Memory Management and Pointers in C++

1. Dynamic Memory Allocation

- C++ provides dynamic memory allocation using `new` and `delete`. Dynamic memory is allocated on the heap, allowing memory to persist beyond the function's scope.

- - `new` allocates a single object: `int* ptr = new int;`

- - `new[]` allocates an array: `int* arr = new int[10];`

- - `delete` deallocates a single object: `delete ptr;`

- - `delete[]` deallocates an array: `delete[] arr;`

- Important: Always match `new` with `delete` and `new[]` with `delete[]` to prevent memory leaks.

2. Smart Pointers

- Smart pointers automatically manage memory, preventing memory leaks by releasing memory when it's no longer in use.

- Types of Smart Pointers in C++:

- - `std::unique_ptr`: Exclusive ownership. Only one `unique_ptr` can own a resource at a time. Automatically deletes the object when it goes out of scope.

- Example: `std::unique_ptr<int> p1 = std::make_unique<int>(10);`

- - `std::shared_ptr`: Shared ownership. Multiple `shared_ptr`s can point to the same resource. The object is deleted when the last `shared_ptr` goes out of scope.

- Example: `std::shared_ptr<int> p2 = std::make_shared<int>(20);`

- - `std::weak_ptr`: Non-owning pointer that works with `shared_ptr`. Prevents circular references, as it does not increase the reference count.

- Example: `std::weak_ptr<int> p3 = p2;`

3. Pointer Arithmetic

- Pointer arithmetic is essential for working with arrays, as pointers allow element access by shifting the address location.

- - Increment (`ptr++`): Moves the pointer to the next memory location based on the data type's size.

- - Decrement (`ptr--`): Moves the pointer to the previous memory location.

- - Addition/Subtraction: You can add or subtract integers to move the pointer by several locations.

- Example: `int arr[5] = {1, 2, 3, 4, 5}; int* ptr = arr; int second = *(ptr + 1); // Access second element`

- Warning: Ensure pointers do not go out of bounds to avoid undefined behavior.

4. Raw Pointers vs. Smart Pointers

- - **Raw Pointers**: Require manual memory management. You must `delete` allocated memory to avoid leaks.

- - **Smart Pointers**: Automatically manage memory, reducing the risk of memory leaks. Use smart pointers wherever possible for safety and maintainability.

5. Memory Leaks and Common Issues

- - **Memory Leak**: Happens when dynamically allocated memory is not deallocated, causing reduced available memory.

- - **Dangling Pointer**: Occurs when a pointer is used after the memory it points to has been freed.

- Prevent memory issues by always deallocating dynamically allocated memory and using smart pointers when feasible.