- architecture and compilers
- data
  - data representation
    - integer
      - representation
        - unsigned
          - range → [0, 2^w −1]
        - signed, aka two's complement
          - range → [-2^(w-1), 2^(w-1)-1]
      - conversion & casting
        - let pattern is kept, but reinterpreted
        - signed and unsigned in single expression, signed values implicitly cast to unsigned
      - expanding & truncating
        - expanding
          - signed → pad 1
          - unsigned → pad 0
        - truncating
          - like doing mod
          - result sometimes unexpected
      - calculation
        - adding
          - if overflow, discard carry bits
          - using shift
        - minusing → mind the range too
        - multiplication
          - different mode / division
        - division
          - for multiply → ✦ << n: k gives n * 2^k
          - for divide
            - unsigned
              - logic shift
              - ✦ >> k gives ⌊ n / 2^k ⌋
            - signed
              - arithmetic shift
              - rounding toward left → ✦ >> k gives ⌊ n / 2^k ⌋
              - rounding toward 0 → ✦ Categories: 1. ⌊n/2^k⌋ if n ≥ 0 2. ⌈n/2^k⌉ if n < 0 • Biased division: n + (1 << k) - 1 / 2^k → add 0..1 to favor 10 → load to modify according to the remainder part
    - float
      - IEEE 754
        - scientific notation → (−1)^s M 2^E → Sign bit s → determines whether number is negative or positive; Significand M → normally a fractional value in range [1.0,2.0); Exponent E → weights value by power of two
        - numerical form → (−1)^s · M · 2^E → Significand M → it means, normally holding a implicit leading 1 + 0.M; Exponent E → E need to minus a bias → have a sign bit → not a signed number actually → because when this bit equals 1, the exponent of the float in scientific notation is positive → the bias is for distinguishing whether the exponent of the float in scientific notation is positive or negative
        - encoding → s exp frac → 1 bit + 8 bits exp + 23 bits frac
        - single-precision 32bits → can represent 7 decimal digits, 10^(-38)
        - double-precision 64bits → 1 bit + 11 bits exp + 52 bits frac → can represent 16 decimal digits, 10^(-308)
        - types regrading to E
          - denormalized → exp = 00...00 → M = 0.frac; when exp = 00...01, frac = 00...00 → represent 0; when exp = 00...01, frac = 00...00 → represent number closest to 0 → exp != 0 && exp != 11...11 → M = 1.frac; E = exp - Bias → construction examples → ✦ Experiment: 1. When exp=0, frac=0, M=0, E=-126, V=0 2. When exp=0, frac=1, M=0.00000000000000000000001, E=-126, V=2^-149 3. When exp=0, frac=2, M=0.00000000000000000000010, E=-126, V=2^-148 4. When exp=0, frac=3, M=0.00000000000000000000011, E=-126, V=3*2^-149 5. When exp=0, frac=4, M=0.00000000000000000000100, E=-126, V=4*2^-149 6. When exp=0, frac=5, M=0.00000000000000000000101, E=-126, V=5*2^-149 7. When exp=0, frac=6, M=0.00000000000000000000110, E=-126, V=6*2^-149
          - normalized → exp = 01...10 → M = 1.frac; E = exp - Bias → construction examples → ✦ Experiment: 1. When exp=1, frac=0, M=1, E=-126, V=2^-126 2. When exp=1, frac=1, M=1.00000000000000000000001, E=-126, V=(1+2^-23)*2^-126 3. When exp=1, frac=2, M=1.00000000000000000000010, E=-126, V=(1+2^-22)*2^-126 4. When exp=1, frac=3, M=1.00000000000000000000011, E=-126, V=(1+3*2^-23)*2^-126 5. When exp=1, frac=4, M=1.00000000000000000000100, E=-126, V=(1+4*2^-23)*2^-126 6. When exp=1, frac=5, M=1.00000000000000000000101, E=-126, V=(1+5*2^-23)*2^-126 7. When exp=1, frac=6, M=1.00000000000000000000110, E=-126, V=(1+6*2^-23)*2^-126
