Language Design Criteria
Language Design

- Readable.
- Provides a useful set of abstractions.

Complexity control
  - Humans can only retain a certain amount of detail at once.
Language Design

- **Language Goal:**
  - C (UNIX)
  - Java (Internet, Platform Independence)
  - C++ (Efficient OO language)

- Useful API Libraries

- Ease of interface with other languages and technologies.
History
Programming Language Eras.

- 1950s.
- 1960s.
- 1970s.
- 1980s.
- 1990s.
- 2000.
- Futuristic Trends.
Language Design Principles
Language Design

Language design is one of the most difficult and poorly understood areas of computer science.

A language cannot be merely a collection of “neat” features. (Bjarne Stroustrup, C++ Designer).
Earlier, the one principal design criteria was efficiency of execution.

- Extremely slow machines.
- Program speed was a necessity.

Earlier FORTRAN code was designed to resemble as much as possible the machine code to be generated.
Language Design

- **Efficiency:**

  - **Efficiency of target code:** the language design should be such that a translator can generate efficient executable code (optimizability).

  - Example: static variables.

  - Examples: classes in C++, when not used with advanced OO features, is not much different in memory consumption and overhead than a simple C struct.
Language Design

■ Efficiency:

- **Efficiency of translation**: the source code should be translated quickly and by a reasonably sized translator.

  ■ Example: can a one-pass compiler be used? Pascal and C force you to declare variables before using them.
  ■ In C++, this is a bit relaxed, compilers must make a second pass.
  ■ Do not trade efficiency of translation for reliability: the assurance that a program will not behave in unexpected or disastrous ways during execution!
Language Design

- **Efficiency:**
  - **Implementability:** The efficiency with which a translator can be written.
    - Usually a function of language complexity.
    - The size and complexity of Ada for example hindered Ada compiler development, and impaired its availability and use.
Language Design

Efficiency:

- **Programming efficiency**: How quickly and easily can programs be written in the language?

- An expressive language allows for easy representation of complex processes and structures.
- How easy can the design in the programmer’s head be mapped to code in that language?
Language Design

- Efficiency:

  - Reliability and Maintainability: Could be viewed as an efficiency issue.
    - If programs are not reliable, they cost significantly at later stages.
    - If programs are significantly difficult to maintain, they can cost significantly also.
    - May entirely waste development efforts.
    - Efficiency of resource utilization.
Language Design

- Regularity:
  - Is a measure of how well a language integrates its features, so that there are no unusual restrictions, interactions, or behavior.
  - Generally, there should be no surprises in the way the language features behave.
Regularity:

- Regularity is divided into three more definite concepts.
  - Generality.
  - Orthogonality.
  - Uniformity.
Language Design

- Regularity:

- Generality:
  - Avoiding special cases in the availability or use of constructs.
  - Combining closely related constructs into a single more general one.
  - Too much generality is bad!
Language Design

- Regularity:

  - Examples of Lack of Generality:

    - C lacks nested function definitions.
    - Pascal has no variable-length arrays, arrays lack generality.
    - In C, two structures or arrays cannot be directly compared using the equality (==) operator, but must be compared element by element. Ada on the other hand allows totally new operators to be defined. C++ can overload operators.
    - In Pascal, constants may not be expressions, opposite to Ada.
    - Java does not have multiple inheritance, but interface inheritance implementation is a good enough substitute.
Language Design

- Regularity:
  - Orthogonality:
    - In mathematics, it means perpendicularity, or in a completely independent direction.
    - Language constructs should NOT behave differently in different contexts.
    - The language constructs can be combined in a meaningful way.
    - The interactions of constructs, or the context of use, should not cause unexpected restrictions or behavior.
    - There should be no strange interactions!
Language Design

- Regularity:

  - **Examples of Lack of Orthogonality:**

    - C passes all parameters by value, except arrays, which are passed by reference.

    - In Java, primitive data types are passed by value, the rest by reference, yet they look the same! (This is also non-uniformity)

    - In Java, assigning objects is an assignment of references, while assigning primitive data types is done by value.

    - In C and C++, values of all data types, except array types, can be returned from a function.

    - In C, local variables must be defined at the beginning of a block, in C++ variable definitions can occur anywhere inside a block, but before use of course.
Language Design

- Regularity:
  - Uniformity:
    - Similar things should look similar and have similar meanings
    - Inversely, different things should look different.
    - i.e. consistency of appearance and behavior.
    - Non-uniformity and non-orthogonality may be very closely related in some instances.
Language Design

- Regularity:

  - Examples of lack of uniformity:

    - In C++, a semicolon is necessary after a class definition but forbidden after a function definition.
      - class A { … };
      - int f() { … }
      - This non-uniformity was forced to allow C++ to be compatible with C.

    - Returned values from functions in Pascal look like assignments.

