Lecture 2: Memory, Programming Languages, and Objects

Intro to Memory

Your computer is equipped with Memory, or Random Access Memory, or RAM.

- You've all looked at the specs when computer shopping.
- It can be thought of as a long sequence of bytes
- Used to maintain state.
- **Non-Persistent** (doesn't maintain state when machine is off)
    - Your persistent storage (hard drive, SSD, flash) is used to store files, applications, etc.
- **Random Access**: means that each byte has an address, and the machine can grab what is in that address in a consistent amount of time.
    - In other words, it doesn't have to spin a disk, or browse some long list of things, the hardware is built to grab that data near-instantaneously.
- When your CPU fetches items in memory, it puts that data into one of its registers, where the circuitry is build to do calculations and manipulations on it (ex. adding, conditional logic, bit shifts, etc).
    - Computation cannot be done directly on RAM

32-bit system will address memory using 32 bits

- This means there can be 2^32 addresses
- That's 4,294,967,269 address, or about 4 Gb
- But now we have > 4Gb of RAM
- So, most systems are 64 bit systems
    - 1.84 * 10^19, Or 18 Quintillion
    - Not likely to need more any time soon
        - (Even using Ray Kurzweil's law of accelerating returns to project future requirements)
- The address bus will take a number and fetch or store data in memory. Usually it fetches more than a byte (words) - and it stores them in registers.
    - Fractions of a nanosecond

So, great! How do I use memory?

- Every **process** that runs is allocated a block of memory (this includes the code you write)
    - This is done by the **operating system**, called **memory management**
- This is further divided into several parts

1) **The code** itself, which is your code compiled down into machine code. Those instructions are read sequentially in memory unless it's an instruction that tells it to go somewhere else (change the **program counter**: the current machine code line number)
2) **The Stack** (this tells you what function you're in, and what your local variables are)
    a) We are going to introduce a data structure called a "stack" and this is indeed a stack
    b) Each Stack frame contains all the information needed when you are in a function
        i) All the local variables created or might be created (declared first)
        ii) All the parameters (inputs) to the function
        iii) In Java, and Object Oriented languages, this function will be a **method** inside the larger object. So the stack frame knows about that object (**this** in java)
    c) Think of the stack frames as living on top of each other, and when you return from a function, you return back to the previous one below it.
    d) Let's print a stack trace!
        i) LIVE CODING: StackTraceExamples.java
            (1) TODO: Post this.
    e) Let's cause a stack overflow!
        i) Have a cycle of function calls: a calls b calls c, which then calls a again
        ii) Use StackTraceExamples.java
3) **The heap** - this is where you, the programmer, can keep data. (Not global, not within a single function)
    a) There is also a heap data structure in this class. The "heap" in this context is DIFFERENT from the heap data structure (ugh - sorry nothing I can do)
    b) In Java, when you see the word "**new**" you are putting something into the heap
    c) All the objects you create in java will live on the heap
    d) So how do you access things on the heap? The stack will contain local variables with references to items on the heap.
        i) LIVE CODING: SampleVenue.java
            (1) TODO: provide this?
4) **Global Variables/Data** (in Java this is part of the heap)
    a) Doesn't need to be because like the code it is fixed in size and doesn't need to be dynamically allocated.
    b) An example would be a literal string you have in your code

Programming Languages And Java

Basic Idea from Computer Systems Organization

The machine has basic instructions that are built into the hardware.
Examples: Bitwise Operations, Addition, Multiplication, Shifts

- LOAD from memory
- and of course allocating basic primitive types (like int and float).

- Also Goto (a different instruction instead of just incrementing the program counter by 1), call and return - that can be used to build up loops and functions.

Machine Language: series of bits that represent instructions interfacing directly with the hardware.
Assembly Language: Machine Language as written by humans. This human short-hand is converted into machine language (binary)
Grace Hopper (1952) - use words to point to subroutines (proto-functions, proto-variables)

C (1972 - Dennis Richie): General purpose. Now loops and functions are more explicit. Pointers and Structs. But it still works with machines more directly, so if you're working with a different machine, in some cases you'll get different outputs.

- Use C if you want to have direct control over what the machine is doing in terms of allocating memory.

C++ (1985): an extension of C that has objects and classes (object oriented programming)

Each machine can be slightly different, so machine instructions that work with one architecture might not work with another. This causes a **portability** problem.

Java (released 1995 - James Gosling, Sun microsystems)

- The idea behind java is to first compile your **source code** down to ByteCode, which is like machine instructions, but it's instructions for the JVM, the Java Virtual Machine
- The JVM has to be installed on each device (that's when you install java)
    - You should install the JDK, so you can both compile (javac command) and run (java command)
- The JVM then converts Java Instructions into an instruction set for your particular machine
- Therefore, it's portable across all devices with the JVM installed

Other important facts about java

- Memory management automated
    - Garbage collection: If something on the heap no longer has a reference to it, tell the memory manager it can be freed so that it can be reused
        - Without this we had "memory leaks" which any really annoying problems that can come up in C and C++ if not careful about freeing memory
- A bit of clean up around the OOP story from C++ (streamlined)
- And of course, lots of features and further cleanup has been added over the last 30 years. (Some of them we'll go over; Generics, Regular Expressions)

What Sun Microsystems was doing with Java was immediately recognized as important by developer communities, universities, and business leaders.

- When did NYU start using Java for courses?

- My guess was the early 2000s, or even as early as 1998/1999, but I'll have to ask!

Letter from Bill Gates

- Worried about the portability of Java, could be used to build an operating system that's independent of computer architecture.
    - Undermines Windows market lockin (monopoly power)

From: Bill Gates
Sent: Monday, September 30, 1996 9:36 PM
To: Nathan Myhrvold
Cc: Aaron Contorer
Subject: Java runtime becomes the operating system

I am worry a lot about how great Java/Javabeans and all the runtime work they are doing is and how much excitement this is generating. I am literally losing sleep over this issue since together with a move to more server based applications it seems like it could make it easy for people to do competitive operating systems.

I am very interested to get your thoughts on this. Prior to the advanced work you are driving what kind of defenses do we have against this? I certainly havent' come up with enough to relax about the situation and it is undermining my creativity.

Talking About Types

Every variable in Java needs to be declared as a type.

What types exist in Java

- Primitives (int, long, char)
- Classes (which are stored as references to instances)

One reason we use data types: validation (compile time)

- The type checker
- You can design your classes to make it more likely to catch errors

Another reason

- We need to know how much data to allocate (both in the stack and in the heap - for objects, mostly in the heap)

Basic Data Structure: Object

Also structs in C

- Put several data types sequentially in memory

Show pictures of Objects with References

- Show a Venue Object
    - Refers to a Coordinates object
- TODO - show code, and drawing

“Secret Code” Reference example

- How could a secret code be changed without you knowing?
    - Complex code calls before retrieving it. If no function has access to your pointer, then you are safe. If you can confirm that nothing changes it down the call chain, you are safe
    - Why final will not save you (unless applied all the way down!)

Different threads: Outside this course, but a process can have multiple threads running. What does that mean?

- Your code is being executed in several places independently (using the same memory, multiple cores)
- How can you ensure this won’t happen?
    - If you allocated the memory yourself, and you can ensure that nothing downstream from your method fixes it, then you are good
    - If you got the reference from somewhere else, not so much
        - You could audit your code. If (and this is common) there’s no place in the code that changes anything, you are good
        - Also final in every spot will save you
- You can see why permissions and abstraction in OOP grows up around this