Language Design Principles

Language Design

- Most difficult and poorly understood area of computer science.
- Success or failure of a language depends upon complex interactions among the language mechanisms.
- Practical matters (Availability of the translator, Price and quality of the translator) not directly connected to language definition have a major effect on success or failure of a language.

Other factors affecting success or failure of a language

- Timing and markets
- Politics
- Geography

Reason for success

- C programming language has been a success at least partially because of the success of the UNIX operating system.
- COBOL though chiefly ignored by the computer science community, continues as a significant language because of its use in industry, and because of the large number of legacy applications.
- Ada success is because of its required use in certain U.S. Defense Department projects.
- Java and Python have achieved importance through the growth of internet.

Goals of design

Fortran:- Efficiency of execution.

COBOL:- English like non technical readability.

Algol60:- Block structured language to describe algorithm.

Pascal:- Simple language to promote top-down design.

C++:- Compatibility with c and real world objects.

Design Criteria

- Efficiency
- Regularity
- Simplicity
- Expressiveness
- Extensibility
- Restrictability
- Consistency
- Preciseness
- Machine Independence
- security

Efficiency

- Programming efficiency or Writability
- Readability
- Efficiency of target code
- Efficiency of translation
- Reliability
- Implementability
- maintainability

Efficiency

Writability

- How quickly and easily can programs be written in the language.

Efficiency

Readability

- The quality of a language that enables a programmer to understand and comprehend the nature of computation easily and accurately.

Efficiency

Efficiency of target code

- The language design should be such that the translator can generate efficient executable code.
- Eg:-static variable

Efficiency

Efficiency of translation

- The language design must permit the source code to be translated efficiently.

Eg:- one pass compiler

Efficiency

Reliability

- The assurance that a program will not behave in unexpected or disastrous way during execution.

Efficiency

Implementability

- The efficiency with which translators can be written.

Efficiency

Maintainability

- The ease with which errors can be found and corrected and new features can be added.

Regularity

- It expresses how the features of a language are integrated.

- Greater regularity means fewer unusual restrictions on the use of particular constructs.

Regularity

- Generality
- Orthogonality
- Uniformity

Regularity

Generality

- A language achieves generality by avoiding special cases in the use of constructs and by combining closely related constructs into a single more general one.
- Eg:- In C two structures or arrays cannot be compared directly using ‘==‘operator but must be compared element by element. So Equality operator lacks generality.

Regularity

Orthogonality

- Orthogonality in a programming language means that language constructs can be combined in any meaningful way and that the interaction of constructs or the context of use should not cause an unexpected restriction or behavior.

Regularity

Lack of Orthogonality

- In C and C++ array types cannot be returned from a function
- Local variables are defined at the beginning of the block in C
- C passes all parameters by value except arrays which are passed by reference.

Regularity

Uniformity

- This principle focuses on the consistency of appearance and behavior of language constructs.

Regularity

Non Uniformity

- Similar things do not look similar or behave similarly when they should.
- Dissimilar things actually look similar or behave similarly when they should not.
- Eg:-In C++ a semicolon is necessary after a class definition but forbidden after a function definition.

Simplicity

Regular languages need not be simple.

Eg:-algol68

Having very few basic constructs does not make language simple

Eg :-LISP and prolog

Language must be simple but oversimplicity can make a language cumbersome to use.

Eg:-pascal

Expressiveness

- Expressiveness is the ease with which a language can express complex processes and structures.
- Eg:-C++ real life situations

  LISP, Prolog and Algol68 are expressive

Extensibility

- User must be able to add features to a language. Such as,
- To be able to define a new data type
- Add new functions to a library
- Add keywords and constructs to the translator itself.

Restrictability

- With increasing size and complexity of languages Restrictability becomes as important as extensibility.
- In principle if a program does not use certain features of a language, then there should be no performance penalty as a result of those unused features.

Restrictability

- Eg:- a program in C++ that does not use exception handling should not run slower than an equivalent program in C (which does not have exception handling) simply because exception handling is available in C++.

Consistency with accepted Notions and conventions

- The language must follow the features and concepts that have become standard. Standard concepts such as program, function and variables should be clearly recognizable.
- Eg:-FORTRAN’s ignoring of blanks can cause major problems in readability and security.
- DO 99 I=1.10

Preciseness

- Preciseness is the existence of a precise definition for a language, so that the behavior of programs can be predicted.
- A precisely defined language will have more predictable translators.

