Understanding std::string Memory Allocation in C++

1. Case: string arr = "Nikhil";

- The variable 'arr' is declared as a local std::string object.
- 'arr' itself is stored on the stack.
- Internally, std::string dynamically allocates memory on the heap to store the characters of the string.
- So, while the object lives on the stack, the actual data ('N', 'i', etc.) is on the heap.

2. Case: string* arr = new string[n];

- A pointer 'arr' is declared on the stack.
- new string[n] creates an array of n string objects on the heap.
- Each string object inside the array also manages its internal data (characters) on the heap.
- You must call delete[] arr to release this memory.

3. Stack vs Heap Comparison

Declaration	Variable Location	String Data Location
string s = "hi";	s on Stack	chars on Heap (managed)
string* s = new string;	s on Stack	object on Heap
string* arr = new string[n];	arr on Stack	n objects on Heap

4. std::string Internals

- std::string contains a char* pointer, length, and capacity internally.
- When a string is assigned a value, it allocates heap memory and manages it internally.
- This allows features like dynamic resizing, copy-on-write (older implementations), etc.

5. Summary

- You can use std::string with heap or stack memory freely.
- In templates like Stack<T>, string works perfectly since it's a proper C++ object.
- Heap allocations give dynamic size, stack variables give fast access.

Understanding std::string Memory Allocation in C++

- Always manage heap memory manually if you use 'new'.