          - special → exp = 11...11 → when exp = 11...11, frac = 00...00 → represent infinity; when exp = 11...11, frac != 00...00 → Not a Number (NaN)
        - rounding → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 1 → Round Bit: R = 1 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 1 → Round Bit: R = 0 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 0 → Round Bit: R = 1 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 0 → Round Bit: R = 0 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 0 → Round Bit: R = 0 → Sticky Bit: X = 0 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor)
        - calculation
          - addition
            - fixing → If M ≥ 1, shift M right, increment E; if M < 1, shift M left k positions, decrement E by k; Overflow if E out of range; Round M to fit frac precision → ✦ Categories: 1. 1.BBGRXXX 2. 1.BB 3. 1.BB + 0.00000000000000000000001 4. 1.BB + 0.00000000000000000000001 5. 1.BB + 0.00000000000000000000001
            - properties → ✦ Commutative: a + b = b + a → Not associative: (a + b) + c ≠ a + (b + c) → Monotonicity: a ≥ b → a + c ≥ b + c → Values of +∞ and −∞ can be used in a meaningful way
          - multiplication
            - fixing → If E out of range, overflow; Round M to fit frac precision → ✦ Categories: 1. 1.BBGRXXX * 2.1000*2^-127 2. 1.BB * 2.1000*2^-127 3. 1.BB * 2.1000*2^-127 4. 1.BB * 2.1000*2^-127 5. 1.BB * 2.1000*2^-127
            - properties → ✦ Commutative: a * b = b * a → Not associative: (a * b) * c ≠ a * (b * c) → Monotonicity: a ≥ b → a * c ≥ b * c → Values of +∞ and −∞ can be used in a meaningful way
        - conversion & casting
          - double/float → int → Truncates fractional part → Like rounding toward zero → Not defined when out of range or NaN. Generally sets to TMin
          - int → double → Exact conversion, as long as int has a 53 bit word size
          - int → float → Will round according to rounding mode
- Programs and Data
  - bit-level manipulation
    - A and → all 1 then 1, 0 otherwise
    - | or → all 0 then 0, 1 otherwise
    - ^ xor → diff then 1, same then 0
    - ~ not → all flipped
    - << left shift → pad 0
    - >> right shift → logic shift → pad 0; arithmetic shift → pad 1 if negative, pad 0 if positive
    - ! → if x == 0 return 1, else return 0
    - && → if x == 0 return 0, else if y == 0 return 0, else return 1
    - || → if x != 0 return 1, else if y != 0 return 1, else return 0
    - ? : → if x == 0 return z, else return y
    - a ^ a = 0
    - a ^ 0 = a
    - a ^ ~0 = ~a
    - a ^ b ^ b = a
    - ~0 → 0xFFFFFFFF
    - ~(1 << 31) → 0x7FFFFFFF
    - 0xAA → 0b10101010
    - 0x55 → 0b01010101
    - 0xFF → 0b11111111
    - 0x00 → 0b00000000
    - 0x80 → 0b10000000
    - 0x7F → 0b01111111
    - 0x01 → 0b00000001
    - 0xFE → 0b11111110
    - 0x03 → 0b00000011
    - 0xFC → 0b11111100
    - 0x0F → 0b00001111
    - 0xF0 → 0b11110000
    - 0x1F → 0b00011111
    - 0xE0 → 0b11100000
    - 0x07 → 0b00000111
    - 0xF8 → 0b11111000
    - 0x3F → 0b00111111
    - 0xC0 → 0b11000000
  - assembly program
    - compiling C
      - .s → asm → object → binary (executable)
      - operation
        - basic definition
          - memory address
            - x86-64
            - covered in 08033
            - 0x8fc3ff → Mem[0x8fc3ff]-8byte[0:8]
            - 0fc3fc32 → Mem[0x8fc3ff]-8byte[24]
