# SE Computer- Division A

**Course Name :Principles of Programming  Languages**

**Course Code: 210255**

**Subject InCharge: Mrs.Savita Mane**

# Unit 1 Fundamentals of Programming

Importance of Studying Programming Languages, History of Programming Languages, Impact of Programming Paradigms, Role of Programming Languages, Programming Environments.

Impact of Machine Architectures: The operation of a computer, Virtual Computers and Binding Times.

Programming paradigms- Introduction to programming paradigms, Introduction to four main Programming paradigms- procedural, object oriented, functional, and logic and rule based.

| Topic | Book To Refer |
| --- | --- |
| **Importance of Studying Programming Languages, History of Programming Languages, Impact of Programming Paradigms, Role of Programming Languages, Programming Environments. Impact of Machine Architectures: The operation of a computer, Virtual Computers and Binding Times.** | **T. W. Pratt, M. V. Zelkowitz, "Programming Languages Design and Implementation‖, 4th Ed**, PHI, ISBN 81-203-2035-2. |

| Topic | Book To Refer |
| --- | --- |
| **Programming paradigms- Introduction to programming paradigms, Introduction to four main Programming paradigms- procedural, object oriented, functional, and logic and rule based.** | **T. W. Pratt, M. V. Zelkowitz, "Programming Languages Design and Implementation‖, 4th Ed**, PHI, ISBN 81-203-2035-2. **1.4.2 Page No.28 - 30** |

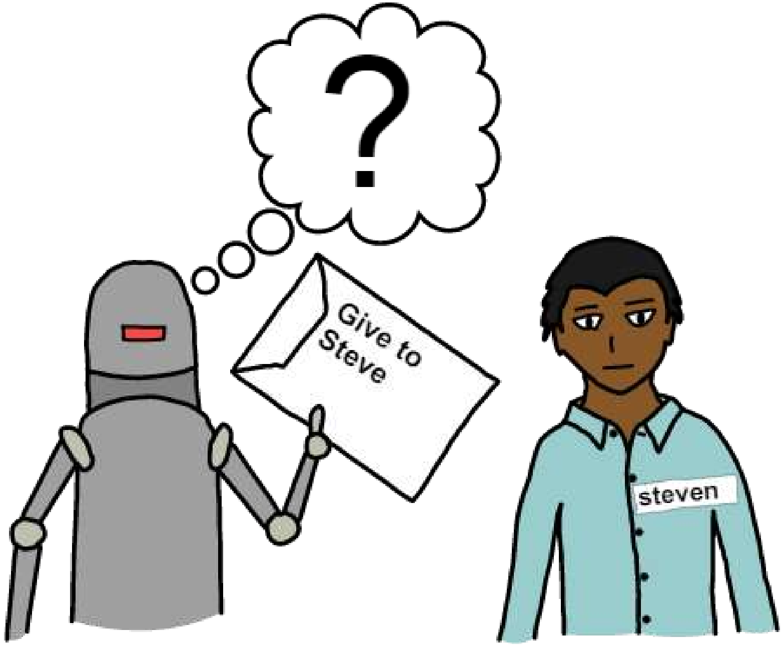
## Importance of Studying Programming Languages

Computers aren't very smart

Ask the computer to draw a picture of bird…

Will it draw?

## Importance of Studying Programming Languages

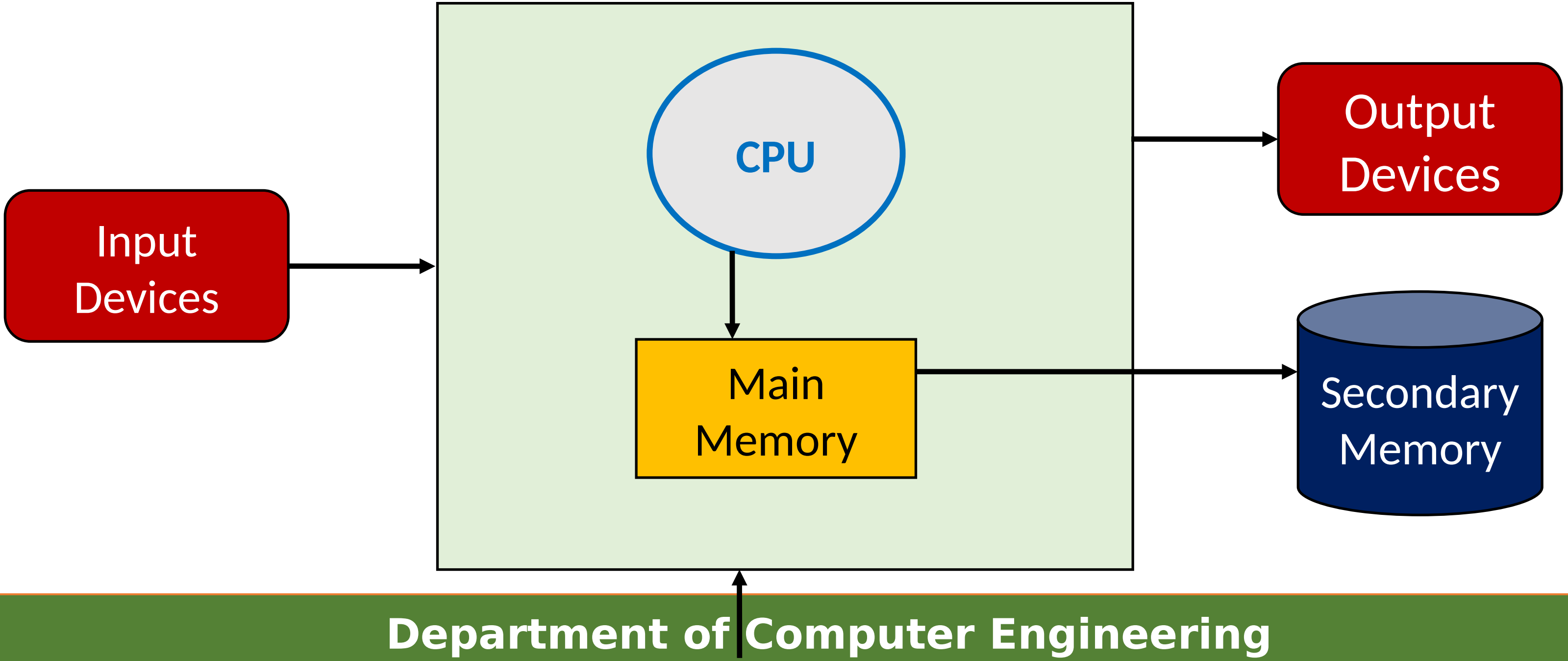



Computers are bad at understanding things

## Importance of Studying Programming Languages


Computers cannot understand English

# Functional View of a Computer



# Functional View of a Computer

E.g., Keyboard and mouse

Input Devices

- Humans interact with **computers via Input and Output (IO) devices**
- Information from Input devices are processed by **the CPU and may be shuffled off to the main or secondary memory**
- When information need to be displayed, **the CPU sends them to one or more Output devices**

E.g., Monitor

Output Devices

- A *program* is just a sequence of instructions telling the computer what to do
- Obviously, we need to provide these instructions in a language that computers can understand
  - We refer to this kind of a language as a *programming language*
  - Python, Java, C and C++ are examples of programming languages
- Every structure in a programming language has an exact form (i.e., *syntax*) and a precise meaning (i.e., *semantic*)

## Six reasons to Learn Programming Languages

- To improve your ability to develop effective algorithms
- To improve your use of your existing programming language
- To increase your vocabulary of useful programming constructs
- To allow a better choice of programming language
- To make it easier to learn a new language
- To make it easier to design a new language

## Reason 1:

**To improve your ability to develop effective algorithms**

- **The depth at which people can think is heavily influenced by the expressive power of their language.**
- **It is difficult for people to conceptualize structures that they cannot describe, verbally or in writing.**

## Reason 2:

To improve your use of your existing programming language

- Many professional programmers have a limited formal education in computer science, limited to a small number of programming languages.
- They are more likely to use languages with which they are most comfortable than the most suitable one for a particular job.

## Reason 3

- **Computer science is consider as a young discipline and most software technologies (design methodology, software development, and programming languages) are not yet mature. Therefore, they are still evolving.**
- **The understanding of programming language design and implementation makes it easier to learn new languages.**

## Reason 4

- It is often necessary to learn about language implementation; it can lead to a better understanding of why the language was designed the way that it was.
- Fixing some bugs requires an understanding of implementation issues.

## Reason 4

- **Some languages are better for some jobs than others.**
  - **(i) FORTRAN and APL for calculations, COBOL and RPG for report generation, LISP and PROLOG for AI, etc.**
- **Improve your use of existing programming language**
- **By understanding how features are implemented, you can make more efficient use of them.**

## Reason 5

- To improve your use of existing programming language
- By understanding how features are implemented, you can make more efficient use of them.
- Examples:
- Creating arrays, strings, lists, records.
- Using recursions, object classes, etc.

## Reason 6

- **Designing a new language require prior knowledge of previous one to make it effective, efficient and convenient to users.**
- **The previous knowledge as well as concepts are usual to design a new language irrespective of their work domains.**

## History of Programming languages

**Click Here for Video**

# Unit 1 Fundamentals of Programming

## History of Programming languages

**https://www.quiz-maker.com/QKM3653ME**

## History of Programming languages

- Development of Early Language
- Evolution of Software Architecture
- Application Domains

## History of Programming languages

- 1951- 55: Experimental use of expression compilers.
- 1956- 60: FORTRAN, COBOL, LISP, Algol 60.
- 1961- 65: APL notation, Algol 60 (revised), SNOBOL, CPL.
- 1966- 70: APL, SNOBOL 4, FORTRAN 66, BASIC, SIMULA, Algol 68, Algol-W, BCPL.
- 1971- 75: Pascal, PL/1 (Standard), C, Scheme, Prolog.
- 1976- 80: Smalltalk, Ada, FORTRAN 77, ML, C++.

## History of Programming languages

- 1981- 85: Smalltalk-80, Prolog, Ada 83.
- 1986- 90: SML, Haskell.
- 1991- 95: Ada 95, TCL, Perl.
- 1996- 2000: Java.
- 2000- 05: C#, Python, Ruby, Scala.

