Memory Allocation

What is memory allocation? And what is manual memory management?

Every variable we have seen so far is a local variable. And local variables go out of scope when the function ends. But sometimes, this is not what we want. That's where the *free store* comes in. The free store is memory for longer-lived variables.

Manual Memory Management

To avoid wastage of memory, you can dynamically allocate any memory required during runtime using the `new` and `delete` operators in C++.

```
int main() {

  // memory allocation
  ptr = new int[num];

  ...

  // memory is released
  delete ptr;

}
```

The `new` operator:

```
ptr = new int[num];
```

The `delete` operator:

```
delete ptr;
```

If you create something with `new`, at some point you must `delete` it. When something is not deleted, it will cause a memory leak.

The Rule of Three

The rule of three is a rule of thumb in C++ that claims that if a class defines one of the following special member functions, it should define all three:

- Destructor
- Copy constructor
- Copy assignment operator

The Rule of Five

With C++11, a new rule emerged: the rule of five. This adds two more special functions to the rule of three list:

- Destructor
- Copy constructor
- Copy assignment operator
- Move constructor
- Move assignment operator

As you can see, manual memory management is difficult and old-fashioned. Instead of using `new` and `delete`, there's something in the standard library that will make your life a lot easier.

Standard Library Smart Pointers

A *smart pointer* is an abstract data type that was popularized by C++ during the early 1990's. It simulates a pointer while providing additional features, such as automatic memory management.

In the standard library, we have the following:

- `unique_ptr`: a smart pointer that owns and manages another object through and disposes of that object when the `unique_ptr` goes out of scope.
- `shared_ptr`: a smart pointer that retains shared ownership of an object through a pointer. Several `shared_ptr` objects may own the same object.