          - instruction suffixes → %rdi, %rsi, %rdx, %rcx, %r8, %r9 → 1st, 2nd, 3rd, 4th, 5th, 6th argument; if more, storing in stack
          - registers → %rax → return value; %rsp → stack pointer; %rbx, %rbp → callee saved; %rip → current code control point; eax e.g.; %eax holds 4B, %ax holds 2B → 1 byte; ➊ word (2 bytes); ➋ long (4 bytes); ➌ quad (8 bytes)
          - condition code → CF Carry Flag → for unsigned; → when carry or borrow; ZF Zero Flag → when 0; SF Sign Flag → for signed; → depending on the sign bit; OF Overflow Flag → for signed; → when positive overflow & negative overflow → positive → 1070 + 80 = 1150 → 0x7FFFFFFF + 0x00000050 = 0x8000004F; negative → -1070 + -80 = -1150 → 0x80000000 + 0xFFFFFFB0 = 0x7FFFFFB0
        - control
          - loops
            - do-while → ✦ e.g. → Assembly Code → Control Flow
            - while → ✦ e.g. → Assembly Code → Control Flow
          - switch → ✦ e.g. → Jump table mapping
        - procedure
          - mechanism
            - passing control → ➊ beginning of procedure code; → ➋ back to return point
            - passing data → ➊ Procedure arguments; → ➋ Return value
            - memory management → ➊ Allocate during procedure execution; → ➋ Deallocate upon return
          - stack structure
            - structure → x86-64 Stack → memory goes up, stack goes down
            - stack frames
              - basic
                - operation
                  - push → decrement %rsp by 8B; → write something into this place
                  - pop → when calling a function, need to pop; → incrementt %rsp by 8B
                  - no change in memory
                - register saving
                  - end of a function, need to pop
                  - ✦ "Callee Saved" → Callee must save values in its stack frame before call
                  - ✦ "Caller Saved" → Caller saves values in its frame before using
                  - ✦ Callee restores values before returning
              - stack frames → Stack Frame for main → Stack Frame for multstore → Stack Frame for mult2 → ➊ Stack is frame pointer, usually for accessing local variables and function parameters (?); → ➋ Stack is stack pointer
              - ✦ e.g. → Control Flow Example #1 → calling a function → Control Flow Example #2 → store return address in 0x118 → using rsp remembering the return position → continue traverse the code of function → Control Flow Example #3 → return of function → Control Flow Example #4
      - code optimization
        - assembly code
        - machine code
        - executable
  - memory hierarchy
    - cache
    - virtual memory
    - memory allocation
    - garbage collection
    - memory-related bugs
    - system-level I/O
    - network programming
    - concurrency
    - concurrent server design
    - synchronization
    - thread-level parallelism
    - other topics
- scientific debugging
  - process → conclusion → diagnosis → fix → confirm
  - principal → given a the bug/data
    - approach → brute-force → 1. debugger (e.g. gcc) → 1.1 gcc -Wall -Werror → all warnings being treated as errors → e.g. → gcc -Wall -Werror -O1 -o hanlfib hanlfib.c; 1.2 try optimization level → try at least -O2, -O3 → e.g. → gcc -O2 -o hanlfib hanlfib.c; gcc -O3 -o hanlfib hanlfib.c; gcc -Og -o hanlfib hanlfib.c; gcc -O0 -o hanlfib hanlfib.c; 2. valgrind → a memory error detector → e.g. → valgrind hanlfib
    - principal → quick and dirty → 4 minutes — Wall, valgrind; 1 - 10 minutes — Informal Debugging; 10 - 60 minutes - Scientific Debugging; 60 minutes - Take a break / Ask for help
