `std::unordered_map<string, int> map = {{"key", 1}}`
- **Can resize dynamically** during runtime
- Size accessed with **`.size()`** method
- Includes header **`#include <unordered_map>`**
- **Example:**
`std::unordered_map<string, int> m;`
`m["alice"] = 25;`
`m.at("alice");`

Tree

- **Hierarchical** data structure with nodes and edges
- Each node has **parent-child relationships** (except root)
- No built-in tree class - typically **implemented manually**

- **Example:**

```
struct TreeNode {  
    int val;  
    TreeNode* left;  
    TreeNode* right; };
```

- Cannot be initialized with simple syntax
- **Can resize dynamically** by adding/removing nodes
- Size calculated by **traversing all nodes**
- No standard header required (custom implementation)
- **Example:**

```
TreeNode* root = new TreeNode(1);  
root->left = new TreeNode(2);
```