      ```pascal
      function f : boolean;
      begin
        ...
        f := true;
      end;
      ```
Language Design

- Regularity:
  
  - Examples of lack of uniformity:
    
    - In C++, the operators & (bitwise and), && (logical and) yield very different results, but look confusingly similar.
Language Design

- Simplicity:
  - Overly simple programming languages can make the task of using them more complex.
  - BASIC is a very simple language, but lacks fundamental constructs such as blocks.
  - One of Pascal’s primary reasons for success was its simplicity, and was also a reason for its failure and replacement.
Simplicity:

- C was also designed to be simple, but efficient in generating target code, and is excellent for creating UNIX operating system code, device drivers, small compilers.

- C however also has some major flaws such as somewhat obscure operator syntax, weak type checking.
Language Design

- Simplicity:
  - Einstein: “Everything should be made as simple as possible, but not simpler!”
  - Too much simplicity can fire back.
Expressiveness:

- It is the ease with which a language can express complex processes and structures (Being concise).
- One of the original advances in expressiveness was the addition of recursion to programming languages (Lisp and Algol60).
Expressiveness:

- Expressiveness can conflict with simplicity, and hence conflict with readability also.

- Example, in C, what does the following mean?
  - while (*s++ = *t++);
Expressiveness:

- Expressiveness can conflict with simplicity, and hence conflict with readability also.

- Example, in C, what does the following mean?
  - while (*s++ = *t++);
  - It actually copies a string to another!
  - Very expressive, very concise, but very unreadable!
Language Design

- Extensibility:
  - There should be some general mechanism by which the user can add features to a language.
  - Otherwise, the language becomes extremely closed.
  - Example: defining new data types, creating libraries, adding functions to a library, adding keywords.
Language Design

- Extensibility:
  - The common practice is to allow users to define:
    - New data types
    - Operations that service the data types
Extensibility:

- In C++ and Ada, overloading of operators such as “+” is limited to the existing operators only.

- In Java, overloading operators is not permitted.

- In functional languages such as ML and Haskell, one can add user-defined operators such as +++
Extensibility:

- Extensibility permits language designers to make different choices in which features to make available in the core language, and which others to leave as extensions, or to third party implementations.

- Can lead to smaller core languages.
Language Design

- Extensibility:
  - Java puts networking and multithreaded programming in its standard library.
Language Design

- **Restrictability:**
  - A language design should make it possible for a programmer to program usefully using
    - Minimal knowledge of the language.
    - Minimal knowledge of constructs.
  - Example, Java syntax is very close to C++, programmers can start programming almost immediately.
Consistency with Accepted Notations and Conventions:

- A programming language should be easy to learn and understand for the experienced programmer.

- This is one of your primary goals as Computer Scientists!
Language Design

- Consistency with Accepted Notations and Conventions:
  - Example:
    - Ignore white spaces where applicable.
    - Ignore blank lines.
  - Example: What does the following FORTRAN code do?
    - Do 99 I = 1.10
Language Design

- Consistency with Accepted Notations and Conventions:
  - Example:
    - Ignore white spaces where applicable.
    - Ignore blank lines.
  - Example: What does the following FORTRAN code do?
    - Do 99 I = 1.10
    - It actually assigns 1.1 to the variable Do99I !!!
    - Totally out of convention!
Language Design

- Consistency with Accepted Notations and Conventions:
  - The law of least astonishment:

    Things should not act or appear in completely unexpected ways.
Language Design

- Preciseness:
  - Also called definiteness.
  - It is the existence of a precise definition for a language so that:
    - The behavior of programs can be predicted.
    - Translators can be developed, and their behavior predicted.
Preciseness:

- Achieved by
  - Publication of language manuals.
  - Developing standards such as ANSI and ISO.

- The Algol68 designers for example developed manuals using many new terms to describe the language.

- The Algol68 reference manual was extremely difficult to understand, and language acceptance was lost.
Machine Independence:

- A language definition could attempt to be independent of any particular machine.

- Java uses the JVM to achieve machine independence.

- C has implementation defined constants in standard libraries such as “limits.h” and “float.h”

- Ada has many facilities to specify the precision of numbers within a program, and thus removes precision dependencies of particular machines.
Language Design

- Security:
  - Programs should not do unexpected damage.
  - Features like types, type checking, and variable declarations avoid unexpected damage.
  - Java uses security features in its JVM.
Language Design – C++ Case Study

- Designed by Bjarne Stroustrup
- C++ is not only a great success story, but also the best-documented language development effort in history:

Language Design – C++ Case Study

- OO features: class, inheritance
- Strong type checking for better compile-time debugging
- Efficient execution
- Portable
- Easy to implement
- Good interfaces with other tools
Language Design – C++ Case Study

- C compatibility (but not an absolute goal: no *gratuitous* incompatibilities)
- Incremental development based on experience.
- No runtime penalty for unused features.
- Multi-paradigm
- Stronger type checking than C
- Learnable in stages
- Compatibility with other languages and systems
Language Design – C++ Case Study

- Too big?
  - C++ programs can be hard to understand and debug
  - Not easy to implement
  - Defended by Stroustrup: multiparadigm features are worthwhile

- No standard library until late (and even then lacking major features)
  - Stroustrup agrees this has been a major problem