Preciseness

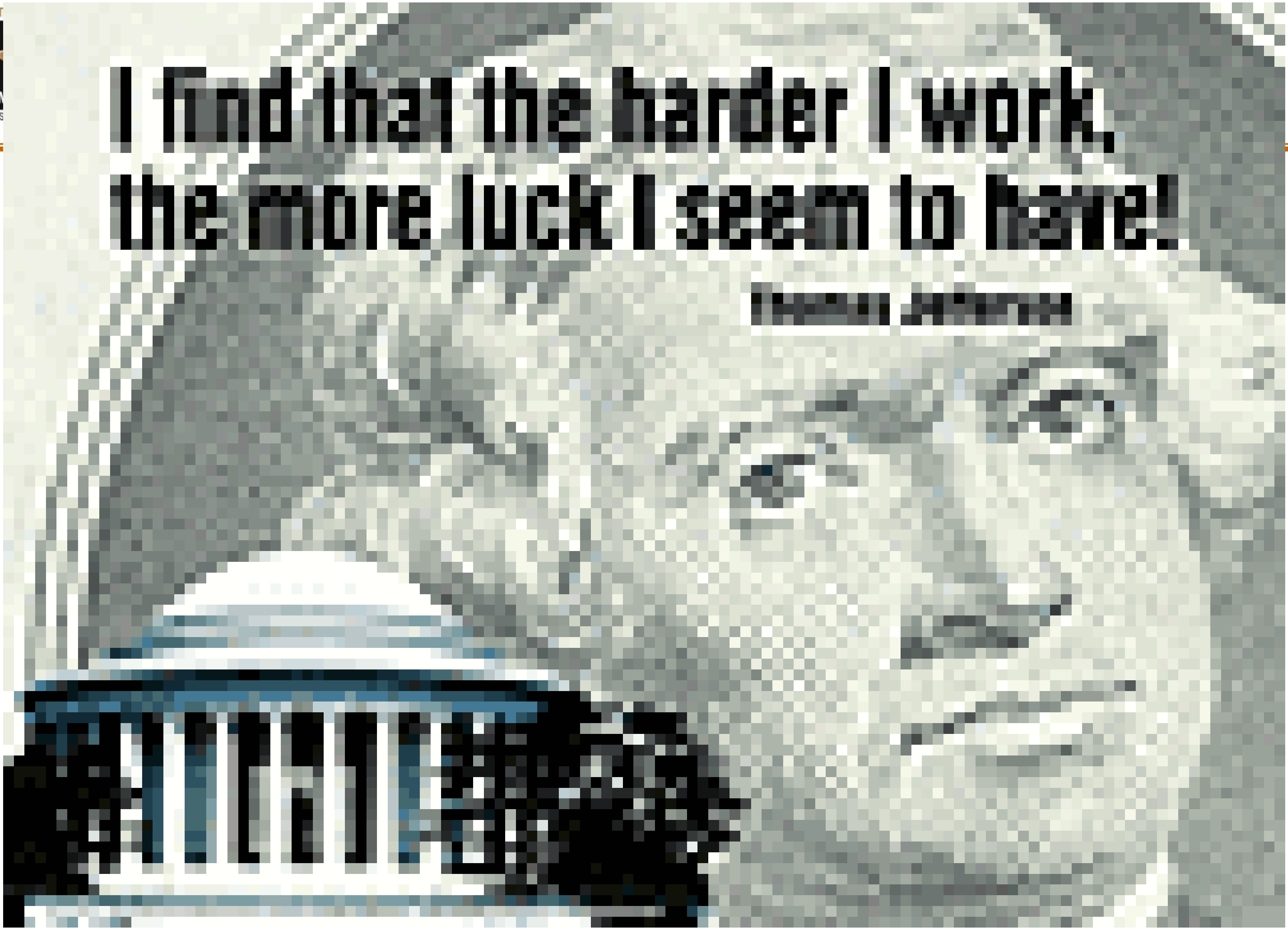
- Steps in achieving preciseness
- Publication of language manual
- Adoption of standard by a national or international standards organization such as ANSI or ISO.

Machine Independence

- A language must be independent of a particular machine. The primary method of achieving machine independence is the use of predefined data types.

Security

- Discourages programming errors and allows errors to be discovered and reported.
- Eg:-type checking and variable declarations.


I find that the harder I work,
the more luck I seem to have!
Thomas Jefferson

References

Text book

- Kenneth C. Louden “Programming Languages Principles and Practice” second edition Thomson Brooks/Cole Publication.

Reference Books:

- Terrence W. Pratt, Masvin V. Zelkowitz “Programming Languages design and Implementation” Fourth Edition Pearson Education.
- Allen Tucker, Robert Noonan “Programming Languages Principles and Paradigms second edition Tata MC Graw –Hill Publication.