Classification of Pascal’s type system
Criteria for the classification of type systems

- Existence: Does the language include a type system at all?
- Sophistication level: How rich is the type system?
- Orthogonality
- Strength
- Time of enforcement
- Responsibility
- Equivalence
- Flexibility
Existence

- Typed
  - The set of values can be broken into sets, with more or less uniform behavior under the same operation of the values in each set.
  - Examples: C, Pascal, ML, Ada, Java, and most other PLs
Existence

- Untyped
  - Each value has its own unique set of permissible operations, and their semantics are particular to the value.
  - Examples: Assembly, Lisp, Prolog, Mathematica
Sophistication level

1. No typing
2. Degenerate typing
3. Non-recursive type systems
4. Recursive Type systems
   a. as in Pascal
5. Functions as first-class values
   a. Pascal functions are not first-class values
6. Highly advanced type constructors
   a. Pascal does not have genericity, user defined overloading, user defined coercions, inheritance
Orthogonality in Pascal

- A type system is discriminatory if one of its type constructors is “discriminatory”, i.e., it is applicable to some types, but not to others.

- Non-discriminatory type constructor:
  - You can create arrays of almost anything, but not of functions
Orthogonality in Pascal

- Discriminatory type constructor:
  - You can create sets of Booleans and of Char, but not of “set of Boolean”
  - You can create File OF X for almost any X (not functions), but you would be cheating! File of Student is not type-safe! (Nominal equivalence)
Pascal discriminates functions

- We can pass functions as arguments

- But there are no:
  - Arrays of functions
  - Records of functions
  - Sets of functions

- **Conclusion**: Functions are second-class types in Pascal
First and second-class values in Pascal

- First-class values
  - Only simple, atomic values: truth values, characters, enumerands, integers, reals, and also pointers

- Second-class values
  - Can be passed as arguments, but cannot be stored, or returned, or used as components in other values
Strong vs. weak typing

- Strongly typed (as in Pascal)
  - It is impossible to break the association of a value with a type from within the framework of the language
  - It is impossible to subject a value to an operation which is not acceptable for its type

- Weakly typed
  - Values have associated types, but it is possible for the programmer to break or ignore this association
Static vs. dynamic typing

- Language implementation applies type checking to ensure that no type errors occur

- Statically typed PLs
  - Type rules are enforced at compile time
  - Every variable and every formal parameter have an associated type
  - Examples: C, Pascal, Eiffel, ML
Static vs. dynamic typing

- Dynamically typed PLs
  - Type rules are enforced at run-time
  - Variables, and more generally—expressions, have no associated type
  - Only values have fixed types
  - Examples: Smalltalk, Prolog, Snobol, AWK, python
Dynamic type checking in Pascal

● **Array range**
  ○ Arrays are defined as mapping from range of ordinal types to some type. Invalid index is a type error, checked at run time.

● Note that in C, arrays are mapping from type integer to some type. There is no static or dynamic checking. Array overflow is not a “type error”, but rather a pseudo-type error.
Dynamic type checking in Pascal

- **Division by zero:** not a type error; a pseudo-type error.
- **Null pointer reference:** not a type error; a pseudo-type error.
Type information responsibility

- Manifest typing
  - Example: Pascal

- Inferred typing
  - Example: ML

- Semi Implicit typing