    - common error → Use of uninitialized variables; Unused values; Unreachable code; Duplicated code; Bloated functions/methods; Memory leaks; Interface misuse; Null pointers
    - recommendation → http://www.whyprogramsfail.com/index.php
  - code design → principal → good for testing; → passing, direct, short; → comment → improve readability instead of comment; → control → clear, use branches; → code review; → design patterns
    - recommendation → "The Art of Readable Code", Boswell and Fowcher, 2011
  - program
    - procedure
      - basic
        - operation
          - jumping to different part of code according to condition codes
          - the process
          - JTC Address → ✦ Address: 0x4003ee → Instruction: 0x74 0x0a → Disassembly: je 0x4003fa → Address: 0x4003fa → Instruction: 0x5d → Disassembly: pop %rbp
        - condition operation
          - calculation 0 - b
          - set low-order byte of Dest to 0 or 1 → do not alter other 7 bytes
          - sete(%dl)
          - sete → ✦ Assembly with Figure 3.14 → Machine code with Figure 3.14
          - setl → ✦ Assembly with Figure 3.14 → Machine code with Figure 3.14
      - assembly program
        - movq Src,Dest → Dest = Src
        - leaq Src,Dest → Dest = address computed by expression Src
        - addq Src,Dest → Dest = Dest + Src
        - subq Src,Dest → Dest = Dest - Src
        - imulq Src,Dest → Dest = Dest * Src
        - salq Src,Dest → Dest = Dest << Src; Also called shlq
        - sarq Src,Dest → Dest = Dest >> Src; Arithmetic → arithmetic
        - shrq Src,Dest → Dest = Dest >> Src; Logical → logic
        - xorq Src,Dest → Dest = Dest ^ Src
        - andq Src,Dest → Dest = Dest & Src
        - orq Src,Dest → Dest = Dest | Src
        - negq Src,Dest → computing A & B without setting the destination
      - control
        - switch
          - ✦ Jump table mapping
      - stack structure
        - stack frames
          - ✦ e.g. → calling a function → store return address in 0x118 → using rsp remembering the return position → continue traverse the code of function → return of function
  - data
    - representation
      - integer
        - representation
          - unsigned
            - range → [0, 2^w −1]
          - signed, aka two's complement
            - range → [-2^(w-1), 2^(w-1)-1]
        - conversion & casting
          - let pattern is kept, but reinterpreted
          - signed and unsigned in single expression, signed values implicitly cast to unsigned
        - expanding & truncating
          - expanding
            - signed → pad 1
            - unsigned → pad 0
          - truncating
            - like doing mod
            - result sometimes unexpected
        - calculation
          - adding
            - if overflow, discard carry bits
            - using shift
          - minusing → mind the range too
          - multiplication
            - different mode / division
          - division
            - for multiply → ✦ << n: k gives n * 2^k
            - for divide
              - unsigned
                - logic shift
                - ✦ >> k gives ⌊ n / 2^k ⌋
              - signed
                - arithmetic shift
                - rounding toward left → ✦ >> k gives ⌊ n / 2^k ⌋
                - rounding toward 0 → ✦ Categories: 1. ⌊n/2^k⌋ if n ≥ 0 2. ⌈n/2^k⌉ if n < 0 • Biased division: n + (1 << k) - 1 / 2^k → add 0..1 to favor 10 → load to modify according to the remainder part
      - float
        - IEEE 754
