Pointers and Dynamic Memory

So far, we've only discussed the behavior of pointers with the stack. Let's see how pointers can be used in dynamic memory.

WE'LL COVER THE FOLLOWING

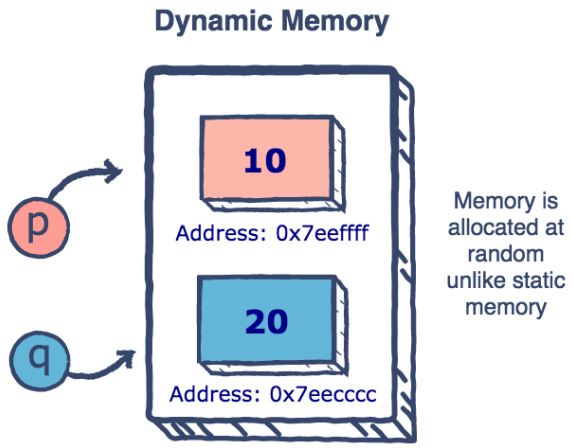
Heap: The Dynamic Memory Store

In the first lesson, we learned about the static portion of memory in C++ known as the **stack**. The counterpart for this is the dynamic section of memory known as the **heap**. This is a vast space of memory which is not being managed by the CPU.

We can specify the amount of space we want, and a random portion of the heap will be reserved for us. While the heap does allow us to use as much space as we want, the look-up operation is slower compared to a stack. However, a variable in dynamic memory (heap) can have a “global” scope instead of just being a local variable for a function.

Objects in the Heap

So, how do we access the heap? ***Pointers***. This is the true purpose of a pointer. It allows us to create dynamic objects using the `new` command.



The `delete` command releases the memory reserved by a certain pointer, making it available for future use again.

Do keep in mind that the pointer itself is stored in the stack, but can point to objects in both the stack and heap.

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

int main() {
  int *p = new int;   // dynamic memory reserved for an integer
  *p = 10;   // the object is assigned the value of 10
  cout << "The value of the object p points to: " << *p << endl;

  int *q = p;   // both pointers point to the same object
  cout << "The value of the object q points to: " << *q << endl;

  double *arr = new double[500]; // an array of size 500 has been created in the heap
  arr[0] = 50;
  cout << "arr[0]: " << arr[0] << endl;

  // delete pointers and free up space
  delete p, q;
  delete[] arr;
  cout << "p now points to a random value and should not be accesed: " << *p << endl;
  p = new int(5); // The pointer can now be re-used to point to something else
  cout << "The value of the object p points to: " << *p << endl;
}
```

In the code above, the `new` objects are created during runtime (instead of compile time). This is an advantage since we don't need to specify the amount of memory we need at compile time.

Pointer Cheat Sheet

Syntax	Purpose
`int *p`	Declares a pointer `p`
`p = new int`	Creates an integer variable in dynamic memory and places its address in p
`p = new int(5)`	Creates an integer object in dynamic memory with the value of 5
`p = new int[5]`	Creates a dynamic array of size 5 and places the address of its first index in p
`p = &var`	Points `p` to the `var` variable
`*p`	Accesses the value of the object `p` points to
`*p = 8`	Updates the value of the object `p` points to
`p`	Accesses the memory address of the object `p` points to

We've learned the basics about the nature of pointers. In the next lesson, we'll take a look at the interaction between pointers and functions.