Pointers, a modern approach

Chantilly Robotics - Team 612

Preface

Pointers have long been considered the most complicated aspect of C and C++. Many people struggle to understand them at first, and if you don’t either that’s OK. The hard part *isn’t* the pointers themselves, but thinking logically and thoroughly. Pointers are an extremely powerful tool that set C/C++ apart from languages like Python, C# or Java. If you are patient, ask questions, and practice, you’ll be good enough to teach me about pointers in no time!

- Ahmad

A closer look at variables

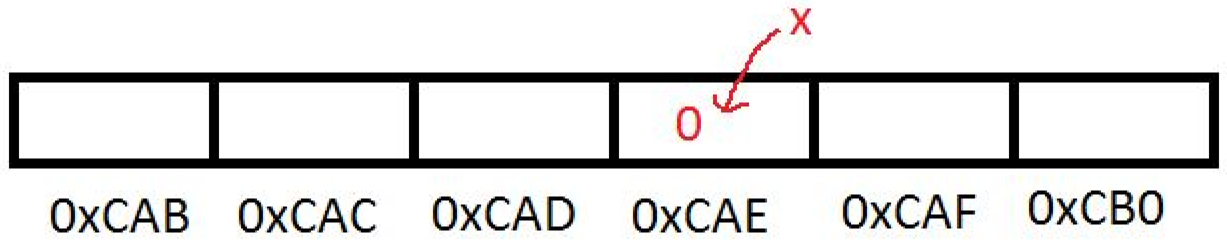
At this point we all know that

```
int x = 0;
```

Defines an integer “box” in code called “x” and sets its contents to 0. But on a fundamental level, what does that mean?

Introduction to memory

At its core, “all computer programming is memory addresses and operations on memory addresses” (Travis Axtell, lit 612 mentor). Using our box analogy, after your code creates its variable “box”, it is numbered put in a row of other boxes for your computer to use.



Memory addresses are usually represented in hexadecimal notation

A little bit of thinking...

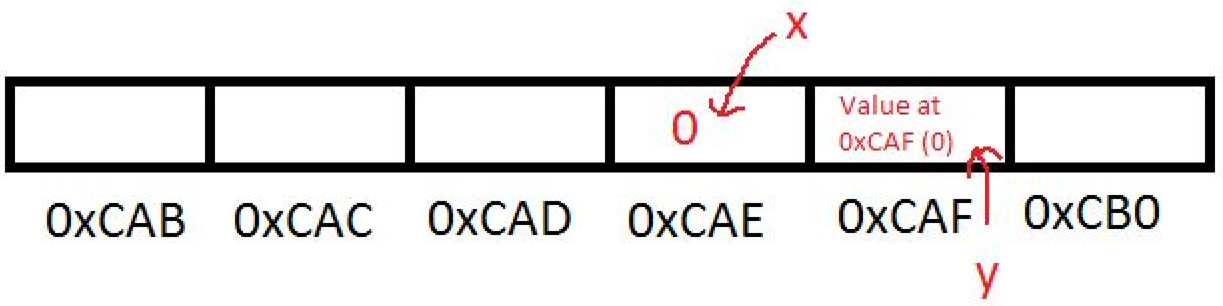
What do the next two lines of code do under the hood?

```
int x = 0;

int y = x;
```

Unforeseen consequences

You may have said that the code on the previous slide sets y equal to the value of x, 0. Which is **100% true**, but something more is going on. y has now become a *copy* of x.



Xerox machine

This isn’t really important when dealing with integers. If you want two number variables to be the same why NOT copy the contents of one number box to another? But imagine, instead of tiny integers, copying a huge class:

```
Robot robo = Robot(0, .5f, “612”);

auto robo2 = robo;
```

The code above does NOT set robo to the *same* object as robo2, but instead *copies* the contents of robo’s box and pastes it into the box of robo2. In fact, if you were to chance robo, robo2 would NOT be changed because it is not its own box.

Xerox continued

Sometimes you do want to make a copy of an object, but there are many downsides to copying objects like this:

1. It’s hard to keep track of which object has what version of the object
2. Class objects take up a huge amount of space and can make a program extremely slow.
3. Can lead to unpredictable behavior.

Pointers

A pointer is an object that *points* to a location in memory instead of its actual object.

- Analogy: You and your friend want to hang out. Instead of making a copy of your house and its contents and rebuilding it in front of your friends house (huge amount of effort and cost, destructive), you give him your address and make him drive to you.

Raw vs Smart Pointers

There are two types of pointers in C++, raw/C-style pointers and smart pointers.

- Raw pointers simply point to a memory address
- Dangerous if used incorrectly: code will compile but then can crash (segmentation fault) while running due to invalid memory handling
- Confusing for beginners (and everyone, tbh)

Lucky for you, we're not going to spend any time at all on raw pointers; they are outdated. Since C++11, we now have *smart pointers,* which are beginner (and sane person) friendly and do things behind the scenes to keep our code safe and crash free :)

Smart Pointer Introduction

The smart pointer we're going to use most often is called a *shared pointer*. One of the problems with raw pointers in the past is that, if a developer was not careful, he could give objects control over a pointer after it has been deleted in some other part of the code.

In addition to other things, a *shared pointer* will be deleted *automatically* only after all parts of the code are done using it.

Example

Study this example ***carefully***. Go through it line by line and read *all* the comments. Run it multiple times until you are comfortable with everything going on. Feel free to make edits and practice on your own. When you are done, ask your questions on Slack.

https://repl.it/E5rC/2

Unique pointers

A unique pointers is a pointer that cannot be copied. It is deleted automatically after it leaves scope and doesn't care about if the rest of the code is using it because the rest of the code can't use it unless it's directly.

- Unique pointers are declared exactly like shared pointers but with the word unique instead of shared Example:
    - `auto plane1 = std::make_unique<Plane>("Air France", 435, false);`
- Unique pointers cannot be copied. For example
    - `auto plane2 = plane1; //won't work if plane1 is unique`
- When you code, make a shared pointer first. Then, if you realize it's not/shouldn't be used anywhere else in the code, change it to a unique pointer.