**Development of Early Language**

**Numerically based languages**

| FORTRAN, Algol, Pascal, PL/1 | BASIC, C, C++ |
|---|---|

↓

**Business languages**

| COBOL (Common Business Oriented Language) | English-like notation |
|---|---|

↓

**Artificial intelligence languages**

| Tree search; Rule-based paradigm | LISP (LISt Processing), Prolog (Programming in Logic) |
|---|---|

↓

**System languages**

| C | C++ |
|---|---|

↓

**Script languages:**

| Java Script | AWK, Perl |
|---|---|

↓

**Web programming**

| HTML, XML | Java, Microsoft *.NET family |
|---|---|

## Development of Early Language

- **Numerically based languages**

    Computing mathematical expressions

    FORTRAN, Algol, Pascal, PL/1, BASIC, C, C++

- **Business languages**

    COBOL (Common Business Oriented Language)

    English-like notation

## Development of Early Language

- **Artificial intelligence languages**

  Tree search; Rule-based paradigm

  LISP (LISt Processing)

  PROLOG (PROgramming in LOGic)

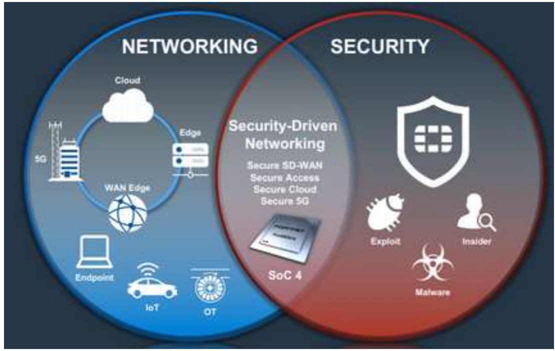
- **System languages :**C, C++
- **Script languages:** AWK, Perl, TCL/TK
- **Web programming**: HTML, XML, Java, Microsoft *.NET family

## Evolution of Software Architecture

MainFrame Era

Personal Computer

Networking Era




## Evolution of Software Architecture

- **Mainframe Era**
  - Batch processing (batches of files)
  - Interactive processing (time sharing)
- **Effects on language design**
  - File I/O in batch processing
  - Error handling in batch processing
  - Time constraints in interactive processing

## Evolution of Software Architecture

- **Personal Computer**
    - Interactive processing
    - Embedded system environments

**Effects on language design**

- No need for time sharing
- Good interactive graphics
- Non-standard I/O devices for embedded systems

## Evolution of Software Architecture

- **Networking Era**
    - Client-server model of computing
    - Server: a program that provides information
    - Client - a program that requests information

**Effects on language design**

- Interaction between the client and server programs
- Active web pages, Security issues, Performance

## Application Domain

- Business Processing
- Scientific
- System
- Artificial Intelligence
- Publishing
- Process

| Era | Application | Major languages | Other languages |
|---|---|---|---|
| 1960s | Business | COBOL | Assembler |
| | Scientific | FORTRAN | ALGOL, BASIC, APL |
| | System | Assembler | JOVIAL, Forth |
| | Artificial intelligence | LISP | SNOBOL |
| Today | Business | COBOL, C++ , Java, spreadsheet | C, PL/I, 4GLs |
| | Scientific | FORTRAN, C, C++ , Java | BASIC |
| | System | C, C++ , Java | Ada, BASIC, Modula |
| | Artificial intelligence | LISP, Prolog | |
| | Publishing | TeX, Postscript, word processing | |
| | Process | UNIX shell, TCL, Perl, Javascript | AWK, Marvel, SED |
| | New paradigms | ML, Smalltalk | Eiffel |

**Table 1.1.** Languages for various application domains.

## Impact of Programming Paradigms

This Causes Impact on

- **Way Programmers are Solving the Program**
- Challenge to Describe Needs of the Stakeholders and Solution Requirements
- **Provided an Underlying Model to Verify and Validate** the Program in a Reliable Manner.
- **Minimize the Design Errors**
- **Provides Variety of Techniques to manage Complexity.**
- How to Design a Software

## Influences in Design of Programming Languages

| | |
|---|---|
| Computer Capabilities | • According to Size, OS what features of PL be used |
| Applications | • According to Application- Scientific, Internet etc. |
| Programming Method | • Large , Complex Programs |
| Implementation Methods | • Divide and Conquer, Structured etc. |
| Theoretical Studies | • Research in Conceptual Foundations |
| Standardization | • Easily be Implemented on Variety of Systems |

## Some influences on the development of Programming Languages

| Years | Influences and New Technology |
|---|---|
| 1951–55 | **Hardware:** Vacuum-tube computers; mercury delay line memories<br>**Methods:** Assembly languages; foundation concepts: subprograms, data structures<br>**Languages:** Experimental use of expression compilers |
| 1956–60 | **Hardware:** Magnetic tape storage; core memories; transistor circuits<br>**Methods:** Early compiler technology; BNF grammars; code optimization; interpreters; dynamic storage methods and list processing<br>**Languages:** FORTRAN, ALGOL 58, ALGOL 60, LISP |
| 1961–65 | **Hardware:** Families of compatible architectures; magnetic disk storage<br>**Methods:** Multiprogramming operating systems; syntax-directed compilers<br>**Languages:** COBOL, ALGOL 60 (revised), SNOBOL, JOVIAL |
| 1966–70 | **Hardware:** Increasing size and speed and decreasing cost; microprogramming; integrated circuits<br>**Methods:** Time-sharing systems; optimizing compilers; translator writing systems<br>**Languages:** APL, FORTRAN 66, COBOL 65, ALGOL 68, SNOBOL4, BASIC, PL/I, SIMULA 67, ALGOL-W |

## Some influences on the development of Programming Languages

| | |
|---|---|
| 1971–75 | **Hardware:** Minicomputers; small mass storage systems; semiconductor memories<br>**Methods:** Program verification; structured programming; software engineering<br>**Languages:** Pascal, COBOL 74, PL/I (standard), C, Scheme, Prolog |
| 1976–80 | **Hardware:** Microcomputers; mass storage systems; distributed computing<br>**Methods:** Data abstraction; formal semantics; concurrent, embedded, and real-time programming techniques<br>**Languages:** Smalltalk, Ada, FORTRAN 77, ML |
| 1981–85 | **Hardware:** Personal computers; workstations; video games; local-area networks; ARPANET<br>**Methods:** Object-oriented programming; interactive environments; syntax-directed editors<br>**Languages:** Turbo Pascal, Smalltalk-80, use of Prolog, Ada 83, Postscript |

## Some influences on the development of Programming Languages

| | |
|---|---|
| 1986–90 | **Hardware:** Age of microcomputer; engineering workstation; RISC architectures; Internet<br>**Methods:** Client/server computing<br>**Languages:** FORTRAN 90, C++, SML (Standard ML) |
| 1991–95 | **Hardware:** Very fast inexpensive workstations and microcomputers; massively parallel architectures; voice, video, fax, multimedia<br>**Methods:** Open systems; environment frameworks<br>**Languages:** Ada 95, Process languages (TCL, PERL), HTML |
| 1996-2000 | **Hardware:** Computers as inexpensive appliances; Personal digital assistants; World wide web; Cable-based home networking; Gigabyte disk storage<br>**Methods:** E-commerce<br>**Languages:** Java, Javascript, XML |

## Role of Programming Languages

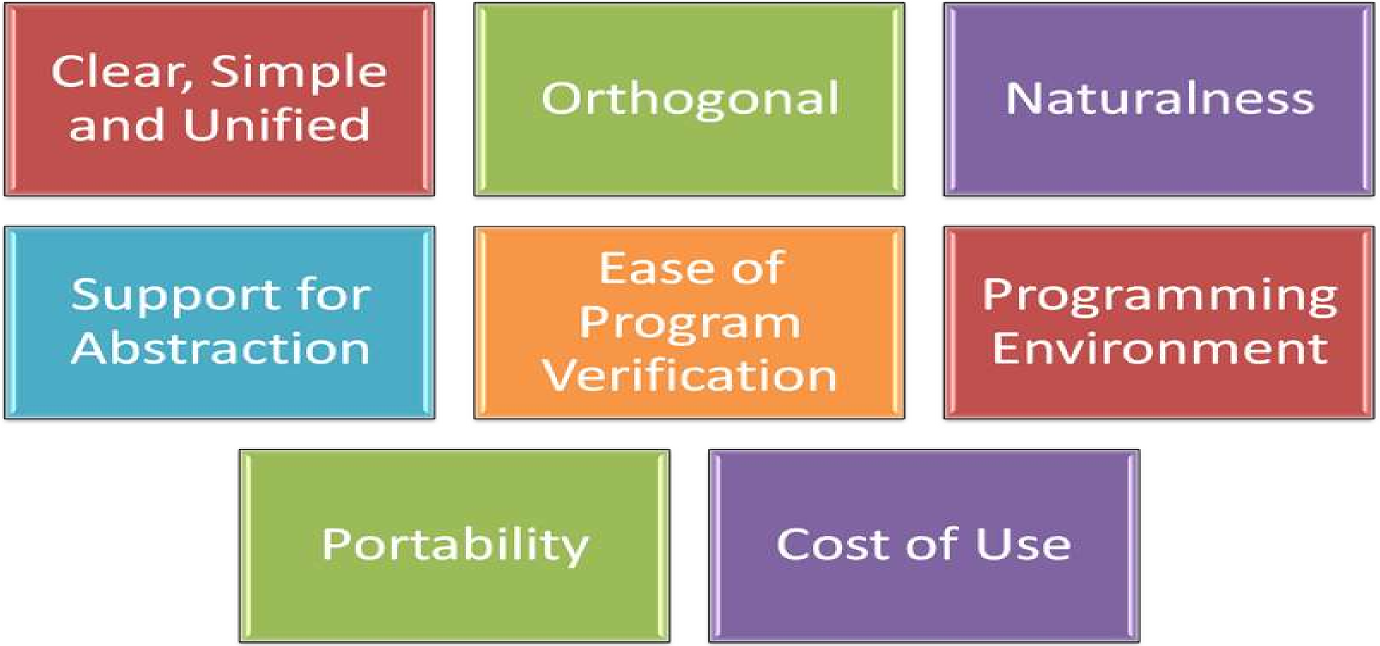
What Makes a Good Language?

Language Paradigms

Language Standardization

Internationalization

## Good Language



## Clarity, Simplicity And Unity

- A Programming language provides **both a conceptual framework for Algorithm planning and means of expressing them**.
- It should provide a clear, simple and unified set of concepts that can be used as primitives in developing algorithms.
- It should have

• It has minimum number of different concepts
• with Rules for their combination being
• simple and regular.

This attribute is called conceptual integrity.

## Orthogonality

- It is one of the most important feature of PL orthogonality is the property that means " Changing A does not change B".
- If I take Real world example of an orthogonal system Would be a radio, where changing the station does not change the volume and vice versa.
- When the features of a language are orthogonal, language **is easier to learn and programs are easier to write** because only few exceptions and special cases to be remembered.

## Naturalness for the application

- Language should provide appropriate data structures, operations, control structures and proper natural syntax

## Support for Abstraction

There is always found that a substantial gap remaining between the abstract data structure and operations that characterize the solution to a problem and their particular data structure and operations built into a language.

## Programming Environment

- An appropriate programming environment adds an extra utility and make language to be implemented easily like
- **The availability of- Reliable- Efficient - Well documentation Speeding up creation and testing by-special Editors- testing packages**
- Facility- Maintaining and Modifying- Multi Version of program software product.

# Programming Environment


**Programming Environment:**

“A Programming Environment is the collection of tools used in the development of software.”

