Language features (330 recap)

- Languge concerns when designing a compiler or other language tool, for both source and target language
- Language paradigm: procedural, OO, functional, logic, hybrids
- Language purpose and associated priorities
- Data types/operators (primitive/compound, built in/user-defined)
- Exception handling
- Scoping rules/resolution
- Selection/Iteration constructs
- Subroutines
- Type handling/checking

Alphabet and naming rules

- Supported alphabet for language
- Expression of tokens (regex), solution of overlap
- Practical limitations on token length (literals, identifiers,...)
- Rules surrounding reserved words, keywords, user defined rules, etc

Primitive types and operations

- Types supported (integers, reals, booleans, characters, ordinals, etc)
- Value ranges, limits, defaults
- Supported operations, associativity, precedence
- Practical implementation concerns (storage, access, overflow/underflow)

Compound types and operators

- Arrays (single/multidim), vectors, strings, sets, lists, structures, hash tables, unions
- Built in operations and functions
- Shallow/deep copies and comparisons
- Implementation concerns and implications (storage allocation, initialization, defaults, reallocation/deallocation, parameter passing, returns)

Types, checking, conversion

- Static/dynamic typing
- Support/enforcement of run time type checking
- Implicit/explicit type conversion
- User defined types, checking
- Structural vs name type compatibility
- Narrowing/widening conversions

Variables and constants

- Typing rules
- Scoping, resolution rules
- Lifetime
- Memory allocation, deallocation, reallocation
- Initialization, defaults

Block and scoping

- Entry, exit points
- Nested blocks, scopes
- Jumps across boundaries
- Scope support/enforcement
- Storage allocation

Selection

- Single-way, two-way, multi-way
- Test and branch constructs (high and low level)
- Impact on efficiency (size and speed)

Branching and labels

- Branch mechanisms: targets by label, by distance
- Conditional and unconditional branches
- Relationship to scoping, blocks, control structures
- Within/across subroutine boundaries

Iteration

- Loop forms (for, while, repeat, etc)
- Iteration across collections (foreach)
- Entry/exit points (break, continue, final)
- Interaction with scoping (e.g. for control variables)

Subroutines

- Global vs nested
- Definitions and calls
- Parameter passing mechanisms (optional, variadic, keyword, value/reference)
- Relationship to scoping (lexical vs dynamic)
- Implementation of dynamic scoping and/or nested subroutine definitions

Dynamic memory handling

- Allocation/deallocation
- User controlled vs automatic
- Garbage collection approach
- Smart vs “dumb” pointers

ADTs/OO

- Encapsulation of fields and methods
- Abstraction, hiding, access control
- Inheritance (single/multiple, field/method overrides)
- Static vs dynamic binding
- Type checking and conversion
- Allocation/deallocation
- Generic/abstract classes

Exception handling

- Built in vs user-defined exceptions
- Exception heirarchies
- Control structures (try, throw, catch)
- Relationship to scope

And more...

- Many more features and implementation issues that are language specific (for source, or target, or platform)