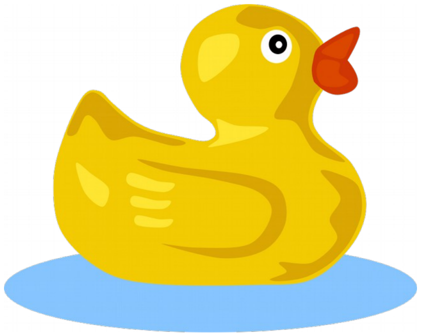
CS 430
Spring 2015

Mike Lam, Professor



$$\frac{\Gamma \vdash e' : \tau' \quad \Gamma, id : \tau' \vdash e : \tau}{\Gamma \vdash \text{let id} = e' \text{ in } e \text{ end} : \tau}$$

Data Types and Type Checking

Type Systems

- *Type system*
  - Rules about valid types, type compatibility, and how data values can be used
- Benefits of a robust type system
  - Earlier error detection
  - Better documentation
  - Increased modularization

Data Types

- *Data type*: collection of data values and their associated operations
  - *Descriptor*: collection of a variable's attributes, including its type
- Primitive data types
  - Integer, floating-point, complex, decimal, boolean, character
- User-defined data types
  - Structured: arrays, tuples, maps, records, unions
  - Ordinal: enumerations, subranges

Data Types

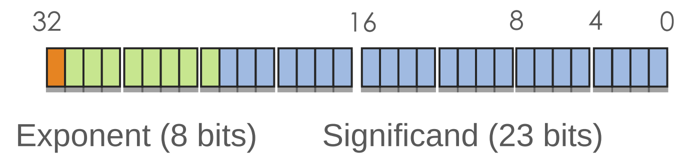
- Primitive data types
  - Integer: signed vs. unsigned, two's complement, arbitrary sizes
    - Tradeoff: storage/speed vs. range
  - Floating-point: IEEE standard (sign bit, exponent, significand), precision, rounding error
    - Tradeoff: precision vs. range
  - Complex: pairs of floats (real and imaginary)
  - Decimal: binary coded decimal
  - Boolean: 0 (false) or 1 (true); usually byte-sized
  - Character: ASCII, Unicode, UTF-8, and UTF-16 (variable-length), UTF-32 (fixed-length)

IEEE Floating Point

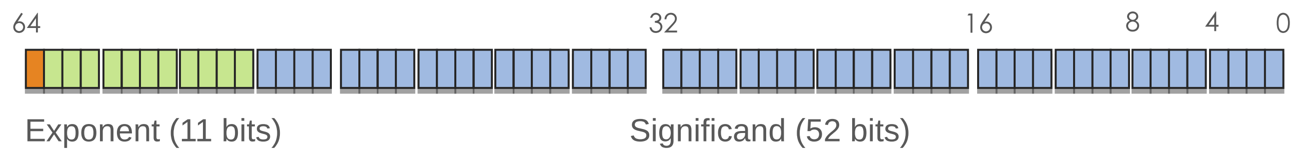
- Sign bit (s)
- Exponent (e)
- Significand (m)

Value:
$(-1)^s \cdot m \cdot 2^e$

Single Precision

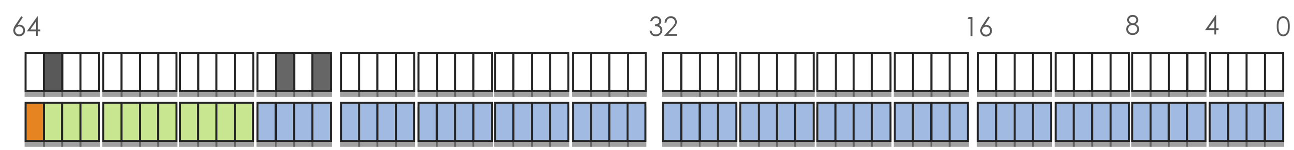


Double Precision



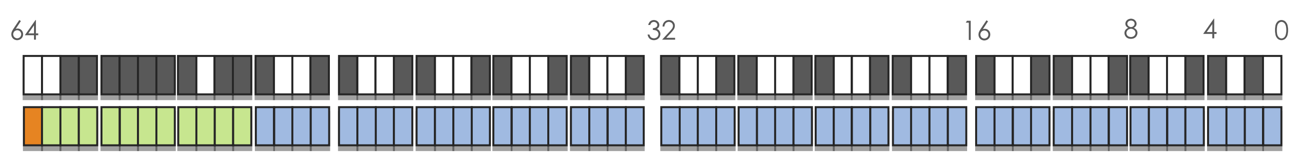
Representing 2.625:

