GPCET
Pioneering Innovative Education

G. PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Accredited by NAAC with 'A' Grade of UGC, Approved by AICTE, New Delhi

Permanently Affiliated to JNTUA, Ananthapuramu

(Recognized by UGC under 2(f) and 12(B) & ISO 9001:2008 Certified Institution)

Nandikotkur Road, Venkayapalli, Kurnool – 518452

Department of Computer Science and Engineering

Bridge Course
On
Principles of Programming Language

By

K. Samunnisa

GPCET
Pioneering Innovative Education

Table of Contents

GPCET
Pioneering Innovative Education

1. Reasons for Studying of Programming Languages

1.1. Increased capacity to express ideas:

- People can easily express their ideas clearly in any language only when they have clear understanding of the natural language.
- Similarly, if programmers want to simulate the features of languages in another language, they should have some ideas regarding the concepts in other languages as well.

1.2. Improved background for choosing appropriate languages

- Many programmers when given a choice of languages for a new project, continue to use the language with which they are most familiar, even if it is poorly suited to the project.
- If these programmers were familiar with a wider range of languages, they would be better able to choose the language that includes the features that best address the characteristics of the problem at hand.

1.3. Increased ability to learn new languages

- In software development, continuous learning is essential.
- The process of learning a new programming language can be lengthy and difficult, especially for someone who is comfortable with only two or more languages.
- Once a thorough understanding of the fundamental concepts of languages is acquired, it becomes far easier to see how these concepts are incorporated into the design of the language being learned.

1.4. Better understanding the significance of implementation

- An understanding of implementation issues leads to an understanding of why languages are designed the way they are.
- This knowledge in turn leads to the ability to use a language more intelligently, as it was designed to use.
- We can become better programmers by understanding the choices among programming language constructs and consequences of those choices.

1.5. Better use of languages that are already known

- By studying the concepts of programming languages, programmers can learn about previously unknown and unused parts of the languages they already use and begin to use those features.

1.6. Overall advancement of computing

- There is a global view of computing that can justify the study of programming language concepts.
- For example, many people believe it would have been better if ALGOL 60 had displaced Fortran in the early 1960s, because it was more elegant and had much better control statements than Fortran. That it did not is due partly to the programmers and software development managers of that time, many of whom did not clearly understand the conceptual design of ALGOL 60.
- If those who choose languages were better informed, perhaps, better languages would eventually squeeze out poorer ones.

2. Different Programming Domains

2.1. Scientific applications

- Large number of floating point computations. The most common data structures are arrays and matrices; the most common control structures are counting loops and selections
- The first language for scientific applications was Fortran, ALGOL 60 and most of its descedants
- Examples of languages best suited: Mathematica and Maple

2.2. Business applications

- Business languages are characterized by facilities for producing reports, precise ways of describing and storing decimal numbers and character data, and ability to specify decimal arithmetic operations.
- Use decimal numbers and characters
- COBOL is the first successful high-level language for those applications.

2.3. Artificial intelligence

- Symbols rather than numbers are typically manipulated
- Symbolic computation is more conveniently done with linked lists of data rather than arrays

- This kind of programming sometimes requires more flexibility than other programming domains
- First AI language was LISP and is still most widely used
- Alternate languages to LISP are Prolog – Clocksin and Mellish

2.4. **Systems programming**

- The operating system and all of the programming support tools of a computer system are collectively known as systems software
- Need for efficiency because of continuous use
- Low-level features for interfaces to external devices
- C is extensively used for systems programming. The UNIX OS is written almost entirely in C

2.5. **Web software**

- Markup languages
  - Such as XHTML
- Scripting languages
  - A list of commands is placed in a file or in XHTML document for execution
  - Perl, JavaScript, PHP

2.6. **Special-purpose languages**

- RPG – business reports
- APT – programmable machine tools
- GPSS – simulation

3. Language Evaluation Criteria

The following factors influences Language evalution criteria

1) Readability
2) Simplicity
3) Orthogonality
4) Writability
5) Reliability
6) Cost
7) Others

3.1. Readability

- One of the most important criteria for judging a programming language is the ease with which programs can be read and understood.
- Language constructs were designed more from the point of view of the computer than of computer users
- From 1970s, the S/W life cycle concept was developed; coding was relegated to a much smaller role, and maintenance was recognized as a major part of the cycle, particularly in terms of cost. Because ease of maintenance is determined in large part by readability of programs, readability became an important measure of the quality of programs

3.2. Overall simplicity

- Language with too many features is more difficult to learn
- Feature multiplicity is bad. For example: In Java, increment can be performed if four ways as:
- Count= count+1
- Count+=1
- Count++
- ++count
- Next problem is operator overloading, in which single operator symbol has more than one meaning