          - scientific notation → (−1)^s M 2^E → Sign bit s → determines whether number is negative or positive; Significand M → normally a fractional value in range [1.0,2.0); Exponent E → weights value by power of two
          - numerical form → (−1)^s · M · 2^E → Significand M → it means, normally holding a implicit leading 1 + 0.M; Exponent E → E need to minus a bias → have a sign bit → not a signed number actually → because when this bit equals 1, the exponent of the float in scientific notation is positive → the bias is for distinguishing whether the exponent of the float in scientific notation is positive or negative
          - encoding → s exp frac → 1 bit + 8 bits exp + 23 bits frac
          - single-precision 32bits → can represent 7 decimal digits, 10^(-38)
          - double-precision 64bits → 1 bit + 11 bits exp + 52 bits frac → can represent 16 decimal digits, 10^(-308)
          - types regrading to E
            - denormalized → exp = 00...00 → M = 0.frac; when exp = 00...01, frac = 00...00 → represent 0; when exp = 00...01, frac = 00...00 → represent number closest to 0 → exp != 0 && exp != 11...11 → M = 1.frac; E = exp - Bias → construction examples → ✦ Experiment: 1. When exp=0, frac=0, M=0, E=-126, V=0 2. When exp=0, frac=1, M=0.00000000000000000000001, E=-126, V=2^-149 3. When exp=0, frac=2, M=0.00000000000000000000010, E=-126, V=2^-148 4. When exp=0, frac=3, M=0.00000000000000000000011, E=-126, V=3*2^-149 5. When exp=0, frac=4, M=0.00000000000000000000100, E=-126, V=4*2^-149 6. When exp=0, frac=5, M=0.00000000000000000000101, E=-126, V=5*2^-149 7. When exp=0, frac=6, M=0.00000000000000000000110, E=-126, V=6*2^-149
            - normalized → exp = 01...10 → M = 1.frac; E = exp - Bias → construction examples → ✦ Experiment: 1. When exp=1, frac=0, M=1, E=-126, V=2^-126 2. When exp=1, frac=1, M=1.00000000000000000000001, E=-126, V=(1+2^-23)*2^-126 3. When exp=1, frac=2, M=1.00000000000000000000010, E=-126, V=(1+2^-22)*2^-126 4. When exp=1, frac=3, M=1.00000000000000000000011, E=-126, V=(1+3*2^-23)*2^-126 5. When exp=1, frac=4, M=1.00000000000000000000100, E=-126, V=(1+4*2^-23)*2^-126 6. When exp=1, frac=5, M=1.00000000000000000000101, E=-126, V=(1+5*2^-23)*2^-126 7. When exp=1, frac=6, M=1.00000000000000000000110, E=-126, V=(1+6*2^-23)*2^-126
            - special → exp = 11...11 → when exp = 11...11, frac = 00...00 → represent infinity; when exp = 11...11, frac != 00...00 → Not a Number (NaN)
          - rounding → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 1 → Round Bit: R = 1 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 1 → Round Bit: R = 0 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 0 → Round Bit: R = 1 → Sticky Bit: X = 1 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor) → ✦ Rounding: 1.BBGRXXX → Value: 1.BB → Precision: 2 bits → Guard Bit: G = 0 → Round Bit: R = 0 → Sticky Bit: X = 0 → Rounding Mode: Round to Even → Rounding Mode: Round to Zero → Rounding Mode: Round Up (Ceil) → Rounding Mode: Round Down (Floor)
        - calculation
          - addition
            - fixing → If M ≥ 1, shift M right, increment E; if M < 1, shift M left k positions, decrement E by k; Overflow if E out of range; Round M to fit frac precision → ✦ Categories: 1. 1.BBGRXXX 2. 1.BB 3. 1.BB + 0.00000000000000000000001 4. 1.BB + 0.00000000000000000000001 5. 1.BB + 0.00000000000000000000001
            - properties → ✦ Commutative: a + b = b + a → Not associative: (a + b) + c ≠ a + (b + c) → Monotonicity: a ≥ b → a + c ≥ b + c → Values of +∞ and −∞ can be used in a meaningful way
