Release of Memory

In this lesson, we will learn about the second component of memory management, i.e., memory deallocation.

WE'LL COVER THE FOLLOWING ^

- delete
- delete[]

delete

The delete operator deallocates the memory allocated by the new operator.

```
Point* p = new Point(1.0, 2.0);
delete p;
```

If the deleted object belongs to a type hierarchy, the destructor of the object and the destructors of all base classes will be automatically called. If the destructor of the base class is **virtual**, we can destroy the object with a pointer or reference to the base class.

After the memory of the object is deallocated, further access to the object results in undefined behavior. We have to point the deleted object's pointer of the object to a different object.

Using delete to deallocate an object allocated with new[] will result in undefined behavior.

delete[]

We have to use the operator delete[] for the deallocation of a C array that
was allocated with new[].

```
Point* p = new Point[15];
delete[] p;
```

In contrast to delete, delete[] calls all the destructors of the C array.

Using delete[] to deallocate an object allocated object with new will result in an undefined behavior.

Let's see a few examples of memory management in the next lesson.