9 17 2018

Static Allocation

- global vars
    - accessed broadly
    - allocated when started and dealloc when end
- static vars
    - scoped to routine
    - allocated when started and dealloc when end
- literal numerical values and string
    - these are used at any point
    - prog can set aside data for these
    - may be embedded in the code
- named constants
    - assuming whose values can be determined at compile time
        - C++ requirement
- Program instructions
    - although there are just-in-time compilers where it is compiled while running

Stack-based allocation

Used for data local to a subroutine, only used while running

- A **call stack** is a structure stored in memory made up of a stack of frames
- When a subroutine is called, a new frame is pushed on top of the stack
- When the subroutine returns its frame is popped off of the stack
- A **stack pointer** register keeps track of the address at the top of the stack
- A **frame pointer** register keeps track of a location within the frame used as a base address for accessing the frame's constants

Activation record contents

- **Return address** for subroutine
- MISC booking information
- space for local vars
- often there is space set aside for temp values
    - if evaluating expressions, it takes multiple steps
- arguments for the subroutine

Heap-based allocation

- dynamically allocated data
    - the heap is not responsible for cleaning it up
- A **heap** is a chunck of memory, usually on the opposite end of the stack
- set aside for the dynamic allocation of objects

    - could be structures
    - could be symbol values
    - Java has everything dyn-alloc
        - except for primitives
- Useful for things like strings, lists, and other dynamically sized objects
- any object is allocated on the heap must be deleted explicitly (dealloc)
    - garbage collection

Heap management

- The heap itself broken up into allocated parts and deallocated parts (referred to as blocks)
    - free blocks of space
- A **free list** - linked list structure - with almost no overhead
    - head alerts size, tail alerts to the next chunk (block)
- A **first fit** algorithm finds the first space big enough and allocates it
- A **best fit** algorithm finds the smallest space that is still big enough to satisfy a request
-