- In a general sense, a programming environment combines hardware and software that allows a developer to build applications.
- Developers typically work in integrated development environments or IDEs.
- These connect users with all the features necessary to write and test their code correctly.
- Different IDEs will offer other capabilities and advantages.

## Programming Environment

**What is an IDE?**

An **I**ntegrated **D**evelopment **E**nvironment integrates common development tools in single software environment.

**An IDE normally consists of at least:-**

- File system
- Text editor
- Linker
- Compiler
- Integrated tools

## Programming Environment

**IDEs includes features/tools like:**

- Debugging
- Syntax highlighting
- Code completion
- Language support
- Code search
- Refactoring
- Version control
- Visual programming

## Ease of program verification:- Reusability

- The reusability of program written in a language is always a central concern. A program is checked by various testing technique like
- Formal verification method Desk checking Input output test checking.
- We verify the program by many more techniques.
- A language that makes program verification difficult maybe far more troublesome to use.
- Simplicity of semantic and syntactic structure is a primary aspect that tends to simplify program verification.

## Syntax and Semantics

- The syntax of programming language is what the program looks like.
- How statements declaration and other constructs are written
- The semantic of Pl is meaning is a meaning given to the various syntactic constructors.

```
int V[10];          V: array[0..9] of integer;
```

## Role of Programming Languages

What Makes a Good Language?

Language Paradigms

Language Standardization

Internationalization

## Language Paradigms

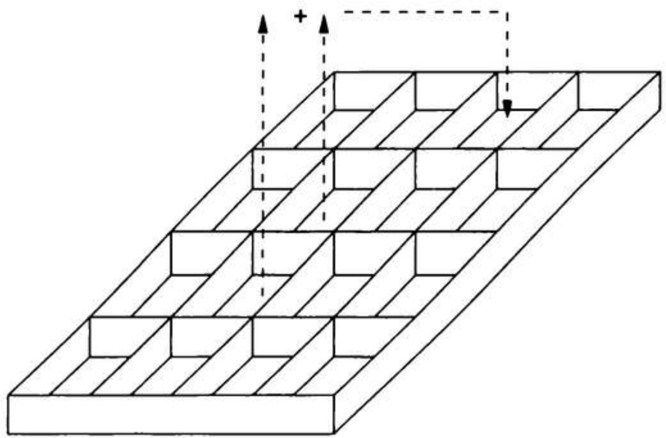
- Imperative language
- Applicative language
- Rule based language
- Object oriented language

## Imperative Languages

- Command driven or statement oriented
- The basic concept is machine state
- A program consists of sequences of statement
- Execution of each instruction causes the computer to change the value of one or more location , to enter a new state.
- Syntax of such languages

$$statement_1;$$
$$statement_2;$$
$$\ldots$$

## Imperative Languages




(a) Imperative languages - Memory is a set of boxes

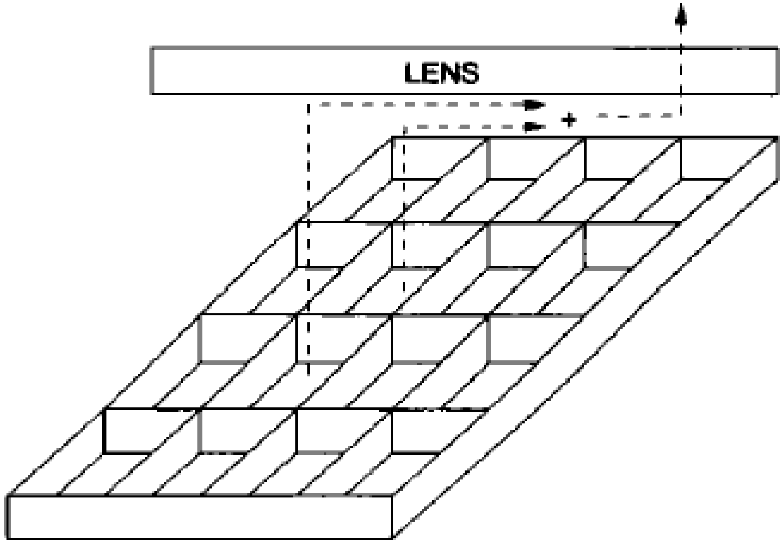
- Many Widely used Languages C, C++, FORTRAN, PL/I, Pascal , Ada, Small Talk and COBOL support this model.
- Most of the all conventional languages

## Applicative Languages

- Programming language is **to look at the function** that the program represents **rather than just the state changes as the program executes**, statement by statement.
- Focus on the **desired result rather than at the available data.**
- **What is the function that must be applied to the initial machine state by accessing the initial set of variables and combining them in specific ways to get an answer?**
- The languages which emphasize this view are called applicative or functional languages.

$$function_n(\ldots function_2(function_1(data))\ldots)$$

## Applicative Languages




(b) Applicative languages -
Change how data is accessed from memory

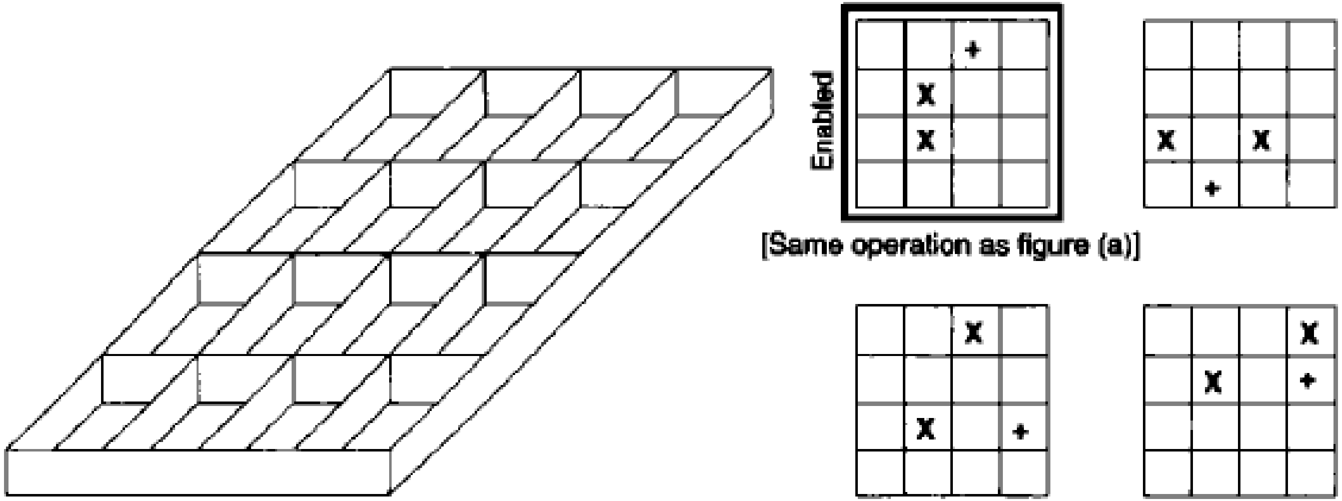
- LISP and ML are two functional languages

## Rule based Languages

- Execute by checking for **the presence of a certain enabling condition** and, when present, **executing an appropriate action.**
- The most common rule-based language is **Prolog, also called a logic programming**

$$
\begin{aligned}
&\textit{enabling condition}_1 \rightarrow \textit{action}_1\\
&\textit{enabling condition}_2 \rightarrow \textit{action}_2\\
&\dots\\
&\textit{enabling condition}_n \rightarrow \textit{action}_n
\end{aligned}
$$

## Rule based  Languages




(c) Rule-based -
Use filters to enable state change

Each grid is a possible filter
signifying operation and operands

## Object Oriented Languages

- **Complex data objects are built**, then a **limited set of functions** are designed to operate on those data.
- **Complex objects** are designed as e**xtensions of simpler objects**, inheriting properties of the simpler object.
- The best of two of the other computational models.
    - By building **concrete data objects, an object-oriented program** gains the efficiency of **imperative languages**.
    - By building **classes of functions** that use a **restricted set of data objects**, we build the flexibility and reliability of the applicative model.

## Generality of Computational Model

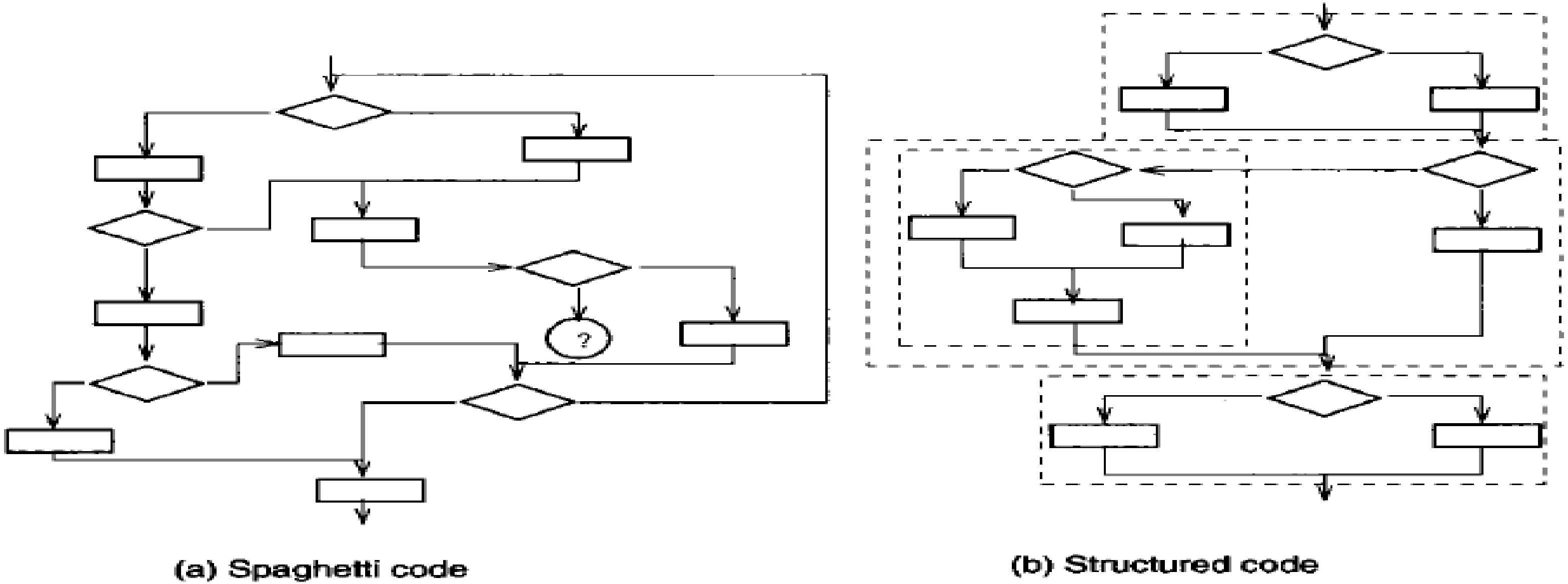



**Figure 1.2.** Applicative techniques in imperative languages.

## Role of Programming Languages

What Makes a Good Language?

