Language Evaluation Criteria

- Readability and Writability aspects
  *(Impact on software development methodologies)*
- Reliability
  * Type checking
  * Exception Handling
- Performance and cost
  * Compilation and Execution
  * Programmer Training and Maintenance

Readability/Writability issues

- Simplicity
- Consistency
- Orthogonality
- Extensibility
  *(support for abstraction)*

Simplicity

- Small number of core concepts and powerful combining forms.
  - E.g., Scheme vs Pascal
  - (define (cube n) (* n n n))
  - Uniform concise syntax
  - function cube (n: int) : int;
    begin return (n * n * n) end;
    The Scheme definition works for reals, while the Pascal definition does not.

Consistency

In Ada, A(1) stands for both procedure call and array element reference. In C, these different concepts have different syntax: A[1] and A[1].

What does the following program print?

```c
main()
{
  if (0 <= 5 <= 1)
    printf("%s
", "ABC");
  else
    printf("%s
", "EFG");
}
```

Orthogonality

The extent to which different language features can be composed in a free and uniform manner, with predictable effects and without limitations.

Contributes to simplicity because of fewer exceptions.

E.g., In Pascal, functions can only return primitive or unstructured type values, whereas, in Ada, composite or structured type values can be returned.
Eg., Add operation and Operand location

IBM Mainframe
A Reg, MemCell
AR Reg1, Reg2

VAX Computer
ADDL opd1, opd2
Reg or MemCell

Eg., Array index must begin with 0/1.
Eg., \texttt{vi/elmacs} editor commands

Superficially Different Constructs
Records Libraries
Subprograms Objects (instances of classes)

Inherent Structure Showing Relationships

<table>
<thead>
<tr>
<th>Record</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name-1 : Data</td>
<td>Name-1 : Func</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Name-n : Data</td>
<td>Name-n : Proc</td>
</tr>
<tr>
<td>Name1 : Data</td>
<td>Name-n : Func</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Name-m : Proc</td>
<td>Object</td>
</tr>
</tbody>
</table>

Orthogonality

Colors
- Red
- Yellow
- Blue
- Green
- Purple
- Black

Structuring the Colors

Layers of Abstraction

Extensibility
"Treat system-defined language primitives and user-defined entities similarly."

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>Procedure</td>
</tr>
<tr>
<td>Control Structure</td>
<td>Loops; Case/Switch;</td>
</tr>
<tr>
<td>Implementations of a Type</td>
<td>Specification of a Type</td>
</tr>
</tbody>
</table>

Reliability
Eg.,
\[
\text{DO17I} = 1,10 \\
\text{OR DO 17 I} = 1,10
\]
- The first is an assignment; the second is a loop.
- (According to folklore, the first space probe to Venus was lost because of this kind of typo.)
- \text{Declare before use.}

Eg., \texttt{.AND.} and \texttt{false}
- Incorrect grouping should be flagged as a type error, rather than processed using coercion.
Substitutivity

Null statement

```
S; S ≡ S
```

```
if B then S; end if
```

in Pascal

```
nul; S; nul; S;
```

in Ada


Also, `A + 3` ≡ `3 + A`


In C, `int foo[]` ≡ `int * foo`;

What does the following mean?

```
int (* (* foo[])())();
```

(use only postfix to improve readability.)

Further Examples


Also, `A + 3` ≡ `3 + A`