0x4005000000000000



Representing 0.1:

0x3FB999999999999A



User-Defined Data Types

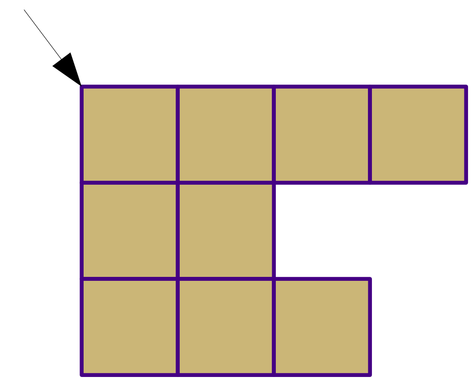
- Structured
  - Arrays and lists: sequences of elements, mapping from integers to elements
  - Tuples: fixed-length sequence of elements
  - Associative arrays: mapping from keys to values, hashing
  - Records: (name, type) pairs, dot notation, a.k.a. "structs"
  - Unions: different types at runtime, tag/discriminant, safety issues
- Ordinal (value <=> integer mapping)
  - Booleans and characters
  - Enumerations: subset of constants
  - Subranges: contiguous subsequence of another ordinal type

Arrays and Lists

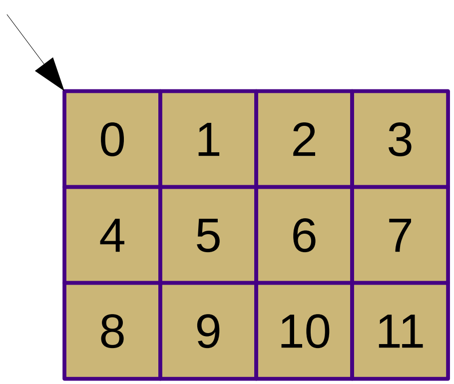
- Arrays
  - Usually homogeneous (with fixed element width)
  - Usually fixed-length
  - Usually static or fixed stack/heap-dynamic
  - Calculating index offsets: `base + index * (element_size)`
- Lists
  - Sometimes heterogeneous
  - Usually variable-length
  - Usually stack-dynamic or heap-dynamic
  - In functional languages: usually defined as `head:tail`

Multidimensional Arrays

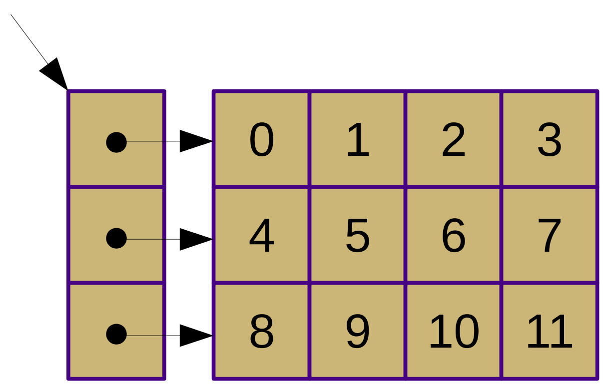
- Multidimensional arrays
  - True multidimensional vs. array-of-arrays
  - Row-major vs. column-major
  - Rectangular vs. jagged
  - Calculating index offsets



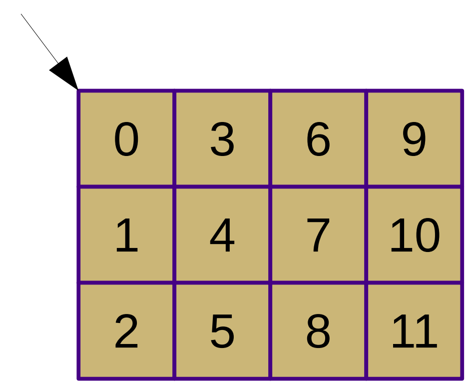
Ragged



Row-major



Row-major arrray-of-arrays



Column-major

Character Strings

- Often stored as arrays of characters
- Common operations: length calculation, concatenation, slicing, pattern matching
- Questions:
  - Should the language provide special support?
  - Should string length be static or dynamic?
    - How should the length be tracked?
  - Should strings be immutable?
- Tradeoffs: speed vs. convenience
- Buffer/length overruns are a common source of security vulnerabilities