Language Paradigms

Language Standardization

Internationalization

## Language standardization

- **What describes a programming language?**
- int i; i = (1 && 2) + 3 , **Is it valid Statement in C?**
- **To have an answer to this we usually follow following approaches (Next Slide)**

## Language standardization

1. Read the definition in the language reference manual to decide what the statement means.
2. Write a program on your local computer system to see what happens.
3. Read the definition in the language standard.

## Language standardization

To address these concerns, **most languages have standard definitions.**

1. **Proprietary standards**.
   a. **These are definitions by the company that develops and owns the language.**
   b. **Do not work for languages that have become popular and widely used**.
   c. Variations in implementations soon appear with many enhancements and incompatibilities.

2. **Consensus standards.**
   d. These are documents produced by **organizations based on an agreement** by the relevant participants.
   e. **Consensus standards**, or simply standards, are the major method to ensure uniformity among several implementations of a language.

## Language standardization

| Proprietary | Consensus |
|---|---|
| | HTML , C#, WWW |
| Need License | No License |

## Language standardization

To use standards effectively, we need to address three issues:

1. **Timeliness : When** do we standardize a language?
2. **Conformance :** What does it mean for a program **to adhere to a standard and for a compiler to compile a standard?**
3. **Obsolescence:** When does a standard age, and **how does it get modified?**

## Language standardization

To use standards effectively, we need to address three issues:

**Timeliness : When** do we standardize a language?

- One would like to standardize a language early enough so **that there is enough experience in using the language, yet not so late** as to encourage **many incompatible implementations.**


## Language standardization

To use standards effectively, we need to address three issues:

- **Conformance :** What does it mean for a program **to adhere to a standard and for a compiler to compile a standard?**
- A program is conformant **if it only uses features defined in the standard.**
- A conforming compiler is one that, **when given a conformant program, produces an executable program** that produces the **correct output.**


## Language Internationalization

To use standards effectively, we need to address three issues:

**Obsolescence:** When does a standard age, and **how does it get modified?**

- **S**tandards have to be reviewed every 5 years and either be renewed or dropped.
- The 5-year cycle often gets stretched out somewhat, but the process is mostly effective
- Problem with updating a standard is **what to do with the existing collection of programs** written for the older standard
- most standards require backward compatibility; the new standard must include older versions of the language.

## Role of Programming Languages

What Makes a Good Language?

Language Paradigms

Language Standardization

Internationalization

## Language Internationalization

- **Collating sequences** : In what collating sequence should the characters be ordered?
  - **Sorting.** The position of non-Roman characters, such as A, @, B, 3, and others s not uniformly defined and may have different interpretations in different countries.
  - **Case.** Some languages like Japanese, Arabic, Hebrew, and Thai have no uppercase—lowercase distinction.
  - **Scanning direction.** Most languages read from left to right, but others exist (e.g., right to left, top to bottom).

## Language Internationalization

- **Country-specific date formats.**

  11/26/02 in the United States is 26/11/02 in England; 26.11.02 in France; 26-XI-02 in Italy, etc.

- **Country-specific time formats**.

  5:40 p.m. in the United States is 17:40 in Japan, 17.40 in Germany, 17h40 in France,and so on.

## Language Internationalization

- **Time zones.**
  - Although the general rule is 1 **hour of change for each 15 degrees of longitude,** it is more a guideline than a reality.
  - Time zones are generally an integral number of hours apart, but some vary by 15 or 30 minutes.
  - Time changes (e.g., daylight savings time in the United States and summer time in Europe) do not occur uniformly around the world.
  - Translating local time into a worldwide standard time is nontrivial.

## Language Internationalization

- **Ideographic systems.**

  Some written languages are not based on a small number of characters forming an alphabet, but instead use large numbers of ideographs (e.g., Japanese, Chinese, and Korean).

  Often 16 bits might be needed to represent text in those languages.

- **Currency**.

  Representation of currency (e.g., $, £, ¥) varies by country.

# Programming Environment

- Effects on Language Design
- Environment Frameworks
- Job Control and Process Languages

## Programming Environment-Effects on Language Design

- it is ordinarily desirable to have **different** programmers or programming groups **design, code, and test parts** of a program before final assembly of all components into a complete program.
- **Language must be structured so that subprograms or other parts can be separately compiled and executed, later merged without change into the final program.**
- **Compiler may need information about other subprograms or shared data**
  - The specification of the number, order, and type of parameters expected by any subprogram
  - to determine the storage representation of the external variable
  - The definition of a data type that is defined externally but is used to declare any local variable within the subprogram

## Programming Environment-Effects on Language Design

- Another aspect of separate compilation that affects language design
  - shared name
  - scoping rules
  - Inheritance
- **Testing and debugging.**
  - Execution trace features.
  - Breakpoints.
  - Assertions.

**assert( X>0 and A=1) or (X=0 and A>B+10).**

## Programming Environment- Environment Frameworks

- Support environment consists of infrastructure services called the **environment framework to manage development of Program.**
- Supplies services such as a data repository, graphical user interface, security, and communication services.
- Languages are sometimes designed to allow for easy access to these infrastructure services.

## Programming Environment- Job Control and Process Languages

- **Click and Execute Environment**
- If the c**ompilation fails, the user could invoke an editor to correct the Program;**
- If the **compilation succeeds, the user could invoke a loader and execute the program.**
- a process or scripting language which generally interpret and have the property that they view programs and files as the primitive data to manipulate.
- **"faster, better, and cheaper."**

In developing a Language architecture of s/w influences the design of language

## Impact of Machine Architectures

### THE OPERATION OF A COMPUTER

- **Computer Hardware** : Data, Operations, Sequence Control, Data Access, Storage Management Operating Environment, Alternative computer architectures, Computer States
- **Firmware Computers**
- **Translators and Virtual Architectures :** Translation (or compilation), Software simulation (software interpretation),

### VIRTUAL COMPUTERS AND BINDING TIMES

- **Virtual Computers and Language Implementations**
- **Hierarchies of Virtual Machines**
- **Binding and Binding Time** : Classes of Binding Times , Importance of Binding Times, Binding Times and Language Implementations

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER
## Computer Hardware

- A computer is an **integrated set of algorithms** and **data structures c**apable of storing
and executing programs.
- A computer may be constructed as an actual physical device using **wires, integrated circuits, circuit boards**, and the like, in which case it is termed an actual computer or hardware computer.
- it may also be **constructed via software by programs running** on another computer, in which case it is a **software-simulated computer**.
- A programming language is implemented by construction of a translator, which translates programs in the language into machine language programs that can be directly executed by some computer.

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER
## Computer Hardware

1. **Data.** A computer must provide **various kinds of elementary data items and data structures to be** manipulated.
2. **Primitive operations.** A computer must **provide a set of primitive operations useful for manipulating the data.**
3. **Sequence control.** A computer must provide mechanisms for controlling the sequence in which the primitive operations are to be executed.
4. **Data access.** A computer must provide mechanisms for **controlling the data** supplied to each execution of an operation.
5. **Storage management**. A computer must **provide mechanisms to control the allocation of storage** for programs and data
6. **Operating environment.** A computer must **provide mechanisms for communication** with an external environment containing programs and data to be processed.

