Matthew Tran
CS164
MT1

- Compiler - source → target language
- Interpreter - source → value
  - ex. CPU x86_64
- assembly → machine code
- runtime - code to run stuff from compiler
- S-expressions - lisp
  - () ← list of more
  - symbol, number, list
- unary operation
  - 1 arg
  - pattern match
  - Op on Rax after compile arg
- testing
  - hand written tests
  - program
  - diff compilers ← } is language ← we use definitional interpreter
  - formal verification
- Boolean
  - add type
  - match symbols
  - "dynamic typing"
  - tag int w/ 0b00
  - tag bool w/ 0b0011111
- Conditionals
  - false → only false value
  - x86 conditional jump, jz, cmp
  - else, continue labels
    - custom labels! need to
- Binary operation
  - need to "save" Rax of 1st arg
  - stack
    - rsp = return address on stack
    - move down (rsp - 8)
    - el at stack index, dec by 8
- Let
  - symtab map var name to value
    - immutable, make new ones
  - values on OCaml to check
  - put on stack, symtab tracks it ↑ idx - 8
- Pairs
  - don't know stack size → heap!
  - heap in rdi
  - 64b-bit addr, multiple of 8 + 0b000 tag
  - no gc yet
- Errors
  - call C function error()
  - just jump to it if not match tag
  - interpreter doesn't check "wellformed" bc dynamically everything
- User-Input
  - OCaml unspecified arg evaluation
  - call C function
    - save rdi
    - move rsp down (custom rsp - 8)
    - return addr at rsp, when call to func
    - align rsp to 16
- User-Output
  - main return
  - C args
    - 1st arg in rdi
- Functions
  - symtab
  - dwarf use env when calling ↑ lexical scope [illegible]
  - 1st arg rsp - 8 → bc return addr at rsp
  - 2nd rsp - 16
  - ... last in Rax?
- Tail Call
  - reuse stack frame (don't ↓ rsp)
    - just jmp, no call
- Parsing
  - tokenization
  - parsing
    - formal

- x86_64
  - rax - return value
  - rsp - stack pointer
  - rdi - our heap pointer
    - match C
- HW1
  - desugaring
- HW2
  - printable characters
  - other cases
    - newline
    - space
  - char tag 0b00001111
- HW3
  - short circuit
  - case
    - val is int
    - left side is int
    - ≥ 1 case
    - last default
    - first match returned
  - break table!
- HW4
  - list
    - () empty
    - pair chain
  - C 1st arg Rdi
  - vector tag 0b101
- HW5
  - channel tag 0b011
  - align to 8 bytes
  - in channel, out channel
    - 0b011111 / 0b0111111
  - C arg
    - 1st - Rdi
    - 2nd - Rsi
    - 3rd - Rdx
- Drills
  - mov rsp, [rsp - 8]

- Class compiler
  - raw numbers
  - (add1 x), (sub1 x)
  - true, false
  - (not e), (num? e), (zero? e)
  - (if <test> <then> <else>)
  - (+ e1 e2), -, =, < too
  - [illegible]
  - (let ((<var> e1)) e2)
    - note 1 variable only!
  - (pair e1 e2), (left e), (right e)
  - (read-num)
  - (print e), (newline), (do e1 ... en) ← eval to en
  - (define (<fname> <arg1> ...) e) ← [illegible]
- HW0 - none
- HW1
  - bin } sep language
  - variadic
- HW2
  - (char? e), (char→num e), (num→char e)
- HW3
  - and, or, multiple let, case ← env var break var
  - *, /
- HW4
  - list?, vector?, vector-length
  - vector, vector-get
  - vector-set
- HW5
  - (string? e), stdin, stdout
  - open-in, open-out, close-in, close-out
  - input, output

Matthew Tran
CS164
Final

- First-class functions
  - function basically a value
  - anonymous function (lambdas)
  - closure - ref outside variable
  - interpreter
    - add lookup table for [illegible]
  - compiler
    - [illegible], store fun [illegible]
- Lambda
  - anonymous function
  - [illegible] by [illegible] to top level and [illegible] name
- Closure
  - save the environment
  - interpreter - just pass env
  - compiler
    - extra vars are "free variables"
      find at?
    - add to stack, like extra arg
      heap
    - pointer to heap w/ args
- Optimization
  - [illegible]
  - goals
    - faster
    - less memory
    - less energy
    - less size
    - less network work
    - better [illegible] (ex. cache)
  - other
    - readability
    - correctness
  - tradeoffs
    ex. size vs. speed
  - when?
    ASTs usually
    peephole (assembly)
  - order? [illegible] phase ordering
    - usually just run and [illegible]
  - Constant folding
    - recursive
    - check if can fold [illegible] it
- Peephole optimization
  - look at small window
  - ideas
    - algebraic simp
    - elim redundant load/store
    - elim unreachable code
    - flow of control optimization
      j L1 / L1: / j L2 → j L2 / L1: / j L2
  - HW8 opt
    - constant propagation
    - inlining
    - common subexpression elim
- Garbage Collection
  - [illegible] idea
    - stop [illegible]
    - find dead obj (not reachable from [illegible]) to free → "mark and sweep" → [illegible]
  - reference counter
  - mark and copy
    - [illegible] space
    - move everything over and switch [illegible]
- AST → parallel
- [illegible] for each call to [illegible] w/ [illegible]
  - heap addr + [illegible]
- Intermediate Representation (IR)
  - not AST [illegible]
- Register Allocation
-
- Mark and Sweep
  - mark, graph [illegible] all reachable
  - sweep all memory, add unmarked to free list
    allocation
  - fragmented
- Stop and Copy
  - 2 semispaces (need 2x memory)
  - copy all reachable, [illegible]
- Generational GC
  - [illegible] GC [illegible]
- Ref Count GC
  - expensive
- Register [illegible] allocation
  - [illegible]