PPL 3/23

Reagan Shirk

March 23, 2020

Environment

- We will see:
  - Binding time
  - Object lifetime and Storage Management
    - stack: static memory allocation
    - heap: dynamic memory allocation
  - Scope rules
    - The variable (memory location) being used in various locations
    - Reach of declarations
    - Visibility of variables, functions, objects, methods, etc
  - These things apply to *every* programming language to a certain extent (the extent depends on the language)
    - Try to relate our knowledge of particular language how we could implement that from a compiler point of view
      - Knowing how to declare variables in C, how can we make our compiler accept these variable declarations?
- At this point, we are talking about the third, fourth, and fifth boxes of the compilation overview
  - Semantic analysis and intermediate code generation, machine independent optimizations, and target code generation

Need for Abstractions

- They simplify our lives
  - What takes 5 lines in Java could take 50 lines in Assembly (and probably 1 line in Python tbh)
- Environments are large abstractions
- Abstraction hides irrelevant details and focuses on main properties

Binding Time

- Binding is an association between two things, such as a name and a storage location or a name and the implementation of a subroutine
  - The name references what is in that storage location or the specific subroutine that we want to run
- Binding time is the time in which that binding is made
  - Static binding = determined at compilation time (I think?)
  - Dynamic binding = determined at program execution
- Different binding times:
  - compile time: mapping of programming language constructs to machine code and memory layout
  - linking time: functions in separate compilation units
  - run time: a function activation, dynamic memory allocation

Object Lifetime

- We want to know when the object is *officially* created and *officially* destroyed

- We want to know about the creation and destruction of both objects and bindings
- Static: objects with an absolute address throughout program execution (memory address is determined before execution and doesn't change)
- Stack: objects are allocated and deallocated in last in, first out order (LIFO)
- Heap: the memory can be allocated and deallocated at arbitrary times during program execution but requires more general and time-expensive storage management
- Imagine a pile of boxes
  - The bottom of the pile is low address memory, the top of the pile is high address memory
  - From bottom to top, the boxes are labeled:
    - code
    - static data
    - heap
    - free memory
    - stack
  - The heap takes its high addresses and puts them into free memory, the stack takes its low addresses and puts it into free memory

Lifetime in General

- Binding lifetime and object lifetime aren't always the same
- Objects can still have a value without a name (binding)
  - In passing by reference, binding lifetime is shorter than object lifetime
- It's normally a sign of a bug, but it's possible to have a longer binding lifetime than object lifetime