          - multiplication
            - fixing → If E out of range, overflow; Round M to fit frac precision → ✦ Categories: 1. 1.BBGRXXX * 2.1000*2^-127 2. 1.BB * 2.1000*2^-127 3. 1.BB * 2.1000*2^-127 4. 1.BB * 2.1000*2^-127 5. 1.BB * 2.1000*2^-127
            - properties → ✦ Commutative: a * b = b * a → Not associative: (a * b) * c ≠ a * (b * c) → Monotonicity: a ≥ b → a * c ≥ b * c → Values of +∞ and −∞ can be used in a meaningful way
        - conversion & casting
          - double/float → int → Truncates fractional part → Like rounding toward zero → Not defined when out of range or NaN. Generally sets to TMin
          - int → double → Exact conversion, as long as int has a 53 bit word size
          - int → float → Will round according to rounding mode
    - bit-level manipulation
      - A and → all 1 then 1, 0 otherwise
      - | or → all 0 then 0, 1 otherwise
      - ^ xor → diff then 1, same then 0
      - ~ not → all flipped
      - << left shift → pad 0
      - >> right shift → logic shift → pad 0; arithmetic shift → pad 1 if negative, pad 0 if positive
      - ! → if x == 0 return 1, else return 0
      - && → if x == 0 return 0, else if y == 0 return 0, else return 1
      - || → if x != 0 return 1, else if y != 0 return 1, else return 0
      - ? : → if x == 0 return z, else return y
      - a ^ a = 0
      - a ^ 0 = a
      - a ^ ~0 = ~a
      - a ^ b ^ b = a
      - ~0 → 0xFFFFFFFF
      - ~(1 << 31) → 0x7FFFFFFF
      - 0xAA → 0b10101010
      - 0x55 → 0b01010101
      - 0xFF → 0b11111111
      - 0x00 → 0b00000000
      - 0x80 → 0b10000000
      - 0x7F → 0b01111111
      - 0x01 → 0b00000001
      - 0xFE → 0b11111110
      - 0x03 → 0b00000011
      - 0xFC → 0b11111100
      - 0x0F → 0b00001111
      - 0xF0 → 0b11110000
      - 0x1F → 0b00011111
      - 0xE0 → 0b11100000
      - 0x07 → 0b00000111
      - 0xF8 → 0b11111000
      - 0x3F → 0b00111111
      - 0xC0 → 0b11000000
    - structure alignment
      - conversion → Struct A → Struct B → Struct C → Struct D → Struct E → Struct F → Struct G → Struct H → Struct I → Struct J → Struct K → Struct L → Struct M → Struct N → Struct O → Struct P → Struct Q → Struct R → Struct S → Struct T → Struct U → Struct V → Struct W → Struct X → Struct Y → Struct Z → Struct AA → Struct AB → Struct AC → Struct AD → Struct AE → Struct AF → Struct AG → Struct AH → Struct AI → Struct AJ → Struct AK → Struct AL → Struct AM → Struct AN → Struct AO → Struct AP → Struct AQ → Struct AR → Struct AS → Struct AT → Struct AU → Struct AV → Struct AW → Struct AX → Struct AY → Struct AZ → Struct BA → Struct BB → Struct BC → Struct BD → Struct BE → Struct BF → Struct BG → Struct BH → Struct BI → Struct BJ → Struct BK → Struct BL → Struct BM → Struct BN → Struct BO → Struct BP → Struct BQ → Struct BR → Struct BS → Struct BT → Struct BU → Struct BV → Struct BW → Struct BX → Struct BY → Struct BZ → Struct CA → Struct CB → Struct CC → Struct CD → Struct CE → Struct CF → Struct CG → Struct CH → Struct CI → Struct CJ → Struct CK → Struct CL → Struct CM → Struct CN → Struct CO → Struct CP → Struct CQ → Struct CR → Struct CS → Struct CT → Struct CU → Struct CV → Struct CW → Struct CX → Struct CY → Struct CZ → Struct DA → Struct DB → Struct DC → Struct DD → Struct DE → Struct DF → Struct DG → Struct DH → Struct DI → Struct DJ → Struct DK → Struct DL → Struct DM → Struct DN → Struct DO → Struct DP → Struct DQ → Struct DR → Struct DS → Struct DT → Struct DU → Struct