Dynamic Memory Allocation in C++

Memory is divided into two parts:

1. Stack
2. Heap

1. In stack, all the variables declared inside the function take up memory from the stack.

2. Heap is unused memory of the program and used for allocating the memory dynamically when program runs.

Difference between Static Memory and Dynamic Memory

Following are the differences between Static Memory Allocation and Dynamic Memory Allocation:

Static Memory Allocation	Dynamic Memory Allocation
In static memory allocation, memory is allocated **before the execution** of the program begins.	In Dynamic memory allocation, memory is allocated **during the execution** of the program.
Memory allocation and deallocation actions are not	Memory allocation and deallocation actions are

It performs execution of program faster than dynamic memory.	It performs execution of program slower than static.
It requires more memory space.	It requires less memory space.
The data in static memory is allocated permanently.	The data in dynamic memory is allocated only when program unit is active.

New Operator

- New operator is used to dynamically allocate the memory.
- The **new** keyword is used to allocate the memory space dynamically followed by a data type specifier.

Syntax:

```
new int;        //dynamically allocates an integer
new float;      //dynamically allocates an float
```

- For creating an array dynamically, use the same form but put the square brackets ([]) with a size after the data type.

Syntax:

```
new int[10];          //dynamically allocates an array of 10 integers.
new float[10];    //dynamically allocates an array of 10 floats.
```

- In the above syntax, the allocated spaces have no names, but the new operator returns the starting

```
int *ptr;              //declare a pointer ptr
ptr = new int;    //dynamically allocate an int and
load address into ptr
```

Delete Operator

- Delete operator is used to deallocate the memory.
- This operator deallocates the memory previously allocated by the new operator.
- If the memory allocated dynamically to a variable is not required anymore, you can free up the memory with delete operator.

Syntax:

```
delete variable_name;
```

Example:

```
delete ptr;    //Releases memory pointed to by ptr
```

- Using delete operator the memory becomes available again for other requests of dynamic memory.

Example : Demonstrating how new & delete operators work

```
#include <iostream>
using namespace std;
int main ()
{
    int *ptr  = NULL;        // Pointer initialized with null
    ptr  = new int;          // Request memory for the
variable
    *ptr = 12345;         // Store value at allocated
address
```

Output:

Value of Pointer Variable *ptr : 12345

Dynamic Memory Allocation for Objects

Example : Demonstrating the dynamic memory allocation using an array of objects

```
#include <iostream>
using namespace std;
class Test
{
    public:
      Test()
      {
         cout << "Constructor called..." <<endl;
      }
      ~Test()
      {
         cout << "Destructor called!!" <<endl;
      }
};
int main( )
{
    Test* t = new Test[2];
    delete [] t;        // Delete array
    return 0;
}
```

Destructor called!!

In the above example, to allocate an array of two **Test** objects, the simple constructor would be called two times and similarly while deleting these objects, the destructor will also be called same number of times.

« Previous Next »