@Static vs Dynamic Allocation

**Dynamic Memory Allocation** in Object-Oriented Programming (OOP) allows you to allocate memory for objects at runtime, rather than at compile-time. In C++, this is particularly important when you don't know the size or number of objects you need to create ahead of time.

C++ provides operators such as new and delete to allocate and deallocate memory dynamically for objects.

**Key Concepts:**

1. **new Operator**: Allocates memory for an object or array of objects during runtime and returns a pointer to the newly allocated memory.
2. **delete Operator**: Deallocates memory that was previously allocated by new to avoid memory leaks.

535 837 
Static vs Dynamic Memory AllocatiorÜ4Z 
o o He 
100M 
10. 
1009 
hew C4uden+ 

**Basic Example of Dynamic Memory Allocation for an Object:**

#include <iostream>

class Car {

public:

int speed;

// Constructor

Car(int s) : speed(s) {

std::cout << "Car created with speed: " << speed << std::endl;

}

// Destructor

~Car() {

std::cout << "Car with speed " << speed << " is being destroyed." << std::endl;

}

};

int main() {

// Dynamically allocate memory for an object of type Car

Car\* myCar = new Car(120); // 'new' allocates memory dynamically for the object

// Accessing the object

std::cout << "Car speed: " << myCar->speed << std::endl;

// Deallocating memory

delete myCar; // 'delete' frees the memory

return 0;

}

Output:

Car created with speed: 120

Car speed: 120

Car with speed 120 is being destroyed.

**How It Works:**

* Car\* myCar = new Car(120);: This line dynamically allocates memory for an object of type Car and initializes it with the constructor (Car(120)). It returns a pointer myCar to the allocated memory.
* delete myCar;: This releases the memory occupied by the object. Failing to call delete after dynamic memory allocation leads to **memory leaks**.

Dynamic Memory Allocation for an Array of Objects:

You can also dynamically allocate memory for an array of objects using new[] and deallocate it with delete[]

#include <iostream>

class Car {

public:

int speed;

// Constructor

Car(int s) : speed(s) {

std::cout << "Car created with speed: " << speed << std::endl;

}

// Destructor

~Car() {

std::cout << "Car with speed " << speed << " is being destroyed." << std::endl;

}

};

int main() {

// Dynamically allocating an array of Car objects

Car\* cars = new Car[3] { {100}, {120}, {140} }; // Using an initializer list

// Accessing the objects

for (int i = 0; i < 3; ++i) {

std::cout << "Car " << i+1 << " speed: " << cars[i].speed << std::endl;

}

// Deallocating the array of objects

delete[] cars; // Using delete[] to free array memory

return 0;

}

Output:

csharp

Copy code

Car created with speed: 100

Car created with speed: 120

Car created with speed: 140

Car 1 speed: 100

Car 2 speed: 120

Car 3 speed: 140

Car with speed 100 is being destroyed.

Car with speed 120 is being destroyed.

Car with speed 140 is being destroyed.

Explanation:

new[]: Allocates memory for an array of objects and calls the constructor for each object.

delete[]: Deallocates the memory for the array of objects and calls the destructor for each object in the array.

**Memory Leaks:**

If you fail to use delete or delete[] to deallocate memory allocated with new or new[], you end up with **memory leaks**—memory that is no longer used by the program but hasn't been returned to the system

**Advantages of Dynamic Memory Allocation in OOP:**

1. **Flexibility**: You can allocate memory at runtime based on actual needs.
2. **Efficiency**: You only use the amount of memory required at a specific time, reducing wastage.
3. **Handling Large Objects**: Allows for the creation of large arrays or objects that cannot fit into the stack (local memory).

-----------------------------------------------------------------------------------------------------------------

s. 
NO 
1 
2 
3 
4 
5 
6 
7 
8 
9 
Static Memory Allocation 
In the static memory allocation, variables get 
allocated permanently, till the program 
executes or function call finishes. 
Static Memory Allocation is done before 
program execution. 
stack for managing the static allocation 
Of memory 
It is less efficient 
In Static Memory Allocation, there is no memory 
re-usability 
In static memory allocation. once the memory is 
allocated, the memory size can not change. 
In this memory allocation scheme, we cannot 
reuse the unused memory. 
In this memory allocation scheme, execution is 
faster than dynamic memory allocation. 
In this memory is allocated at compile time. 
In this allocated memory remains from start to 
10 
end of the program. 
Example: This static memory allocation is 
11 
generally used for array. 
Dynamic Memory Allocation 
In the Dynamic memory allocation, the memory is controlled by the 
programmer. It gets allocated whenever a malloc() is executed gets 
deallocated wherever the free() is executed. 
Dynamic Memory Allocation is done during program execution. 
It uses heap (not heap data structure) of memory for managing the 
dynamic allocation Of memory 
It is more efficient 
In Dynamic Memory Allocation, there is memory re-usability and 
memory can be freed when not required 
In dynamic memory allocation, when memory is allocated the 
memory size Can be changed. 
This allows reusing the memory. The user can allocate more 
memory when required. Also, the user can release the memory 
when the user needs it. 
In this memory allocation scheme. execution is slower than static 
memory allocation. 
In this memory is allocated at run time. 
In this allocated memory can be released at any time during the 
program. 
Example: This dynamic memory allocation is generally used for 
linked list. 