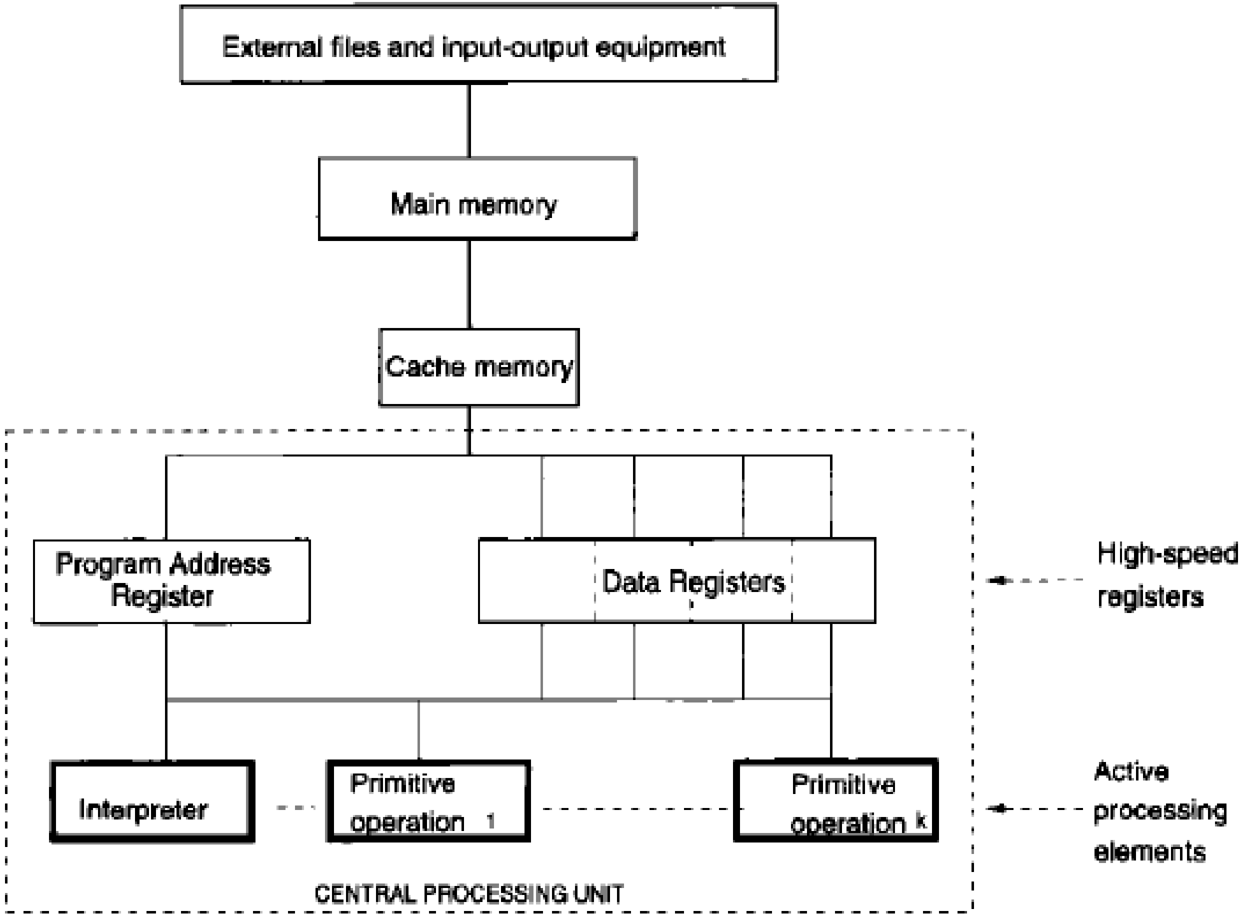



**Figure 2.1.** Organization of a conventional computer.

# Unit 1 Fundamentals of Programming

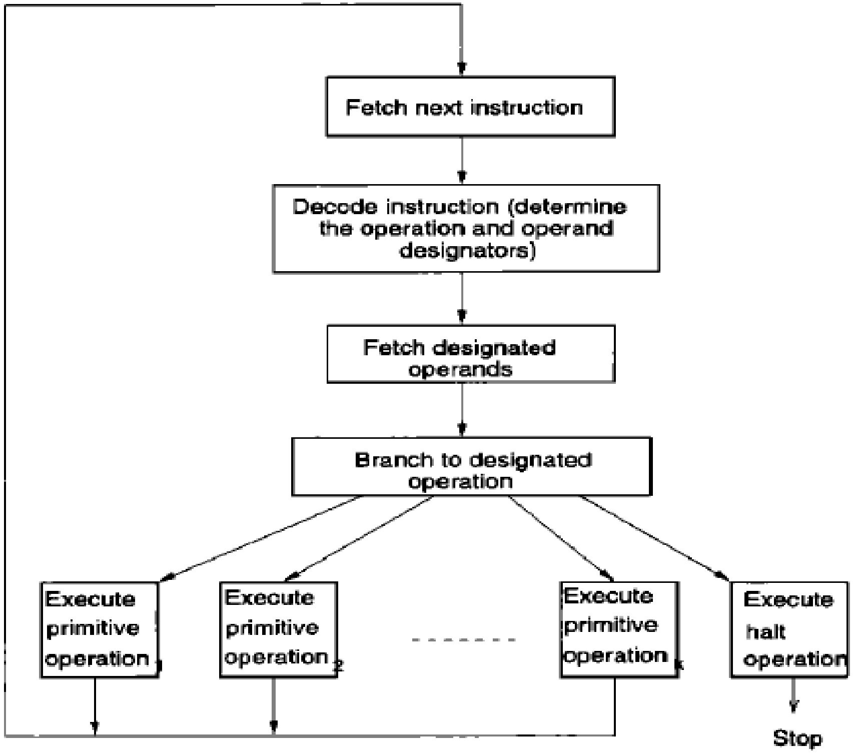



**Figure 2.2.** Program interpretation and execution.

# Unit 1 Fundamentals of Programming

**Impact of Machine Architectures**

THE OPERATION OF A COMPUTER

- **Computer Hardware** : Data, Operations, Sequence Control, Data Access, Storage Management Operating Environment, Alternative computer architectures, Computer States
- **Firmware Computers**
- **Translators and Virtual Architectures :** Translation (or compilation), Software simulation (software interpretation),

VIRTUAL COMPUTERS AND BINDING TIMES

- **Virtual Computers and Language Implementations**
- **Hierarchies of Virtual Machines**
- **Binding and Binding Time** : Classes of Binding Times , Importance of Binding Times, Binding Times and Language Implementations

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

### Firmware of Computers

- Common **alternative to the strict hardware realization** of a computer is the firmware computer simulated by a **microprogram running on a special micro programmable hardware computer.**
- Microprogram simulation of a computer is sometimes **termed emulation.**
- We also refer to the resulting computer as a virtual computer because it is simulated by the microprogram; **without this microprogrammed simulation, the machine would not exist.**

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

## Translators and Virtual Architectures

- **Translator** could be designed to translate programs in the **high-level language into equivalent programs in the machine language** of the actual computer.
- Instead simulate, through p**rograms running on another host computer, a computer whose machine language is the high-level language**.
- We construct w**ith software running on the host computer** (the high-level language computer) that we might **otherwise have constructed in hardware**. This is termed **a software simulation (or software interpretation) of the high-level language computer on the host computer.**

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

### Translators

- An assembler is a translator **whose object language is also some variety of machine language** for an actual computer but whose s**ource language, an assembly language, represents for the most part a symbolic representation** of the object machine code.
- A compiler **is a translator whose source language is a high-level language and whose object language is close to the machine language** of an actual computer,
- A loader or link editor is a **translator whose object language is actual machine code and whose source language is almost identical.**

| Subprogram | Compiled Addresses | Executable Addresses |
|---|---|---|
| *P* | 0-999 | 0-999 |
| *Q* | 0-1,999 | 1,000-2,999 |
| *library* | 0-4,999 | 3,000-7,999 |

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

## Translators

- A preprocessor or a macroprocessor i**s a translator whose source language is an extended form of some high-level language such as C++ or Java and whose object language is the standard form of the same language.**

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

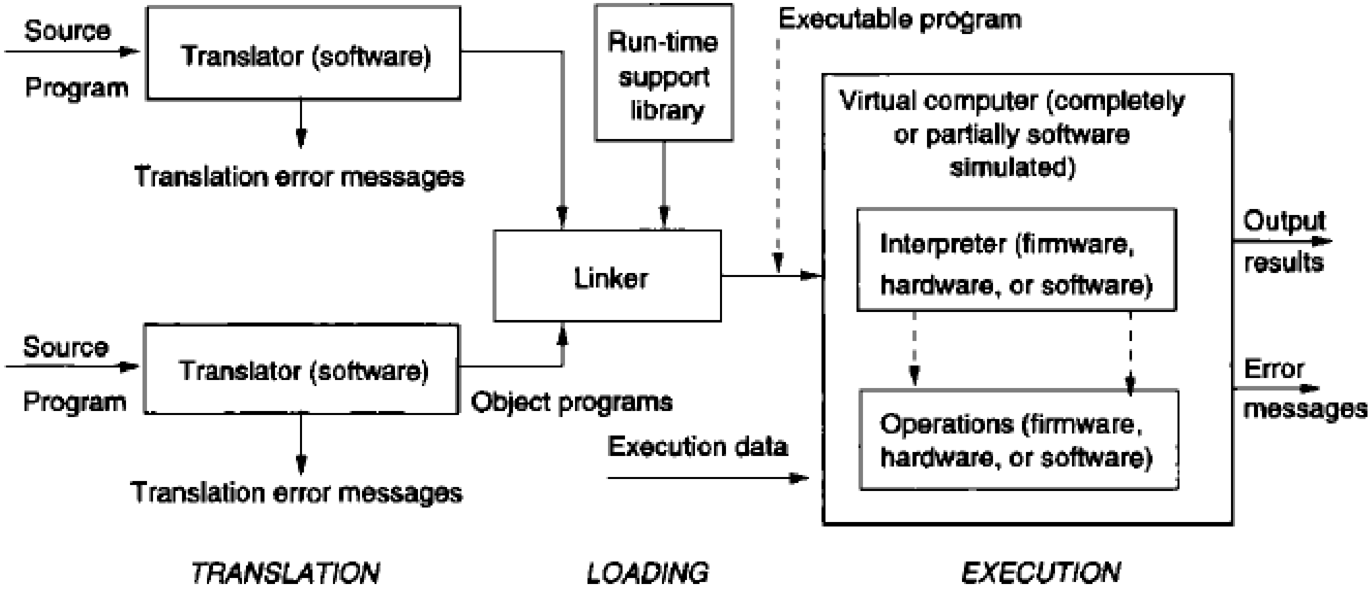



**Figure 2.3.** Structure of a typical language implementation.

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

### Software Simulation

- The simulated computer accepts as **input data a program in the high-level language.**
- The main simulator program performs an **interpretation algorithm similar to that of decoding and executing each statement** of the input program in the appropriate sequence and producing the specified output from the program.
- host computer creates a v**irtual machine simulating**
- **the high-level language.**
- When the host computer is executing the high-level program, it is **not possible to tell whether the program is being executed directly by the hardware or is first converted to the low-level machine language** of the hardware computer.

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER
## Translators and Virtual Architectures

- Translation and simulation provide **different advantages in a programming language implementation**.
- Some aspects of program structure are be**st translated into simpler forms before execution;** other aspects are best **left in their original form and processed only as needed during execution.**
- **The major disadvantage of translation is loss of information about the program.**
- **In Simulation** By leaving statements **in their original form until they need to be executed, no space is needed to store multiple copies of long code sequences**;
- the **basic code need be store**d only once in the simulation routine.
- However, the total cost of **decoding must be paid each time the statement is to be executed.**

## Impact of Machine Architectures - THE OPERATION OF A COMPUTER

## Translators and Virtual Architectures

- The common division of languages

1. **Compiled languages** : translated into the machine language of the actual computer being used before execution begins, eg. C, C++, FORTRAN, Pascal, and Ada are
2. **Interpreted languages** : In such a language implementation, the translator **does not produce machine code** for the computer being used. Instead, the translator produces s**ome intermediate form of the program that is more easily executable than the original program** form yet that is different from machine code.

- **Java and the WWW have changed some of these rules**.

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

1. Through a **hardware realization**, representing the data structures and algorithms directly with physical devices.
2. Through **a firmware realization**, representing the data structures and algorithms by microprogramming a suitable hardware computer
3. Through **a virtual machine**, representing the data structures and algorithm by programs and data structures in some other programming language.
4. Through s**ome combination of these techniques,** representing various parts of the computer directly in hardware, in microprograms, or by software simulation as appropriate.

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

### Virtual Computers and Language Implementations

- The language is **implemented on a different computer, the implementor tends to see a slightly (or very) different virtual computer** in the language definition.
- two different implementations **of the same language may utilize a different set of data structures and operations in the implementation,**
- **three factors** lead to differences among implementations of
  1. Differences in e**ach implementor's conception of the virtual computer**
  2. Differences in the **facilities provided by the host computer** on which the language is to be implemented.
  3. Differences in the c**hoices made by each implementor as to how to simulate** the virtual computer elements