DV → Struct DW → Struct DX → Struct DY → Struct DZ → Struct EA → Struct EB → Struct EC → Struct ED → Struct EE → Struct EF → Struct EG → Struct EH → Struct EI → Struct EJ → Struct EK → Struct EL → Struct EM → Struct EN → Struct EO → Struct EP → Struct EQ → Struct ER → Struct ES → Struct ET → Struct EU → Struct EV → Struct EW → Struct EX → Struct EY → Struct EZ → Struct FA → Struct FB → Struct FC → Struct FD → Struct FE → Struct FF → Struct FG → Struct FH → Struct FI → Struct FJ → Struct FK → Struct FL → Struct FM → Struct FN → Struct FO → Struct FP → Struct FQ → Struct FR → Struct FS → Struct FT → Struct FU → Struct FV → Struct FW → Struct FX → Struct FY → Struct FZ → Struct GA → Struct GB → Struct GC → Struct GD → Struct GE → Struct GF → Struct GG → Struct GH → Struct GI → Struct GJ → Struct GK → Struct GL → Struct GM → Struct GN → Struct GO → Struct GP → Struct GQ → Struct GR → Struct GS → Struct GT → Struct GU → Struct GV → Struct GW → Struct GX → Struct GY → Struct GZ → Struct HA → Struct HB → Struct HC → Struct HD → Struct HE → Struct HF → Struct HG → Struct HH → Struct HI → Struct HJ → Struct HK → Struct HL → Struct HM → Struct HN → Struct HO → Struct HP → Struct HQ → Struct HR → Struct HS → Struct HT → Struct HU → Struct HV → Struct HW → Struct HX → Struct HY → Struct HZ → Struct IA → Struct IB → Struct IC → Struct ID → Struct IE → Struct IF → Struct IG → Struct IH → Struct II → Struct IJ → Struct IK → Struct IL → Struct IM → Struct IN → Struct IO → Struct IP → Struct IQ → Struct IR → Struct IS → Struct IT → Struct IU → Struct IV → Struct IW → Struct IX → Struct IY → Struct IZ → Struct JA → Struct JB → Struct JC → Struct JD → Struct JE → Struct JF → Struct JG → Struct JH → Struct JI → Struct JJ → Struct JK → Struct JL → Struct JM → Struct JN → Struct JO → Struct JP → Struct JQ → Struct JR → Struct JS → Struct JT → Struct JU → Struct JV → Struct JW → Struct JX → Struct JY → Struct JZ → Struct KA → Struct KB → Struct KC → Struct KD → Struct KE → Struct KF → Struct KG → Struct KH → Struct KI → Struct KJ → Struct KK → Struct KL → Struct KM → Struct KN → Struct KO → Struct KP → Struct KQ → Struct KR → Struct KS → Struct KT → Struct KU → Struct KV → Struct KW → Struct KX → Struct KY → Struct KZ → Struct LA → Struct LB → Struct LC → Struct LD → Struct LE → Struct LF → Struct LG → Struct LH → Struct LI → Struct LJ → Struct LK → Struct LL → Struct LM → Struct LN → Struct LO → Struct LP → Struct LQ → Struct LR → Struct LS → Struct LT → Struct LU → Struct LV → Struct LW → Struct LX → Struct LY → Struct LZ → Struct MA → Struct MB → Struct MC → Struct MD → Struct ME → Struct MF → Struct MG → Struct MH → Struct MI → Struct MJ → Struct MK → Struct ML → Struct MM → Struct MN → Struct MO → Struct MP → Struct MQ → Struct MR → Struct MS → Struct MT → Struct MU → Struct MV → Struct MW → Struct MX → Struct MY → Struct MZ → Struct NA → Struct NB → Struct NC → Struct ND → Struct NE → Struct NF → Struct NG → Struct NH → Struct NI → Struct NJ → Struct NK → Struct NL → Struct NM → Struct NN → Struct NO → Struct NP → Struct NQ → Struct NR → Struct NS → Struct NT → Struct NU → Struct NV → Struct NW → Struct NX → Struct NY → Struct NZ → Struct OA → Struct OB → Struct OC → Struct OD → Struct OE → Struct OF → Struct OG → Struct OH → Struct OI → Struct OJ → Struct OK → Struct OL → Struct OM → Struct ON → Struct OO → Struct OP → Struct OQ → Struct OR → Struct OS → Struct OT → Struct OU → Struct OV → Struct OW → Struct OX → Struct OY → Struct OZ → Struct PA → Struct PB → Struct