new and delete Operators in C++ For Dynamic Memory

In C++, stack memory is automatically allocated for variables at compile time and has a fixed size. For greater control and flexibility, dynamic memory allocation on the heap is used, allowing manual allocation with new and deallocation with delete.

It allows the program to request memory from the heap at runtime using the new operator and release it using the delete operator. This is useful when the size of required memory isn't known at compile time, such as for variable-sized arrays or dynamic data structures like linked lists and trees.

The **new** operator in C++ allocates memory from the Free Store (a portion of the heap). If enough memory is available, it initializes the memory with a default value based on its type and returns the address of the allocated memory

Example:

```
#include <iostream>
#include <memory>
using namespace std;
int main() {

   // Declared a pointer to store
   // the address of the allocated memory
   int *nptr;

   // Allocate and initialize memory
   nptr = new int(6);

   // Print the value
   cout << *nptr << endl;

   // Print the address of memory
   // block
   cout << nptr;
   return 0;
}
```

Output

```
6
0xb52dc20
```

Allocate Block of Memory (Array)

A new operator is also used to dynamically allocate a block (an array) of memory of given data type as shown below:

```
new data_type[n];
```

This statement dynamically allocates memory for n elements of given data_type. Arrays can also be initialized during allocation.

Example:

```
#include <iostream>
#include <memory>
using namespace std;
int main() {
  // Declared a pointer to store
  // the address of the allocated memory
  int *nptr;
    // Allocate and initialize array of
  // integer with 5 elements
  nptr = new int[5]{1, 2, 3, 4, 5};
  // Print array
  for (int i = 0; i < 5; i++)
    cout << nptr[i] << " ";
  return 0;
}
```

Output

```
1 2 3 4 5
```

What if enough memory is not available during runtime?

If enough memory is not available in the heap to allocate, the new request indicates failure by throwing an exception of type std::bad_alloc, unless "nothrow" is used with the new operator, in which case it returns a nullptr pointer. Therefore, it may be a good idea to check for the pointer variable produced by the new before using its program.

```
int *p = new (nothrow) int;

if (!p) {

  cout << "Memory allocation failed\n";

}
```

delete Operator

In C++, delete operator is used to release dynamically allocated memory. It deallocates memory that was previously allocated with new.

Syntax

delete ptr;

where, ptr is the pointer to the dynamically allocated memory.

To free the dynamically allocated array pointed by pointer variable, use the following form of delete:

delete[] arr;

Example:

```
#include <iostream>
using namespace std;
int main() {
  int *ptr = NULL;

  // Request memory for integer variable
  // using new operator
  ptr = new int(10);
  if (!ptr) {
    cout << "allocation of memory failed";
    exit(0);
  }
  cout << "Value of *p: " << *ptr << endl;
  // Free the value once it is used
  delete ptr;
    // Allocate an array
  ptr = new int[3];
  ptr[2] = 11;
  ptr[1] = 22;
  ptr[0] = 33;
  cout << "Array: ";
  for (int i = 0; i < 3; i++)
    cout << ptr[i] << " ";
// Deallocate when done
  delete[] ptr;
    return 0;
}
```

Output

```
Value of *p: 10

Array: 33 22 11
```

Errors Associated with Dynamic Memory

As powerful as dynamic memory allocation is it is also prone to one of the worst errors in C++. Major ones are:

Memory Leaks

Memory leak is a situation where the memory allocated for a particular task remains allocated even after it is no longer needed. Moreover, if the address to the memory is lost, then it will remain allocated till the program runs.

Solution: Use smart pointers whenever possible. They automatically deallocate when goes out of scope.

Dangling Pointers

Dangling pointers are created when the memory pointed by the pointer is accessed after it is deallocated, leading to undefined behaviour (crashes, garbage data, etc.).

Solution: Initialize pointers with nullptr and assign nullptr again when deallocated.

Double Deletion

When delete is called on the same memory twice, leading to crash or corrupted program.

Solution: assign nullptr to the memory pointer when deallocated.

Mixing new/delete with malloc()/free()

C++ supports the C style dynamic memory allocation using malloc(), calloc(), free(), etc. But these functions are not compatible. It means that we cannot allocate memory using new and delete it using free(). Same for malloc() and delete.

Placement new

Placement new is a variant of new operator. Normal new operator both allocates memory and constructs an object in that memory. On the other hand, the placement new separates these actions. It allows the programmer to pass a pre-allocated memory block and construct an object in that specific memory.