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

### Hierarchies of Virtual Machines

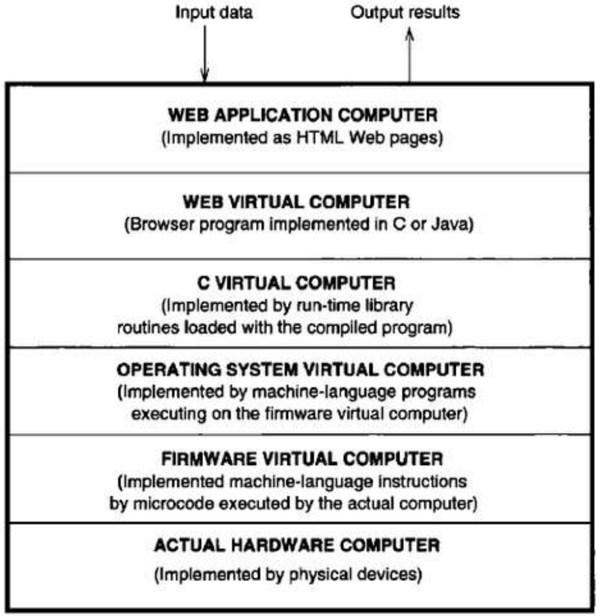



**Figure 2.4.** Layers of virtual computers for a Web application.

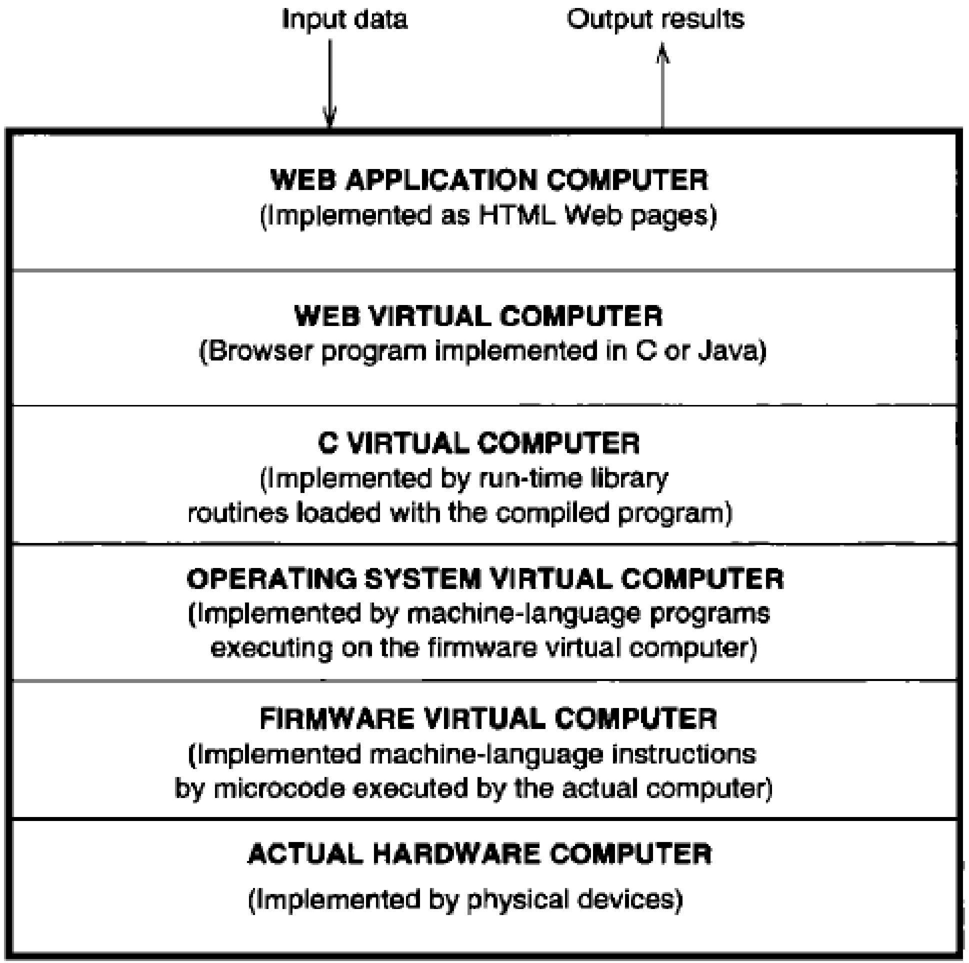



**Figure 2.4.** Layers of virtual computers for a Web application.

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

### Bindings: Names and Attributes

- Names are a fundamental abstraction in languages to denote entities
  - Meanings associated with these entities is captured via attributes associated with the names
- Attributes differ depending on the entity:
  - location (for variables)
  - value (for constants)
  - formal parameter types (functions)
- Binding: Establishing an association between name and an attribute.

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

### Bindings: Names

- Names or Identifiers denote various language entities:
  - Constants
  - Variables
  - Procedures and
  - Functions Types, . . .
- Entities have attributes

| *Entity* | *Example Attributes* |
|---|---|
| Constants | type, value, . . . |
| Variables | type, location, . . . |
| Functions | signature, implementation, . . . |

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

### Bindings: Attributes

- Attributes **are associated with names** (to be more precise, with the entities they denote).
- Attributes describe the **meaning or semantics of names** (and entities).

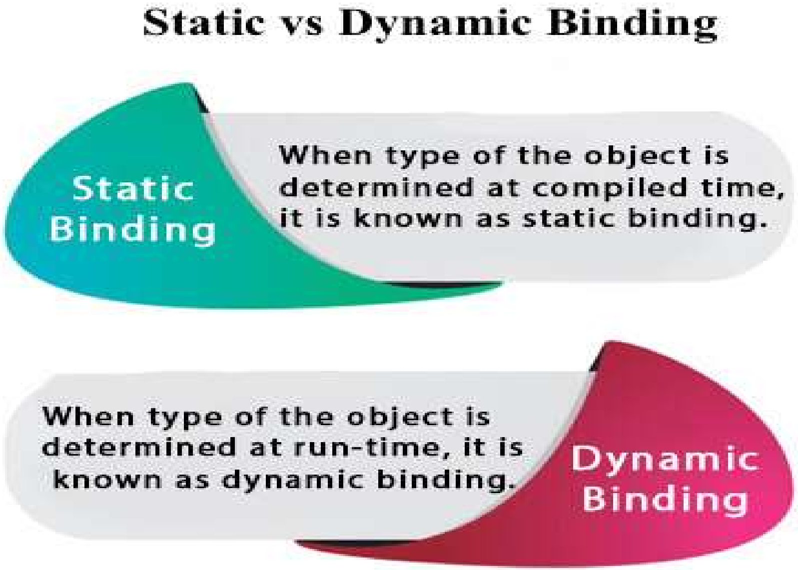
| | |
|---|---|
| `int x;` | There is a variable, named x, of type integer. |
| `int y = 2;` | Variable named x, of type integer, with initial value 2. |
| `Set s = new Set();` | Variable named s, of type Set that refers to an object of class Set |

- An attribute may be
  - static: can be determined at translation (compilation) time, or
  - dynamic: can be determined only at execution time.

# Unit 1 Fundamentals of Programming

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

## Binding and Binding Time -Importance of Binding Times



## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

## Binding Time -Importance of Binding Times

- **Language design time**: built-in features such as keywords
- **Language implementation time:** implementation dependent semantics such as bit-width of an integer
- **Program writing time**: names chosen by programmer
- **Compile time**: bindings of high-level constructs to machine code
- **Link time**: final bindings of names to addresses
- **Load time**: Physical addresses (can change during run time)
- **Run time:** bindings of variables to values, includes many bindings which change during execution

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

## Binding and Binding Time -Classes of Binding Times

1. Execution time (run time)
   a. On entry to a subprogram or block.
   b. At arbitrary points during execution.
2. At arbitrary points during execution.
   a. Bindings chosen by the programmer.
   b. Bindings chosen by the translator.
3. Language implementation time.
4. Language definition time.

## Impact of Machine Architectures - VIRTUAL COMPUTERS AND BINDING TIMES

## Binding and Binding Time -Classes of Binding Times

- consider the simple assignment statement **X=X+10**
- **Points to Think:**
  - Set of types for Variable X.
  - Type of variable X.
  - Set of possible values for variable X.
  - Value of Variable X.
  - Representation of the constant 10.
  - Properties of the operator +.

## Programming Languages

**Machine languages and assembly languages are also called low-level languages**

1. Machine languages.

2. Assembly languages.

3. High-level languages.

## Programming Languages

### Machine languages and assembly languages are also called low-level languages

- A Machine language program consists of a sequence of zeros and ones.
- Each kind of CPU has its own machine language.
- Advantages
  - Fast and efficient
  - Machine oriented
  - No translation required
- DisadvantageS
  - Not portable
  - Not programmer friendly

## Assembly language programs

- Each statement in assembly language corresponds to one statement in machine language.
- Assembly language programs have the same advantages and disadvantages as machine language programs.
- Comp grams:

| 8086 Machine language program for var1 = var1 + var2 ; | 8086 Assembly program for var1 = var1 + var2 ; |
|---|---|
| 1010 0001 0000 0000 0000 0000 | MOV AX , var1 |
| 0000 0011 0000 0110 0000 0000 0000 0010 | ADD AX , var2 |
| 1010 0011 0000 0000 0000 0000 | MOV var1 , AX |

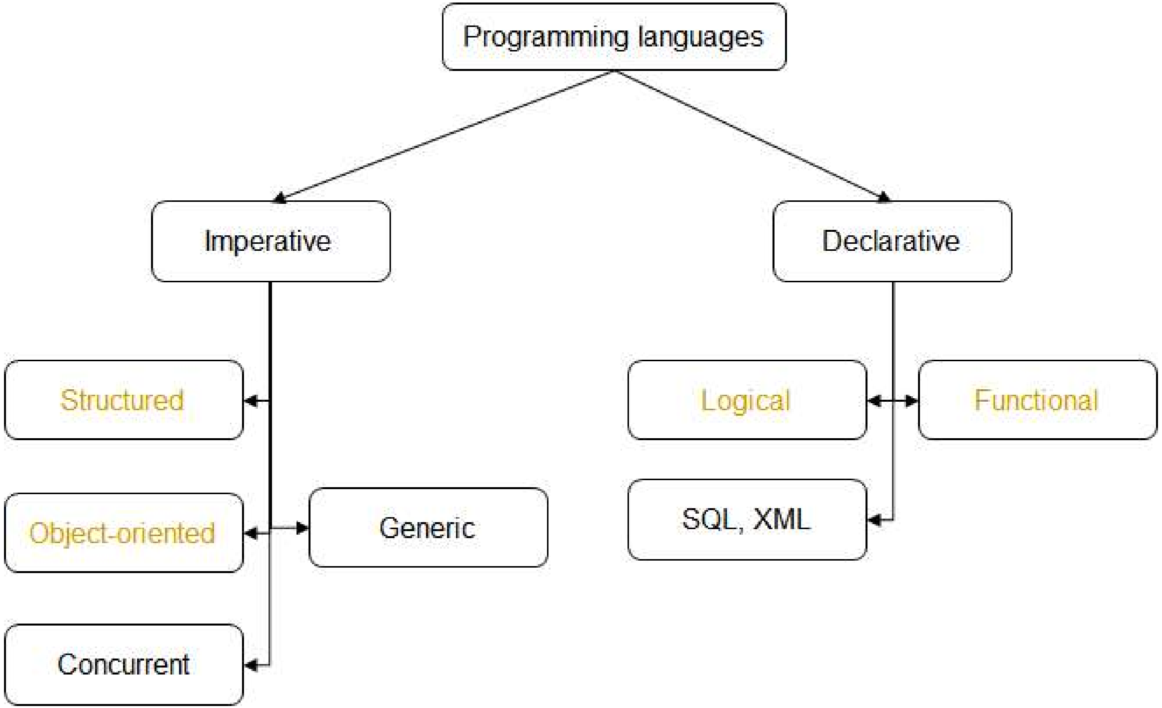
## High-Level Programming Languages

- A high-level language (HLL) has two primary components
  - a set of built-in language primitives and grammatical rules
  - a translator
- A HLL language program consists of English-like statements that are governed by a strict syntax.
- Advantages
  - Portable or machine independent
  - Programmer-friendly
- Disadvantages
  - Not as efficient as low-level languages
  - Need to be translated
  - Examples : C, C++, Java, FORTRAN, Visual Basic, and Delphi.

