C++ Memory Management

by Warwick New

I am not used to modern Cpp pointers. My background is more in using pointers the old fashioned way.

I highly recommend with modern versions of C++ that you look into using **Smart Pointers** over the `new` and `delete` keywords.

Learning the old way will help you understand what's going on under the hood however.

Links for smart pointers with Cpp11

1. Sutter's Mill: Elements of Modern C++
2. Bjarne Stroustrup (Creator of Cpp): Cpp11 Style

What is memory management, why is it useful?

Memory Management is the process of allocating and deallocating memory manually rather than letting the syntax handle the process for you. There are many use cases for it but here are some of the reasons to use it, and not to use it. These are similar to references in C#.

Reasons to use

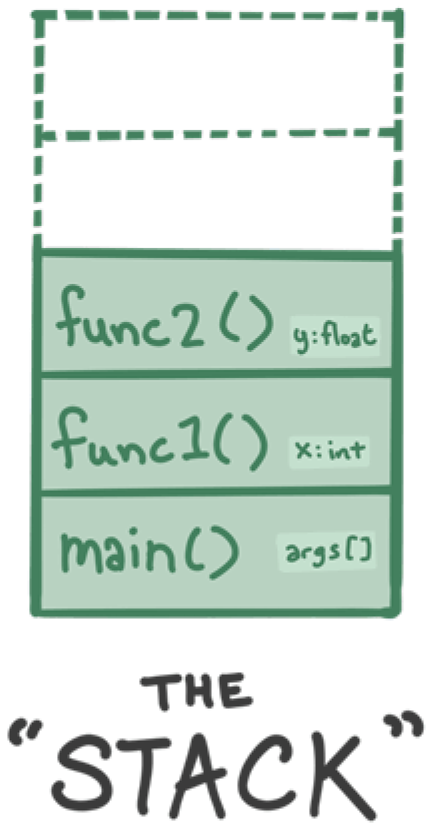
- Optimisation of both ram usage and CPU cycles wasted allocating and reading memory.
- Object lifecycle, keep an object alive even after the original parent object has died.

Reasons to avoid

- Can cause memory leaks
- Can add unnecessary complication to a piece of code.

Stack Memory vs Heap Memory

Many modern languages will manage this aspect of programming for you nowadays but every program at some level will have stack memory and heap memory. C++ and other low level languages just let you place variables and objects in whichever one you desire.



OBJ
THE
"HEAP"

An example of where you might see the two used is, having the world data stored in the heap to be accessed by many different parts of the game engine including things like the render code in the stack.

The Stack

The stack is a special area of memory which stores variables in the way you've probably been mostly using them, within scope of the functions and objects currently being run. Objects and variables used in your programs in stack memory are automatically deleted when the scope (Current function) has finished.

Advantages of Stack

- Fast
- Automatic Code Cleanup at the compiler level.

Disadvantages of Stack

- Limited memory
- Can't design code around members lasting longer than the parent.

The Heap

The Heap however is free form memory that can have objects and variables assigned to it manually using the `new` and `delete` keywords.

Advantages of Heap

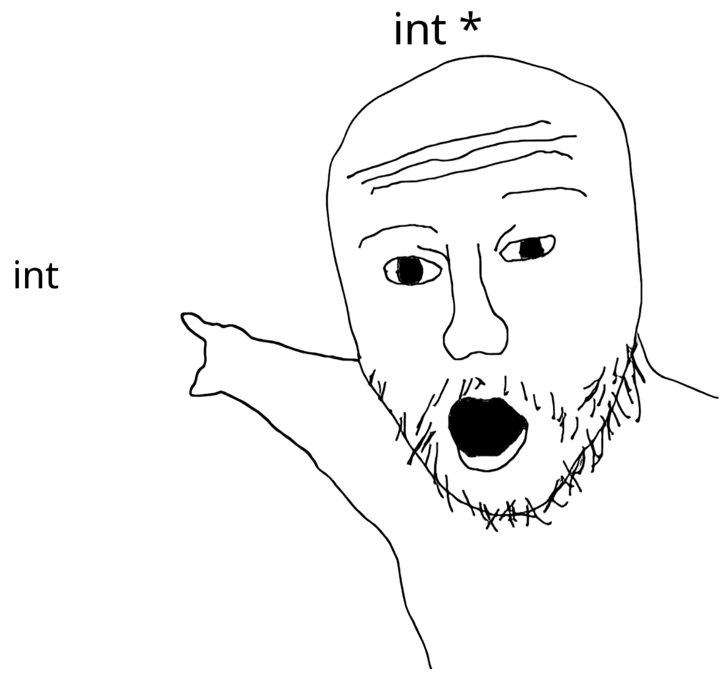
- Memory is only limited by the amount of ram in the system.
- Can design code around members lasting longer than the parent.
- Can be used to make sure your program never goes above a certain amount of ram