PC → Struct PD → Struct PE → Struct PF → Struct PG → Struct PH → Struct PI → Struct PJ → Struct PK → Struct PL → Struct PM → Struct PN → Struct PO → Struct PP → Struct PQ → Struct PR → Struct PS → Struct PT → Struct PU → Struct PV → Struct PW → Struct PX → Struct PY → Struct PZ → Struct QA → Struct QB → Struct QC → Struct QD → Struct QE → Struct QF → Struct QG → Struct QH → Struct QI → Struct QJ → Struct QK → Struct QL → Struct QM → Struct QN → Struct QO → Struct QP → Struct QQ → Struct QR → Struct QS → Struct QT → Struct QU → Struct QV → Struct QW → Struct QX → Struct QY → Struct QZ → Struct RA → Struct RB → Struct RC → Struct RD → Struct RE → Struct RF → Struct RG → Struct RH → Struct RI → Struct RJ → Struct RK → Struct RL → Struct RM → Struct RN → Struct RO → Struct RP → Struct RQ → Struct RR → Struct RS → Struct RT → Struct RU → Struct RV → Struct RW → Struct RX → Struct RY → Struct RZ → Struct SA → Struct SB → Struct SC → Struct SD → Struct SE → Struct SF → Struct SG → Struct SH → Struct SI → Struct SJ → Struct SK → Struct SL → Struct SM → Struct SN → Struct SO → Struct SP → Struct SQ → Struct SR → Struct SS → Struct ST → Struct SU → Struct SV → Struct SW → Struct SX → Struct SY → Struct SZ → Struct TA → Struct TB → Struct TC → Struct TD → Struct TE → Struct TF → Struct TG → Struct TH → Struct TI → Struct TJ → Struct TK → Struct TL → Struct TM → Struct TN → Struct TO → Struct TP → Struct TQ → Struct TR → Struct TS → Struct TT → Struct TU → Struct TV → Struct TW → Struct TX → Struct TY → Struct TZ → Struct UA → Struct UB → Struct UC → Struct UD → Struct UE → Struct UF → Struct UG → Struct UH → Struct UI → Struct UJ → Struct UK → Struct UL → Struct UM → Struct UN → Struct UO → Struct UP → Struct UQ → Struct UR → Struct US → Struct UT → Struct UU → Struct UV → Struct UW → Struct UX → Struct UY → Struct UZ → Struct VA → Struct VB → Struct VC → Struct VD → Struct VE → Struct VF → Struct VG → Struct VH → Struct VI → Struct VJ → Struct VK → Struct VL → Struct VM → Struct VN → Struct VO → Struct VP → Struct VQ → Struct VR → Struct VS → Struct VT → Struct VU → Struct VV → Struct VW → Struct VX → Struct VY → Struct VZ → Struct WA → Struct WB → Struct WC → Struct WD → Struct WE → Struct WF → Struct WG → Struct WH → Struct WI → Struct WJ → Struct WK → Struct WL → Struct WM → Struct WN → Struct WO → Struct WP → Struct WQ → Struct WR → Struct WS → Struct WT → Struct WU → Struct WV → Struct WW → Struct WX → Struct WY → Struct WZ → Struct XA → Struct XB → Struct XC → Struct XD → Struct XE → Struct XF → Struct XG → Struct XH → Struct XI → Struct XJ → Struct XK → Struct XL → Struct XM → Struct XN → Struct XO → Struct XP → Struct XQ → Struct XR → Struct XS → Struct XT → Struct XU → Struct XV → Struct XW → Struct XX → Struct XY → Struct XZ → Struct YA → Struct YB → Struct YC → Struct YD → Struct YE → Struct YF → Struct YG → Struct YH → Struct YI → Struct YJ → Struct YK → Struct YL → Struct YM → Struct YN → Struct YO → Struct YP → Struct YQ → Struct YR → Struct YS → Struct YT → Struct YU → Struct YV → Struct YW → Struct YX → Struct YY → Struct YZ → Struct ZA → Struct ZB → Struct ZC → Struct ZD → Struct ZE → Struct ZF → Struct ZG → Struct ZH → Struct ZI → Struct ZJ → Struct ZK → Struct ZL → Struct ZM → Struct ZN → Struct ZO → Struct ZP → Struct ZQ → Struct ZR → Struct ZS → Struct ZT → Struct ZU → Struct ZV → Struct ZW → Struct ZX → Struct ZY → Struct ZZ
      - code optimization
        - assembly code
        - machine code
        - executable

Presented with xmind