## Programming Paradigms

- Why are there hundreds of programming languages in use today?
  - Some programming languages are specifically designed for use in certain applications.
  - Different programming languages follow different approaches to solving programming problems
- **A programming paradigm is an approach to solving programming problems.**
- **A programming paradigm may consist of many programming languages.**
- Common programming paradigms:
  - Imperative or Procedural Programming
  - Object-Oriented Programming
  - Functional Programming
  - Logic Programming

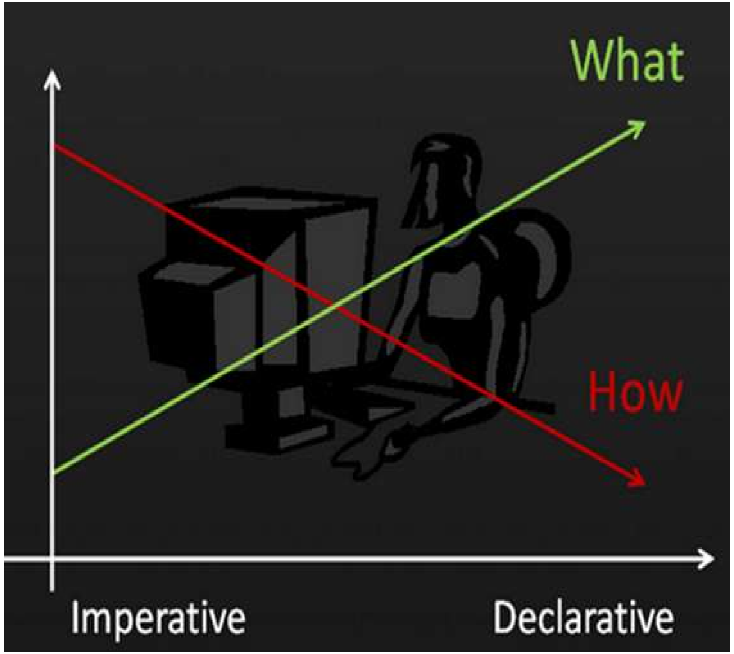
## Programming Paradigms



## Programming Paradigms- IMPERATIVE and DECLARATIVE

- **Imperative programming:** telling the "machine" (computer) **how to do something**, and as a result **what you want to happen will happen.**

- **Declarative programming:** telling the "machine" (computer) **what you would like to happen**, and **let the computer figure out how to do it.**



## Programming Paradigms- IMPERATIVE and DECLARATIVE

**Problem Statement :- Double all the numbers in an array.**

**Imperative style of programming:**

```
var numbers = [1,2,3,4,5]
var doubled = []

for(var i = 0; i < numbers.length; i++) {
  var newNumber = numbers[i] * 2
  doubled.push(newNumber)
}
console.write(doubled) //=> [2,4,6,8,10]
```

**Declarative style of programming:**

```
var numbers = [1,2,3,4,5]

var doubled = numbers.map(function(n)
{
  return n * 2
} )

console.log(doubled) //=> [2,4,6,8,10]
```

## Programming Paradigms- PROCEDURAL/STRUCTURED

- Procedural programming is a computer programming language that organises **our code into small programs" that use and change our datas.**
- **Structured programming** is a programming paradigm recommending h**ierarchical division into blocks of code with one entry point and one or more exit points.**
- In structured programming we use three main structures :
  - sequences (instruction _1; instruction _2;…; instruction _n
  - choices (if, if...else, switch, case)
  - iterations (while, repeat, for).
- **Key words:** variables, types, procedures and abstract datas. Using: network systems, operating systems, etc.
- **Procedural/structured languages:**

Fortran , Cobol , Pascal, c, ,c++ etc,

## Programming Paradigms- PROCEDURAL/STRUCTURED

sum.c — Notatnik

Plik Edycja Format Widok Pomoc

```c
#include<stdio.h>

main()
{
	int x,y;
	printf("Enter two numbers \n");
	scanf("%d%d",&x,&y);
	printf("Sum %d and %d = %d",x,y,x+y);
	return 0;
}
```

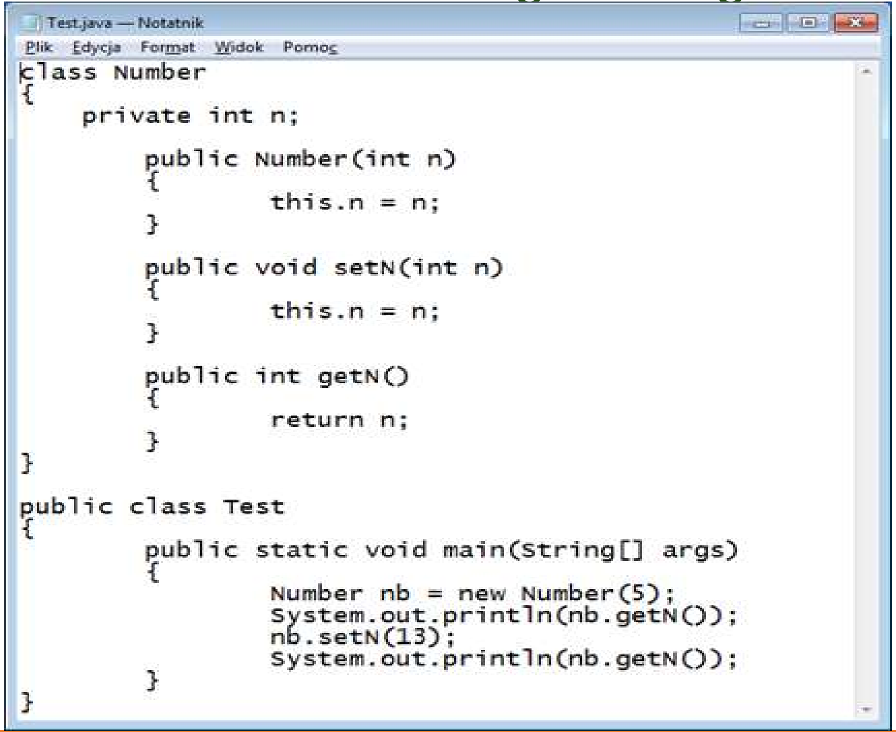
Administrator: C:\Windows\system32\cmd.exe

```
C:\Users\Robert\Desktop>sum.exe
Enter two numbers
4
7
Sum 4 and 7 = 11
C:\Users\Robert\Desktop>
```

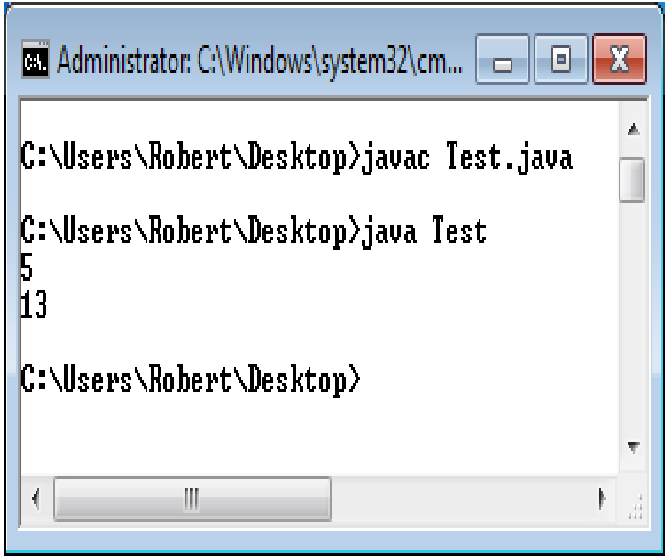
## Programming Paradigms-Object-oriented programming

- Object-oriented programming is a programming paradigm in which **programs are defined using objects - the state of the connecting elements (or fields) and behavior (or method).**
- Object-oriented computer program is expressed **as a set of such objects, which communicate with each other in order to perform tasks.**
- **Key words:** classes and objects, inheritance, encapsulation, polymorphism.
- **Using: www and stand-alone applications.**
- Object-oriented languages
    - Simula,
    - Smalltalk,
    - C++,
    - C#,
    - Java, others.

## Programming Paradigms- Object Oriented




```java
class Number
{
    private int n;

        public Number(int n)
        {
                this.n = n;
        }

        public void setN(int n)
        {
                this.n = n;
        }

        public int getN()
        {
                return n;
        }
}

public class Test
{
        public static void main(String[] args)
        {
                Number nb = new Number(5);
                System.out.println(nb.getN());
                nb.setN(13);
                System.out.println(nb.getN());
        }
}
```

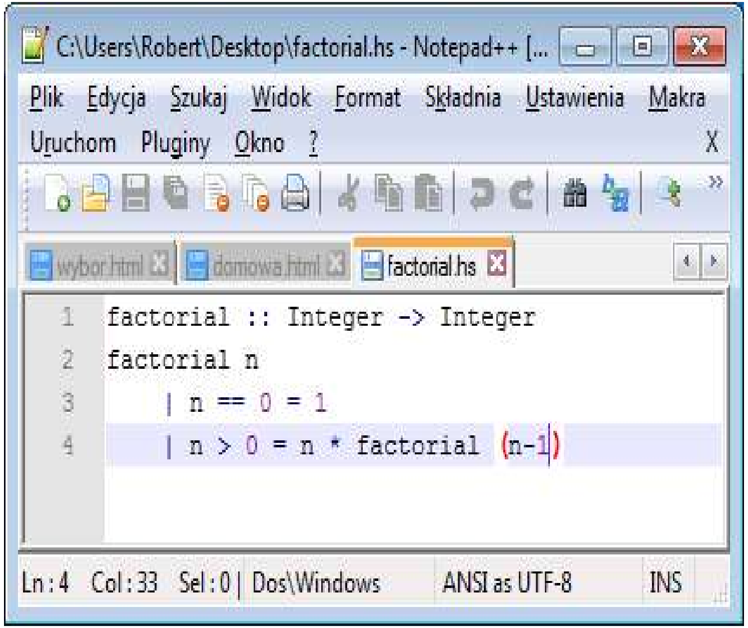


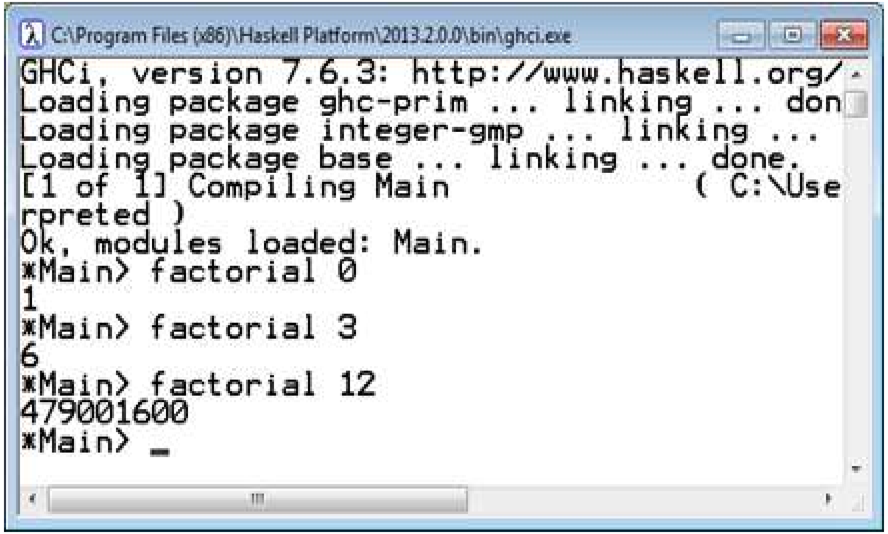

```
C:\Users\Robert\Desktop>javac Test.java

C:\Users\Robert\Desktop>java Test
5
13

C:\Users\Robert\Desktop>
```