3.3. Orthogonality

- A relatively small set of primitive constructs that can be combined in a relatively small number of ways
- Consistent set of rules for combining constructs (simplicity)
- Every possible combination is legal
- For example, pointers should be able to point to any type of variable or data structure
- Makes the language easy to learn and read
- Meaning is context independent
- VAX assembly language and Ada are good examples
- Lack of orthogonality leads to exceptions to rules
- C is littered with special cases

- E.g. - **struct**s can be returned from functions but arrays cannot

Orthogonolity is closely related to simplicity. The more orthogonal the design of a language, the fewer exceptions the language rules require. Fewer exceptions mean a higher degree of regularity in the design, which makes the language easier to learn, read, and understand.

- Useful control statements
- Ability to define data types and structures
- Syntax considerations

• Provision for descriptive identifiers : – BASIC once allowed only identifiers to consist of one character with an optional digit

• Meaningful reserved words

• Meaning should flow from appearance

3.4. Writability

- Most readability factors also apply to writability
- Simplicity and orthogonality
- Control statements, data types and structures
- Support for abstraction
- Data abstraction
- Process abstraction
- It is easy to express program ideas in the language
- APL is a good example
- In C, the notation C++ is more convenient and shorter than C = C + 1
- The inclusion of for statement in Java makes writing counting loops easier than the use of while

3.5. **Reliability**

A program is said to be reliable if performs to its specifications under all conditions.

- Type checking

  • Type checking is simply testing for type errors in a given program, either by the compiler or during the program execution

• Because run time type checking is expensive, compile time type checking is more desirable

• Famous failure of space shuttle experiment due to **int / float** mix-up in parameter passing

- Exception handling
  - • Ability to intercept run-time errors
- Aliasing
  - • Ability to use different names to reference the same memory
  - • A dangerous feature
- Readability and writability both influence reliability

3.6. Cost

- Training programmers to use language
- Writing programs in a particular problem domain
- Compiling programs
- Executing programs
- Language implementation system (free?)
- Reliability
- Maintaining programs

3.7. Others: portability, generality, well-definedness

4. Different types Implementation Methods for programming languages.

We have three different types of implementation methods . They are

- Compilation – Programs are translated into machine language
- Pure Interpretation – Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems – A compromise between compilers and pure interpreters

