Characteristics Clarity, orthogonality, abstraction, verification, IDEs, portability, "cost"

Fortran
First high level PL to be widely used: efficient
Storage allocated statically
Flat register machine: no stacks, no recursion
Types: numeric, boolean, arrays (fixed len.), strings (fixed len.), files
GOTOs in 66, 77 added control structures
Columns relevant due to punch card origins
Implicit types (int for "I"-"L", real otherwise)
Static types, but cannot check calls args. and COMMON block due to separate compilation: these also left unchecked at runtime
COMMON blocks: named shared storage block, format re-declared in every program unit
All arguments by ref., hence allows assignment to constants (cannot statically check for this)

LISP
Motivating application: automated reasoning
Expression based, has a pure subset
Prefix style syntax for ease of parsing
Compute with atoms and cells (leaves and binary tree branches): make S-expressions
Dynamically typed and scoped
Abstract machine has LISP expression, continuation, association list, heap (cons cells)
Programs as data, "eval", e.g. lazy evaluation with quoting on arguments and explicit "eval"

ALGOL
Block structure, colon-separated statements
Functions, procedures, recursion
Supports call by name, but bad w/ side effects
Static typing, but automatic type conversions not fully specified, type of procedure param. to a procedure (proc) does not include its param. types, array param. does not have bounds
Types primitive or compound (array, structure, procedure, set, pointer)

Pascal
Rich set of data structuring concepts (enumerations, subranges, records, variant records, sets, sequential files)
Index checking (array range part of type)
The use of restricted types for procedure parameters simplified compilation

BCPL
One data type: the bit pattern
Abstract machine was a store: numbered storage cells, each holding a bit pattern
Distinguishes between conceptual and internal types (which model the concepts)
Has recursion, but FVs of proc. must be global
Has a "global vector" allowing separately compiled modules to reference each other
Call by value, but since arrays are referenced by the address (@) of the base element, they end up being passed by reference
Static, implicit typing, but there is only one internal data type, so this means nothing ☺

SIMULA
Developed for writing simulations
Extension of ALGOL 60 with classes, reference variables, pass by reference, coroutines
Classes: procedures returning a pointer to a new value of its activation record
Objects: activation records produced by call to a class (i.e. objects are closures)
Inheritance defined by class prefixing, including the ability to redefine parts of a class
Type switching (inspect), safe casting (qua, :-)
Has if `B <: A` then `(B Ref) <: (A Ref)`: this is a type loophole!

Smalltalk
Motivating application: Dynabook
Execution model: everything is an object, messages to communicate between them
Selector: messae name
Message: selector + actual param. values
Methods are public, instance vars protected
"self" always refers to the object that contains this method, directly or by inheritance!
Type of an object is its interface

ML
Designed for theorem proving
Modules: ADTs, with structures and signatures
Functors: structure that takes other structures as parameters, programs can be combined in different ways (separate algorithm, structure?)

Java/C#
Boxing
Delegates vs. anonymous inner classes
Type loophole = security loophole if untrusted code is downloaded from the web
C# generics not compiled away, allow value type instantiations but no wildcards
Iterators (mimicking functional streams)
LINQ, lambdas, type inference

Constructs	Expression: syntactic entity that evaluates to a value Statement: command that alters machine state
Parameters	Formal: names used in a declaration Actual: expressions/values Name association, defaults
Scoping	Static/lexical: variable bound to closest lexical one Easier to understand, can do static analysis (and hence optimisation etc) Dynamic: variable bound to most recent declaration!
Garbage Collection	Reclamation of memory locations not accessible to a program
Evaluation Order	Call by value: reduce arguments to values first Call by name: execute body, reducing arguments to values if necessary
Parameter Passing	Pass by value: value of actual parameter copied into function Pass by reference: actual parameters L-value is copied into function Aliasing: when two names refer to the same location
Block Structure	Organise a program as nested blocks May include nested procedures that reference local declarations
Object Orientation	Dynamic lookup: method is selected directly based on message sent to an object. Means that different objects can respond to the same message differently Abstraction: hide implementation details Subtyping: relation on types that allows value of one type to be used in place of values of another. Allows functionality to be added without modifying general system parts Inheritance: reuse the definition of another object to define another object
Types	Name, organize concepts Ensure bit sequences are interpreted consistently Provide information to the compiler about prog. data Type systems are strong iff only accept safe phrases Static or dynamic typing Explicit or implicit typing How expressive is it?
Type Declarations	Transparent: alternative type name (“type”) Opaque: new type name != to another (“newtype”)
Polymorphism	Constructs that take on different types as needed Parametric, ad-hoc / overloading, subtyping