Disadvantages of Heap

- Slower to allocate and deallocate.
- Risk memory leaks if not used well and you're not using smart pointers properly.
- Has to manually be allocated by the programmer.

Pointers and References

As simple as possible, pointers are just variables that store the location of other variables (aka references). Pointers are one of the hardest concepts for students to get their heads around sometimes, so I reccomend you look at other descriptions yourself such as here: w3schools. But it is an indispensable tool for optimisation and sharing heap memory objects between different objects and classes.



Optimisation? how?

Consider the following program

```
double complexfunction(double data){
// lots of complex arithmatic on data that changes it
return data;
}
double bigbitodata = 999999....;

bigbitodata =  complexfunction(bigbitodata );
```

Something that you might not know here is that every time you pass a variable to a function like this it duplicates it in the stacks memory, so if the variable changes in the function it doesn't overwrite the original variable unless we tell it to with the `=` operator, which is another write to memory.

All that extra CPU/RAM time spent writing and deleting huge objects and variables from memory when we only want to change the original variable.

Introducing passing a variable to a function by reference.

```
void complexfunction(double& data){
// lots of complex arithmatic on data that changes it
}
double bigbitodata = 999999....;

complexfunction(bigbitodata);
```

In the above example you can see that we're using the & symbol in the function description to pass a reference to the data that's in the parent scope rather than the data itself to the function, passing the pointer to the object rather than the object's data. Because of this we can manipulate the object in the function and not even return the data afterwards because the data of the original object we referenced was changed in the function.

What else can they be used for?

A reference can also be stored in a variable denoted by it's type and a `*` (e.g. A pointer that points to a variable of type `int`: `int* variable;`) to be passed to other objects.

This is especially useful for large chunks of data stored in the heap that needs to be manipulated by many pieces of code such as world data in a game.

And has a lifecycle that may last far longer than the code currently running function. Like a block change in minecraft on a chunk.

`new` and `delete`

This is how you manually allocate memory to the heap manually. You use the `new` keyword to create an object and the `delete` keyword to remove it from the stack. Every single variable created with the `new` keyword must be deleted when your done with it if you don't want that memory to hang around potentially build up and cause a memory leak.

You'd use this when you want to personally manage the lifecycle of an object and/or you want an object to last longer than the scope you defined it in.

If you allocate a array of certain object types to the heap this is also a great way of making sure that you don't use more memory on that object than you want to, as by default in C++ most objects don't grow in size, once you block out that memory in the heap you know exactly how much ram you've added to the program for the lifecycle of that array.

RAII

This is all well and good but on larger programs this stuff can become hard to manage. This is where RAII comes in. And is generally in C++ how you safely manage memory.

What is RAII (Resource Acquisition is Initialisation)

- Every object is responsible for getting it's own resources and clearing them.
  - Handle all memory management of pointer content in constructors and destructors.
- Allows you to essentially use your objects like you would in C# safe in the knowledge these components will clean up after themselves when the scope closes.
- Seems to be the method the standard library is moving towards, and is practically the defacto method for C++ systems.

Task

Using this sheet create a tic tac toe board with turns in C++ that only stores the board in the heap once and never in the stack (No need to check win conditions, just prove to yourself that you can manipulate the board through a pointer). Imagine you're trying to run this game in as little an amount of ram as possible.

Showing Off?

Use smart pointers.