Pointers and References

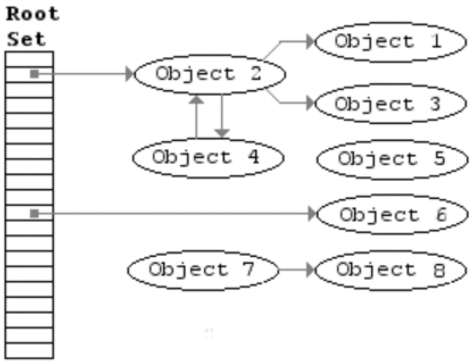
- Pointer: memory address or **null / nil / 0**
  - Example of a *nullable* type
- Reference: object or value in memory
  - Also can be nullable
  - Different semantics than pointers
  - Strictly safer than pointers
- Implementation
  - Allocation/initialization
  - Dereferencing
  - Arithmetic (allowed for pointers, not references)

Pointers and References

- Design issues
  - Scope and lifetime of pointer and associated value
  - Type restrictions (must match? void* allowed?)
  - Language support (pointers, references, or both?)
- Problems
  - Dangling pointer: value has been deallocated
    - Null pointer dereference
    - Debuggers (e.g., gdb) can help!
  - Memory leaks: value is no longer accessible
    - Memory remains allocated
    - Memory analysis tools (e.g., valgrind) can help!

Garbage Collection

- Alternative to explicit reference deallocation
- Reference counters
  - Track # of references to an object
  - Deallocate object when counter hits zero
- Mark-and-sweep
  - Pause the application (sometimes unnecessary)
  - Initialize indicators for all memory cells to "unmarked"
  - Mark reachable heap memory cells by following pointers from stack and static memory
  - Deallocate unmarked cells



Polymorphism

- Object-oriented inheritance
  - Example of *subtypes*
- Parameterized functions
  - Uses generic *type variables*
  - Example: generic list functions in Haskell
    - E.g., `head : [a] → a`
- Abstract data types
  - Models of generic data structure behavior
  - Can use *parameterized* types
    - E.g., a `queue<float>` or `queue<int>`
    - Examples: C++ templates and Java generics

Type Checking

- *Type system*
  - Rules about how data values can be used
- Type compatibility
  - Operators defined for types
  - All operand types are *equivalent*
    - Name vs. structure equivalence
- Type conversions
  - Widening vs. narrowing (may cause information loss)
  - Implicit: *coercion*, e.g., `float x = 5;`
  - Explicit: *casting*, e.g., `int x = (int)3.14;`

Type Checking

- *Type checking*
  - Ensure that operations are supported by types of the operation's operands
  - Ensure that operands are of compatible types
  - Violations are called *type errors*
  - Usually, type errors are considered to be bugs
    - Sometimes are reported only as warnings

Type Checking

- Explicit vs. implicit typing
  - Explicit: types required in declaration
    - E.g., `int x = 5; float y = 4.2;`
  - Implicit: types not required in declaration
    - E.g., `x = 5; y = 4.2;`
    - Types are bound at assignment
    - However, these types can often be inferred statically
  - Tradeoff: readability vs. writability and expressiveness

Type Checking

- Static vs. dynamic type checking
  - Static: compile time (checked by compiler)
    - E.g., C, Haskell
  - Dynamic: run time (checked by runtime system)
    - E.g., Ruby, Python
    - “Duck typing” is a special form of dynamic typing
  - Hybrid: some static, some dynamic
    - E.g., C++, Java
  - Tradeoff: overhead vs. flexibility

Type Checking

- Strong vs. weak typing
  - Strong typing: all type errors are detected
  - Tradeoff: safety vs. expressiveness
  - Terms often used somewhat loosely
- Evidence of strong typing
  - Static type checking
  - Type inference (even for implicit typing!)
- Evidence of weak typing
  - Dynamic type checking
  - Type conversions
  - Pointer or union types