## Programming Paradigms-Functional programming

- Functional programming is a programming paradigm in w**hich the functions are the core values and the emphasis is on valuation (often recursive) function, and not to execute commands.**
- Theoretical basis for functional programming was developed in the 19330s of the Twentieth century **by Alonzo Church's lambda calculus, called lambda calculus with types.**
- Key words: functions, lambda calculus, parametric polymorphism.
- Using: theoretical, in telecommunications, in financial calculations.
- Functional languages:
  - Lisp,
  - ML,
  - Haskell,
  - H#,
  - Erlang
  - others.

## Programming Paradigms- Functional Programming



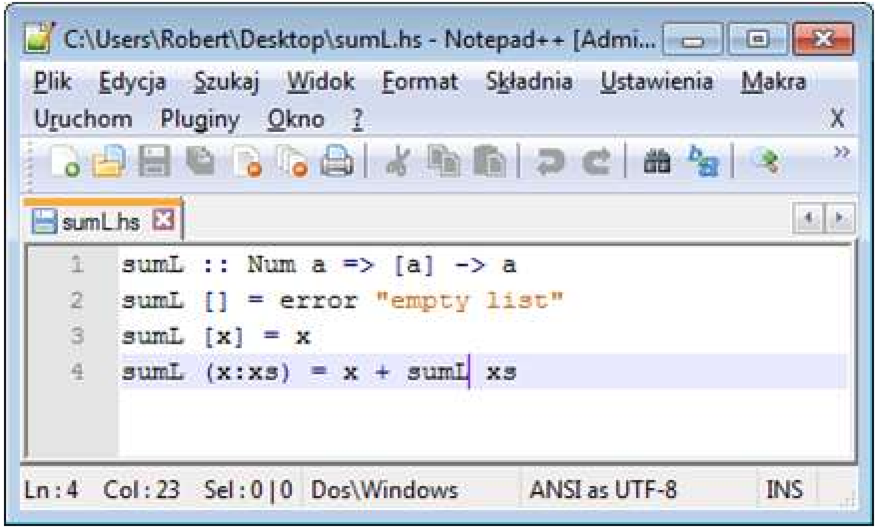
## Programming Paradigms-Logical programming

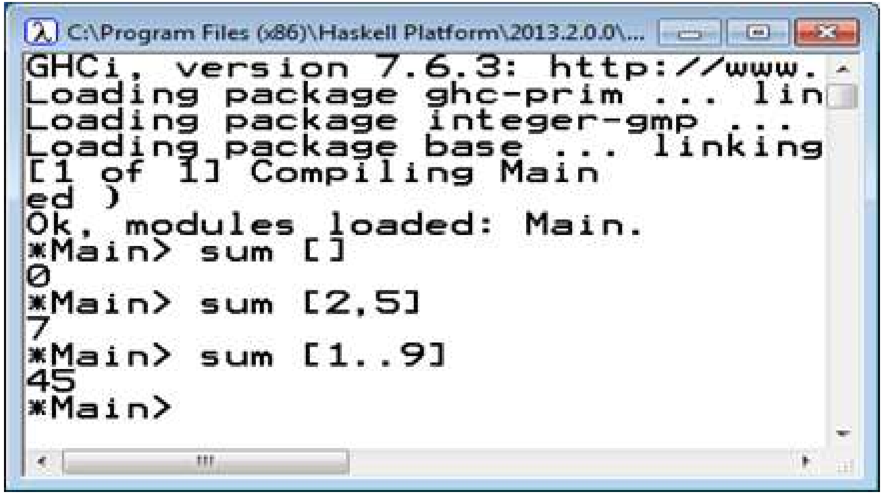
use mnemonics to represent machine instructions

- Each statement in assembly language corresponds to one statement in machine language.
- Assembly language programs have the same advantages and disadvantages as machine language programs.

Compare the following machine language and assembly language programs:

## Programming Paradigms- Functional Programming



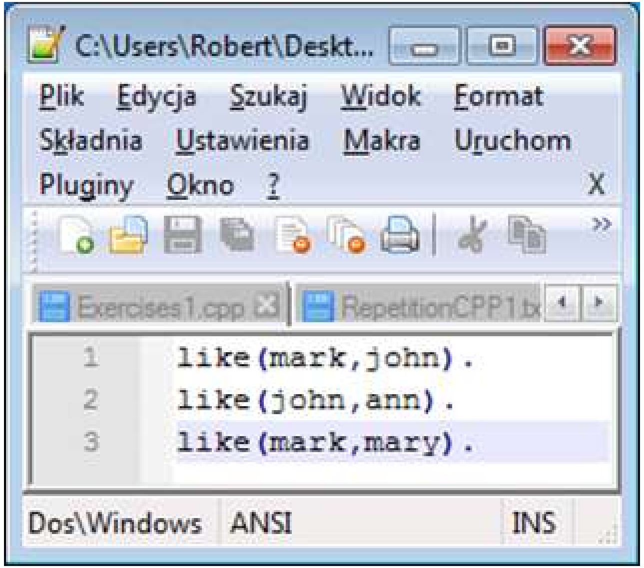



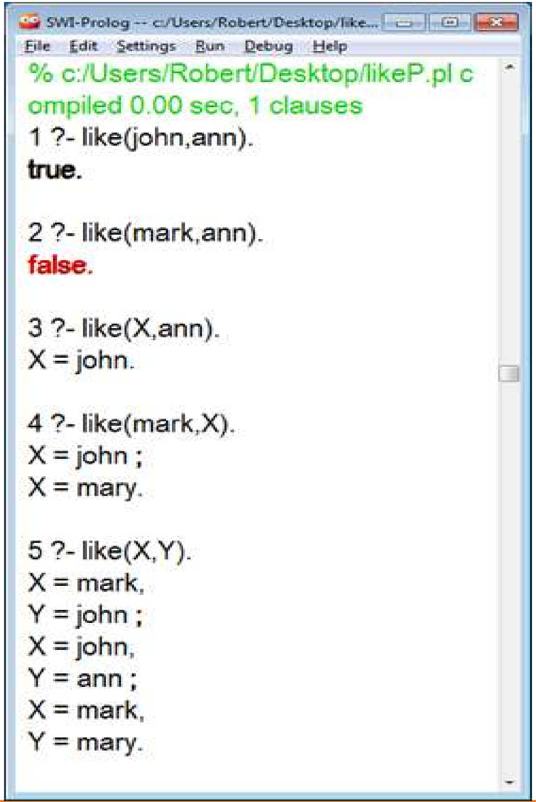
## Programming Paradigms-Logical programming

- The paradigm of logic programming is a programming method in **which the program is given as a set of relations, and the relationship between these dependencies.**
- **Key words**: facts, reports, queries.
- **Using: theoretical, artificial intelligence.**
- **Logical languages:**
  - Gödel,
  - Fril,
  - Prolog, others.

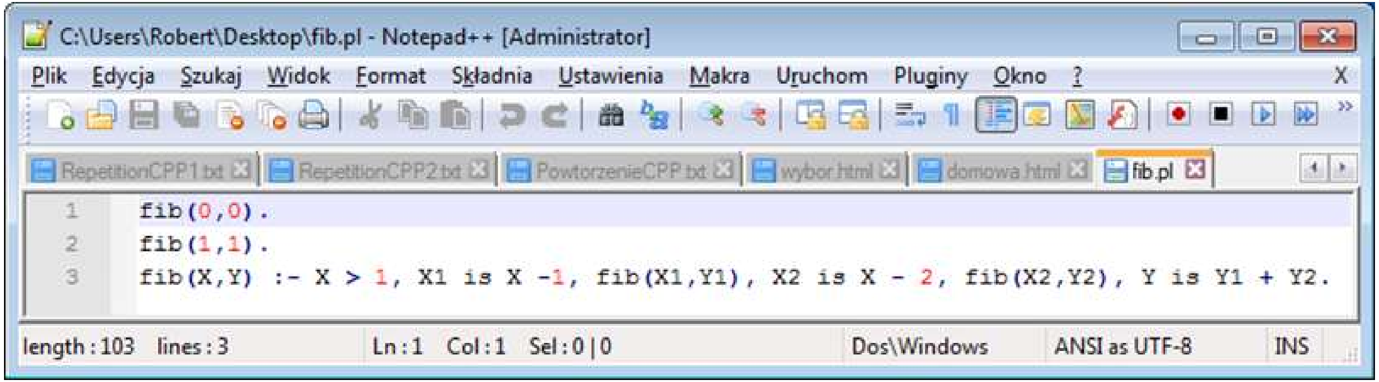
# Unit 1 Fundamentals of Programming

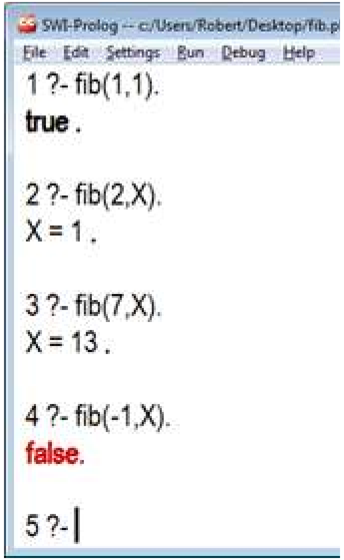
## Programming Paradigms- Logical programming



## Programming Paradigms- Logical programming



| Procedure | Object Oriented | Function Oriented | Logic Oriented |
|---|---|---|---|
| C | C++ | ML | PROLOG |
| COBOL | Java | LISP | CURRY |
| FORTRAN | Ruby | DART | FRIL |
| PASCAL | Python | SCHEME | OZ |
| Java Script | Ada | LOGO | ALMA-0 |
| | Smalltalk | SQL | |
| | PhP | | |