Dynamic Memory Management

The process of allocating or de-allocating a block of memory during the execution of a program is called Dynamic Memory Allocation. The operators **new** and **delete** are utilized for dynamic memory allocation in C++ language, new operator is used to allocate a memory block, and **delete** operator is used to de-allocate a memory block which is allocated by using **new** operator.

C++ Program Memory Management

When we run a C++ program on our machine, it requires some space to store its instructions (statements), local variables, global variables, and various other functions in C++. This space required to run a C++ Program is known as memory in computers.

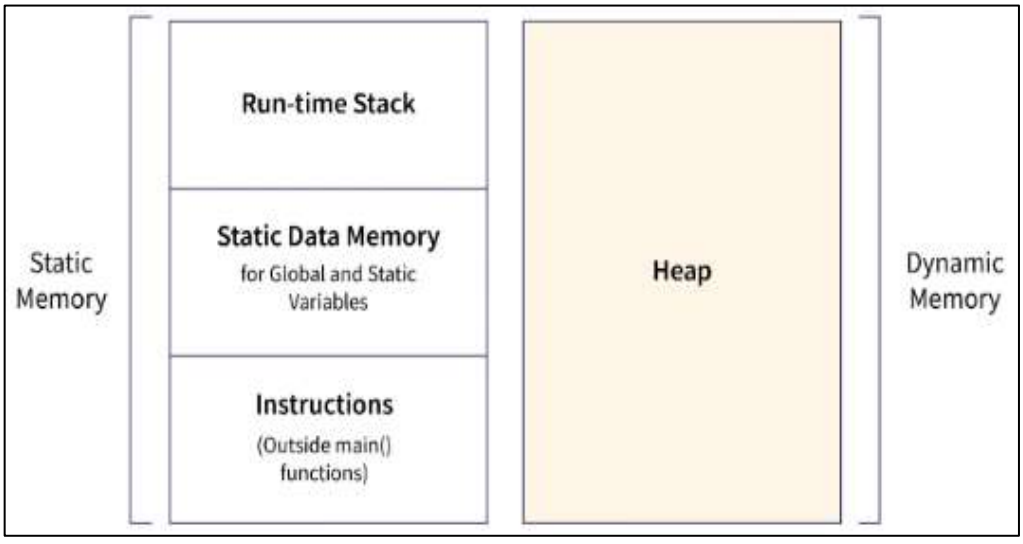
There are two types of memory in our system, **Static Memory** and **Dynamic Memory.**

1. **Static Memory:** It is a constant space allocated by the operating system during the compile time of a C++ program and it internally uses stack data structure to manage the static memory allocation. We can't reallocate the space consumed by the program until its execution is over.

2. **Dynamic Memory:** It is the memory that can be allocated or de-allocated by the operating system during the run-time of a C++ program. It is more efficient than static memory because we can de-allocate and reuse our memory during the run-time of our program.

The memory used by a C++ program can be divided further into four parts:

- Run-time Stack (Static Memory)
- Static Data Memory (for Global and Static Variables)
- Instructions / Statements (Static Memory)
- Heap (Dynamic Memory)



The Stack memory

- Our operating system allocates a constant space during compile-time of a C++ program, this space is known as Stack memory.
- Stack memory is used to hold functions, different variables, and local statements that exist in the function definition.
- Stack is a part of the static memory in our system and it constitutes the majority of our system's static memory.

The Heap Memory

- Heap memory is also known as the dynamic memory in our system. It can be thought of as a large block of memory that is expandable and shrinkable during the execution of a program.
- Allocation and De-allocation of memory blocks during the execution of a program can be done using new and delete operators in C++.
- Heap memory can be expanded as long as we do not exhaust the machine memory itself. It is not good from a programming perspective to completely use the machine memory to avoid errors, thus we must use the heap memory carefully.

Types of Memory Allocation

1. Static Memory Allocation in C++ (Compile-time Memory Allocation)
When memory is allocated at compile-time, it is referred to as Static Memory Allocation.

A fixed space is allocated for the local variables, function calls, and local statements, that cannot be changed during the execution of the program. We cannot allocate or de-allocate a memory block once the execution of the program starts.

We can't re-use the static memory while the program is running. As a result, it is less effective.

2. Dynamic Memory Allocation in C++ (Run-time Memory Allocation)
When memory is allocated or de-allocated during run-time, it is referred to as Dynamic Memory Allocation in C++.

