C++ Memory Management: new and delete

programiz.com/cpp-programming/memory-management

Join our newsletter for the latest updates.

In this tutorial, we will learn to manage memory effectively in C++ using new and delete operations with the help of examples.

C++ allows us to allocate the memory of a variable or an array in run time. This is known as dynamic memory allocation.

In other programming languages such as Java and Python, the compiler automatically manages the memories allocated to variables. But this is not the case in C++.

In C++, we need to deallocate the dynamically allocated memory manually after we have no use for the variable.

We can allocate and then deallocate memory dynamically using the `new` and `delete` operators respectively.

C++ new Operator

The `new` operator allocates memory to a variable. For example,

```
// declare an int pointer
int* pointVar;

// dynamically allocate memory
// using the new keyword
pointVar = new int;

// assign value to allocated memory
*pointVar = 45;
```

Here, we have dynamically allocated memory for an `int` variable using the `new` operator.

Notice that we have used the pointer *pointVar* to allocate the memory dynamically. This is because the `new` operator returns the address of the memory location.

In the case of an array, the `new` operator returns the address of the first element of the array.

From the example above, we can see that the syntax for using the `new` operator is

```
pointerVariable = new dataType;
```

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**.

The syntax for this operator is

```
delete pointerVariable;
```

Consider the code:

```
// declare an int pointer
int* pointVar;

// dynamically allocate memory
// for an int variable
pointVar = new int;

// assign value to the variable memory
*pointVar = 45;

// print the value stored in memory
cout << *pointVar; // Output: 45

// deallocate the memory
delete pointVar;
```

Here, we have dynamically allocated memory for an `int` variable using the pointer *pointVar*.

After printing the contents of *pointVar*, we deallocated the memory using `delete`.

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.

Example 1: C++ Dynamic Memory Allocation

```
#include <iostream>
using namespace std;

int main() {
    // declare an int pointer
    int* pointInt;

    // declare a float pointer
    float* pointFloat;

    // dynamically allocate memory
    pointInt = new int;
    pointFloat = new float;

    // assigning value to the memory
    *pointInt = 45;
    *pointFloat = 45.45f;

    cout << *pointInt << endl;
    cout << *pointFloat << endl;

    // deallocate the memory
    delete pointInt;
    delete pointFloat;

    return 0;
}
```

Output

```
45
45.45
```

In this program, we dynamically allocated memory to two variables of `int` and `float` types. After assigning values to them and printing them, we finally deallocate the memories using the code

```
delete pointInt;
delete pointFloat;
```

Note: Dynamic memory allocation can make memory management more efficient.

Especially for arrays, where a lot of the times we don't know the size of the array until the run time.

Example 2: C++ new and delete Operator for Arrays

```
// C++ Program to store GPA of n number of students and display it
// where n is the number of students entered by the user

#include <iostream>
using namespace std;

int main() {
    int num;
    cout << "Enter total number of students: ";
    cin >> num;
    float* ptr;

    // memory allocation of num number of floats
    ptr = new float[num];

    cout << "Enter GPA of students." << endl;
    for (int i = 0; i < num; ++i) {
        cout << "Student" << i + 1 << ": ";
        cin >> *(ptr + i);
    }

    cout << "\nDisplaying GPA of students." << endl;
    for (int i = 0; i < num; ++i) {
        cout << "Student" << i + 1 << " :" << *(ptr + i) << endl;
    }

    // ptr memory is released
    delete[] ptr;

    return 0;
}
```

Output

```
Enter total number of students: 4
Enter GPA of students.
Student1: 3.6
Student2: 3.1
Student3: 3.9
Student4: 2.9

Displaying GPA of students.
Student1 :3.6
Student2 :3.1
Student3 :3.9
Student4 :2.9
```

In this program, we have asked the user to enter the number of students and store it in the *num* variable.

Then, we have allocated the memory dynamically for the `float` array using *new*.

We enter data into the array (and later print them) using pointer notation.

After we no longer need the array, we deallocate the array memory using the code `delete[] ptr;`.

Notice the use of `[]` after `delete`. We use the square brackets `[]` in order to denote that the memory deallocation is that of an array.

Example 3: C++ new and delete Operator for Objects

```
#include <iostream>
using namespace std;

class Student {
    int age;

   public:

    // constructor initializes age to 12
    Student() : age(12) {}

    void getAge() {
        cout << "Age = " << age << endl;
    }
};

int main() {

    // dynamically declare Student object
    Student* ptr = new Student();

    // call getAge() function
    ptr->getAge();

    // ptr memory is released
    delete ptr;

    return 0;
}
```

Output

```
Age = 12
```

In this program, we have created a `Student` class that has a private variable *age*.

We have initialized *age* to `12` in the default constructor `Student()` and print its value with the function `getAge()`.

In `main()`, we have created a `Student` object using the `new` operator and use the pointer *ptr* to point to its address.

The moment the object is created, the `Student()` constructor initializes *age* to `12`.

We then call the `getAge()` function using the code:

```
ptr->getAge();
```

Notice the arrow operator `->`. This operator is used to access class members using pointers.