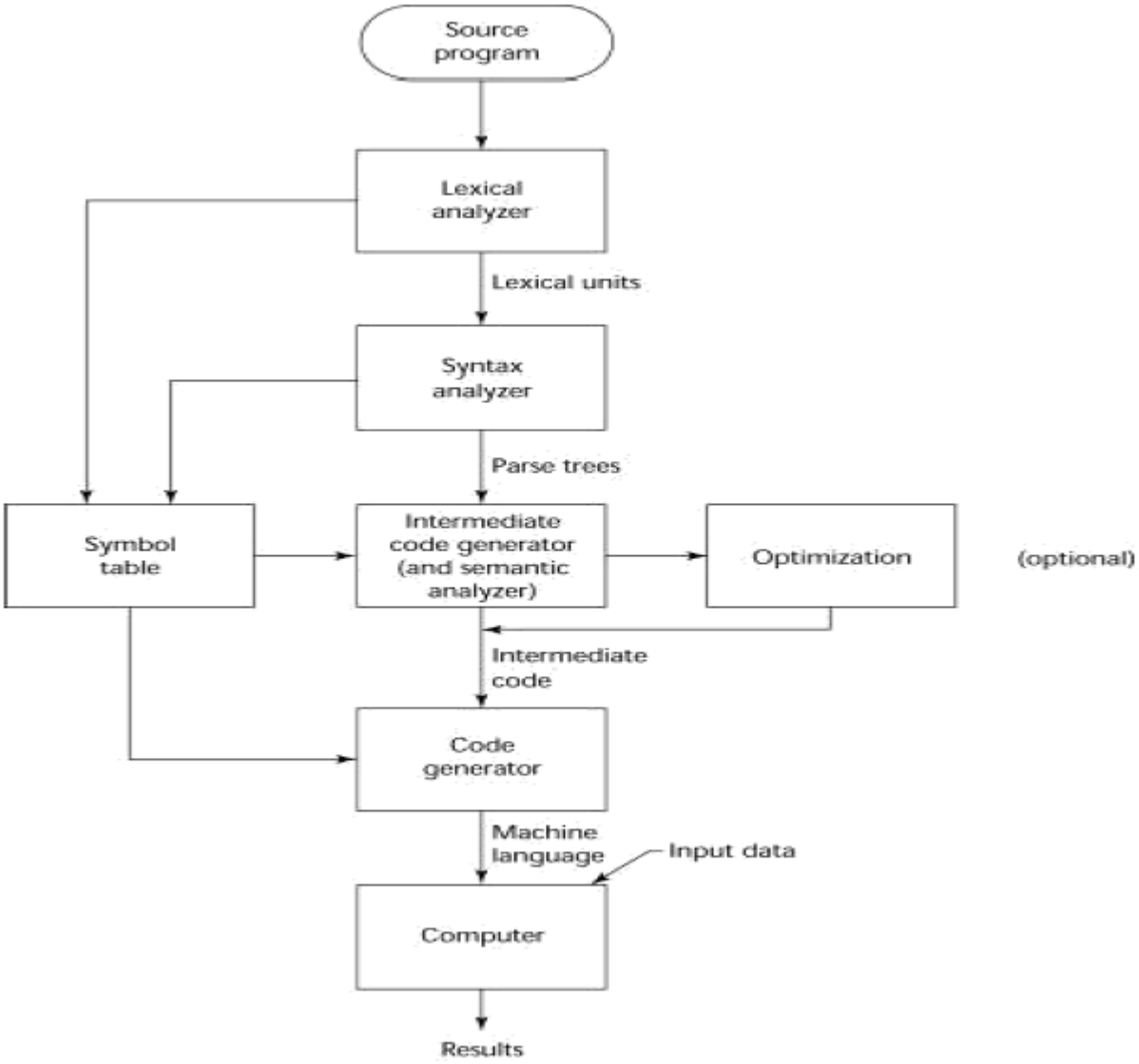
4.1. The Compilation Process

Translate high-level program (source language) into machine code (machine language)

• Slow translation, but fast execution.

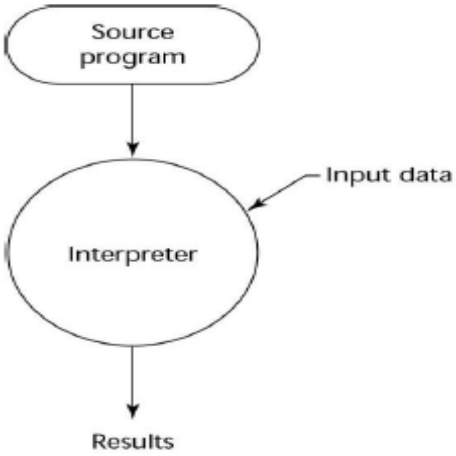
Translates whole source code into object code at a time and generate errors at the end . If no errors ,generates object code. The object code after linking with libraries generates exe code. User input will be given with execution.

Ex: C++ is a compiler



4.2. Pure Interpretation Process Ttranslation of source code line by line so that generates errors line by line.If no errors in the code generates object code. While interpretation it selef user input will be given.

- Immediate feedback about errors
- Slower execution
- Often requires more space



• Used mainly for scripting languages

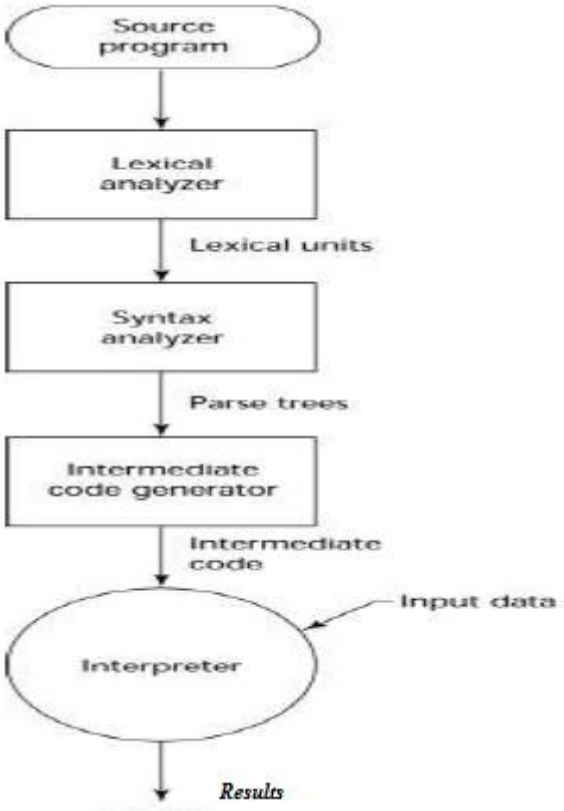
Example for interpreter is Dbase III plus

4.3. Hybrid Implementation Process

A compromise between compilers and pure interpreters

• A high-level language program is is translated to an intermediate language that allows easy interpretation

• Faster than pure interpretation



Example for hybrid implementation is Java. Java is a compiled interpreted language.