Formal Type Theory

- Type systems expressed as a set of type rules
  - Each rule has zero or more premises and a conclusion
  - Apply rules recursively to form proof trees
  - Curry-Howard correspondence ("proofs as programs")
  - Can be applied to *typed* lambda calculus

$$\text{TInt}\quad \frac{}{A \vdash n : \text{int}} \qquad\qquad \frac{x : t \in A}{A \vdash x : t}\quad \text{TVar}$$

$$\text{TFun}\quad \frac{A, x : t \vdash e : t'}{A \vdash \lambda x{:}t.e : t \rightarrow t'} \qquad \frac{A \vdash e : t \rightarrow t' \qquad A \vdash e' : t}{A \vdash e\ e' : t'}\quad \text{TApp}$$

Formal Type Theory

$$\frac{}{A \vdash n : \textcolor{blue}{int}}\ \textbf{TInt}$$

$$\frac{x : \textcolor{blue}{t} \in A}{A \vdash x : \textcolor{blue}{t}}\ \textbf{TVar}$$

$$\frac{A, x : \textcolor{blue}{t} \vdash e : \textcolor{blue}{t'}}{A \vdash \lambda x{:}\textcolor{blue}{t}.e : \textcolor{blue}{t \rightarrow t'}}\ \textbf{TFun}$$

$$\frac{A \vdash e : \textcolor{blue}{t \rightarrow t'} \qquad A \vdash e' : \textcolor{blue}{t}}{A \vdash e\ e' : \textcolor{blue}{t'}}\ \textbf{TApp}$$

TVar: $\dfrac{+ : \qquad \in B}{B \vdash + :}$ TVar: $\dfrac{x : \qquad \in B}{B \vdash x :}$

TApp: $\dfrac{B \vdash + : \qquad B \vdash x :}{B \vdash + x :}$

TApp: $\dfrac{B \vdash + x : \qquad B \vdash 3 :}{B \vdash + x\ 3 :}$

TFun: $\dfrac{B \vdash + x\ 3 :}{A \vdash (\lambda x{:}\textcolor{blue}{int}.+ x\ 3) :}$

TApp: $\dfrac{A \vdash (\lambda x{:}\textcolor{blue}{int}.+ x\ 3) : \qquad A \vdash 4 :}{A \vdash (\lambda x{:}\textcolor{blue}{int}.+ x\ 3)\ 4 :}$

$A = \{ + : \textcolor{blue}{int \rightarrow int \rightarrow int} \}$ $\qquad$ $B = A, x : \textcolor{blue}{int}$

Formal Type Theory

$$\frac{}{A \vdash n : int}\ \textbf{TInt} \qquad \frac{x : t \in A}{A \vdash x : t}\ \textbf{TVar} \qquad \frac{A, x : t \vdash e : t'}{A \vdash \lambda x{:}t.e : t \to t'}\ \textbf{TFun} \qquad \frac{A \vdash e : t \to t' \quad A \vdash e' : t}{A \vdash e\ e' : t'}\ \textbf{TApp}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{+ : i \to i \to i \in B}{B \vdash + : i \to i \to i}\,\text{TVar} \quad \dfrac{x : int \in B}{B \vdash x : int}\,\text{TVar}}{B \vdash +\ x : int \to int}\,\text{TApp} \quad B \vdash 3 : int}{B \vdash +\ x\ 3 : int}\,\text{TApp}}{A \vdash (\lambda x{:}int.+\ x\ 3) : int \to int}\,\text{TFun} \quad A \vdash 4 : int}{A \vdash (\lambda x{:}int.+\ x\ 3)\ 4 : int}\,\text{TApp}$$

$A = \{ + : int \to int \to int \}$ $\qquad$ $B = A, x : int$

Announcements

- Unit 8 Online Quiz
  - Due next Monday (3/23)
- TAP next Tuesday (3/24)
  - In lieu of a midterm feedback survey
- Midterm grading goal: next Tuesday
  - Contact me TODAY if you are considering withdrawing and would like an informal grade assessment