A variable space is allocated that can be changed during the execution of the program.

We use dynamic/heap memory to allocate and de-allocate a block of memory during the execution of the program using new and delete operators.

We can re-use our heap memory during the run-time of our program. As a result, it is highly effective.

Need of Dynamic Memory Allocation?

There were some drawbacks of stack memory or static memory allocation, like the space allocated for the stack cannot be expanded during the execution of a C++ program or we can't keep variables in the program till the time we want. So, to overcome these drawbacks, we use the Dynamic Memory Allocation concepts.

Dynamic Memory Allocation is also a very essential topic in the field of data structures, and it is utilized practically in all data structures. For example, dynamic arrays, linked lists, queues, trees, stack, etc.

Process of Memory Allocation & De-allocation

1. Creating the Dynamic Space in Memory.

During the dynamic memory allocation in C++, first, we have to create a dynamic space (in the heap memory). We use the **new** operator to create a dynamic space.

2. Storing its Address in a Pointer

Once a dynamic space a created, we have to store the address of the allocated space in a pointer variable to access and modify the contents of the memory block.

3. Deleting the Allocated Space

Once the user does not require the memory block, we delete the allocated space using the **delete** operator to free up the heap memory.

1. new operator

The new operator in C++ is used to dynamically allocate a block of memory and store its address in a pointer variable during the execution of a C++ program if enough memory is available in the system.

Syntax:

data_type* ptr_variable = new data_type;

ptr_variable is a pointer, which stores the address of the type **data_type**. Any pre-defined data types, like **int**, **char**, etc., or other user-defined data types, like classes, can be used as the data_type with the new operator.

Example1:

```
#include <iostream>
using namespace std;
int main(void)
{
   int* ptr_variable = new int; // dynamically allocating an integer memory

   *ptr_variable = 12;  // storing a value at the memory pointed by pointer

   cout << "Value at ptr_variable memory= " << *ptr_variable;

} // Output: 12
```

- A memory block is allocated using the new operator and the address of the memory block has been stored in the ptr_variable pointer.
- (*ptr_variable) represents the value at the allocated memory location. We have assigned 12 in the memory using *ptr_variable = 12 expression.

Example2:

```
#include <iostream>
using namespace std;
int main(void)
{
   int* ptr_variable = new int (12); // dynamically allocating an integer memory

   cout << "Value at ptr_variable memory= " << *ptr_variable;

} // Output: 12
```

2. delete operator

Once we no longer need to use a variable that we have declared dynamically, we can deallocate the memory occupied by the variable.
For this, the delete operator is used. It returns the memory to the operating system. This is known as memory deallocation.

Syntax:
delete ptr_variable;

Example1:
```
#include <iostream>
using namespace std;
int main(void) {
   int* ptr_variable = new int;
   *ptr_variable = 12;
   cout << "Value at prt memory= " << *ptr_variable;
   delete ptr_variable;  // deallocating the memory
} // Output: 12
```

- Here, we have dynamically allocated memory for an int variable using the pointer ptr_variable.
- After printing the contents of ptr_variable, we deallocated the memory using delete.

Example2:
```
#include <iostream>
using namespace std;
int main(void) {
   int* ptr_variable = new int;
   *ptr_variable = 12;
   cout << "Value at pointer memory= " << *ptr_variable << endl;
   delete ptr_variable;
   cout << "Value at pointer memory= " << *ptr_variable;
}
// Output: 12                    // Output: Garbage Value
```
Note: If the program uses a large amount of unwanted memory using new, the system may crash because there will be no memory available for the operating system. In this case, the delete operator can help the system from crash.

new and delete Operator for Arrays

```
#include <iostream>
using namespace std;
int main(void)
{
   int n;
   cout << "Enter size of Array: ";
   cin >> n;

   int *ptr_array = new int[n];
   cout << "Enter array elements: ";
   for (int i=0 ; i<n ; i++)
   {
      cin >> *(ptr_array+i);
   }

   cout << "Elements in array: ";
   for (int i=0 ; i<n ; i++)
   {
      cout << *(ptr_array+i) << " ";
   }

   delete [] ptr_array;    // array memory released;

   cout << "\nElements in array: ";
   for (int i=0 ; i<n ; i++)
   {
      cout << *(ptr_array+i) << " ";
   }

}
```