Memory Allocation in C++ Cheat Sheet with Code Examples

Key Concepts to Remember

Stack vs. Heap:

- Stack: Automatically managed memory for local variables with limited size. Fast allocation/deallocation.

- Heap: Manually managed memory for dynamic allocation, larger and flexible but slower.

Pointers:

- Pointers store memory addresses and are essential for dynamic memory management.

- Always initialize pointers to nullptr when declaring them to prevent accidental use.

Memory Allocation Functions

new / delete:

- new: Allocates memory on the heap, returns pointer to allocated memory.

- delete: Frees memory allocated by new, must match the corresponding allocation.

`Code Example:`

```
int* ptr = new int(5); // Allocates and initializes an integer
delete ptr;            // Frees the memory allocated
```

new[] / delete[]:

- new[]: Allocates an array on the heap.

- delete[]: Frees memory allocated for arrays.

`Code Example:`

```
int* arr = new int[10]; // Allocates array of integers
delete[] arr;           // Frees the allocated array
```

malloc / free:

- malloc: Allocates raw memory, returns void pointer (needs typecasting).

- free: Frees memory allocated by malloc.

Code Example:

```
int* ptr = (int*)malloc(sizeof(int) * 10); // Allocates space for 10 integers
free(ptr);                                  // Frees the memory allocated
```

Advanced Memory Management

Placement new:

- Allocates an object at a specified memory location.

Code Example:

```
char buffer[sizeof(int)];
int* num = new (buffer) int(42); // Places 42 in buffer memory
```

Custom Allocators:

- Implement custom allocation strategies by overriding global new and delete operators.

- Useful for memory pooling and performance-critical applications.

Code Example (Custom new/delete):

```
void* operator new(size_t size) { return malloc(size); }
void operator delete(void* ptr) { free(ptr); }
```

Memory Management Best Practices

- Always pair new with delete, new[] with delete[], malloc with free.

- Use smart pointers (std::unique_ptr, std::shared_ptr) to avoid manual memory management.

- Avoid memory leaks by ensuring all dynamically allocated memory is freed.

- Prefer RAII (Resource Acquisition Is Initialization) for automatic memory management.

Code Example (Smart Pointer):

```
#include <memory>
```

```
std::unique_ptr<int> ptr = std::make_unique<int>(10); // Smart pointer for automatic memory management
```

Memory Allocation Rules - 3-5 and Zero Rules

3-5 Rule:

- If you allocate memory for three things, you should ideally allocate memory for five things.

- Example: Instead of allocating one object at a time, allocate in chunks to reduce fragmentation.

Code Example:

```
int* batch = new int[5]; // Allocating a batch of 5 for less frequent allocations
```

Zero Rule:

- Always set pointers to nullptr after deletion or deallocation to avoid dangling pointers.

- Initialize all dynamic memory to zero if needed for safety.

Code Example:

```
int* ptr = new int(0);     // Initialize to zero
delete ptr; ptr = nullptr; // Set pointer to nullptr after delete
```