References

Chapter 9.8 – Dynamic Memory Allocation

Starting Out With C++. From Control Structures through Objects (Eighth Edition)

So far we have used the way like the following to declare and initialize variables.

```
int ii = 0;
```

With such an approach memory is allocated when the program starts. The size of the memory for the variable is fixed when the program is created, and it will never change in the entire life of the application. It applies to global variables, file scope variables, and variables qualified with static defined inside functions. This is called **static memory allocation**.

Memory can also be allocated while the program is running. This is called **dynamic memory allocation (DMA)**, in which the amount of memory used can change depending on what happens while the program is in the middle of running.

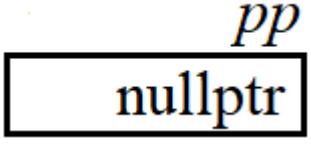
The new operator

The **new** operator is used to allocate storage for **variables**, **arrays** or **objects** while program is running.

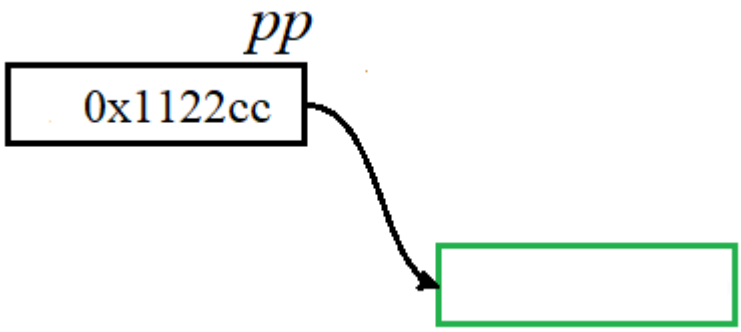
```
int * pp = nullptr;

pp = new int;
```

The first statement declares pp as an int pointer.



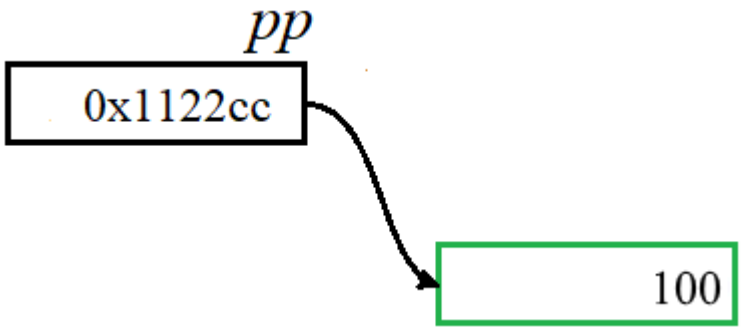
And the second statement would assign the address of an int variable to pp. The amount of the memory allocation in **runtime** is 4 bytes. In the following figure we assume that the address of the box is 0x1122cc, thus the value of pp is 0x1122cc.



Integer value can be assigned to the box at later times. For example, the following assignment:

```
*pp = 100;
```

will assign an integer value of 100 to the box.



We have to be very clear that pp is declared in exactly the same way which a regular integer variable is declared. Thus, the memory allocation for pp is **static.** The dynamic part is the integer 100 whose box was newed. (That is, the memory allocation for the black box is static, while the green one is dynamic.)

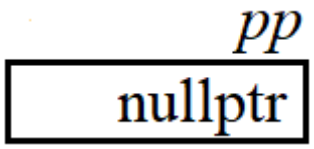
The delete operator

Whenever a variable is "**newed**," it must be deleted by using **delete** operator to avoid memory leak.

```
delete pp;

pp = nullptr;
```

The two statements above will de-allocate (release) the dynamically allocated memory and reset the pp variable back to nullptr again. That is, the green box is released and black block still exist in the memory.



Memory leak

Memory leaks occur when new memory is allocated dynamically and never deallocated. In last section we know that the delete operator is used to de-allocate dynamically allocated memory. If delete operation is not properly used for all possible scenarios, then some memories may not be released.

With memory leaks occurring over the time an application can exhaust available system memory. A memory leak reduces the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become inaccessible and all or part of the system or device stops working correctly. That is why when a software system or a device slows down, we have to reboot it to regain the normal performance.

It may seem to be straightforward to fix memory leak problems. But in reality because of the complexity of system logic and various reasons, memory leaks occur quite often in applications and hardware devices with embedded systems. BoundsChecker, Deleaker, Insure++ and Dr. Memory are some of the more popular memory debuggers for C and C++ programs, which can be used for determining which memory blocks caused the leaks.

A story

After Genuine Software released its software system, John Smith, the engineer of Genuine Software, noticed that the performance of a software system deteriorated over the time and the system had to be rebooted at least once a week. By using BoundsChecker John was able to determine that the leaks were from a report generation library which is provided by a third part company named ABC. He thought that they could easily fix the bugs by proving the information to ABC. So John called the company.

After hearing the descriptions of the bug, the support engineer at ABC responded: "Well, this bug was already recoded three months ago."

John gladly asked: "Great! So the fix must be on the way."

The support engineer at ABC said: "No. Not quite."

With confusions, John continued: "Why?"

Pausing a few seconds, the support engineer said: "It is .... I mean the leak. The leak is generated from a third party library."

How to fix the bugs of leaking memory?

There are at least two ways to fix the memory leak problems. One way is by designing a manager class which "news" its variables (or objects) in the constructors and deletes them in the destructor. The other way is use Smart Pointers, such as unique_ptr, shared_ptr and weak_ptr.

A comparison between Static and Dynamic Memory Allocations

The following table briefly compares the differences between these two types of memory allocations.

	Static Memory Allocation	**Dynamic Memory Allocations**
Declaration/ Initialization – Memory allocation	int ii = 0;	int * pp = nullptr; pp = new int;
Memory release	When ii is out of its scope, the memory is **automatically** released.	The memory is released after the pointer is deleted: delete pp;
Memory leak	It cannot occur.	It occurs if pp is deleted in some scenarios.
Comment		The pointer pp has to be retained so that the data can be accessed.

The table above gives us an impression that the second approach is quite burdensome. But Dynamic Memory Allocation actually gives us at least **three benefits**:

(1) **Memory saving** - It saves the amount of memory which is used in runtime since the system can dynamically release the unneeded memory.
(2) **Various amount of needed memory** - We may not know how much memory will be needed since that depends on numerous factors.
(3) **Polymorphism** - The use of pointers and DAM enable programmers to design Polymorphism.

See also:

Pointers and Dynamic Arrays

Manager class for automatic garbage collection

Smart pointers