GENERATION OF PROGRAMMING LANGUAGES

by
Dr. Sumit Srivastava
Dept. of Computer Science & Engineering

Unit -I

1

Index

- Programming Language
- Historical Environment
- Features of Programing Languages
- Programing Language Paradigms
- Generation of Programing Languages
- Types of Programing Language
- Implementation of Language

2

Computer Language

- Language

  A system of communication.

- Computer Language

  Means of communication used to communicate between people and the computer.

3

Computer Language

- Difference Between Natural Language And Computer Language

  Natural language has a very large vocabulary whereas computer languages mostly have a very limited vocabulary.

4

Computer Program

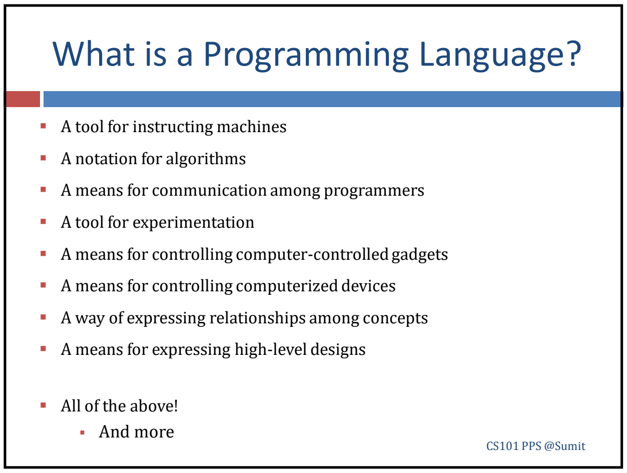
- A program is a set of instructions following the rules of the chosen language.
- Without programs, computers are useless.
- A program is like a recipe.
- It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables.

5

Programming Language

- A vocabulary and set of grammatical rules (syntax) for instructing a computer to perform specific tasks.
- Programming languages can be used to create computer programs.
- The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal.

6

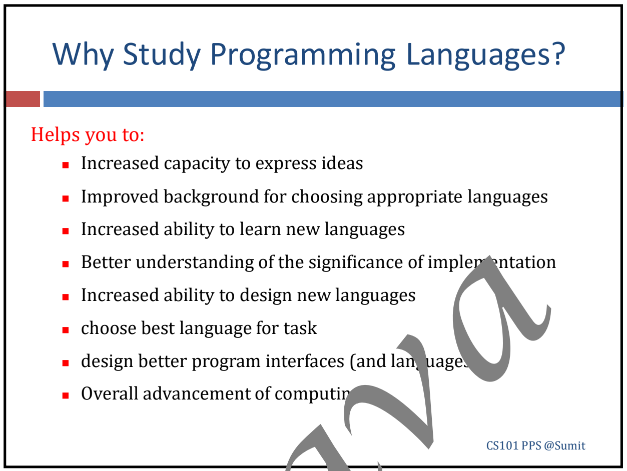


What is a Programming Language?

- A tool for instructing machines
- A notation for algorithms
- A means for communication among programmers
- A tool for experimentation
- A means for controlling computer-controlled gadgets
- A means for controlling computerized devices
- A way of expressing relationships among concepts
- A means for expressing high-level designs
- All of the above!
  - And more

CS101 PPS @Sumit

7



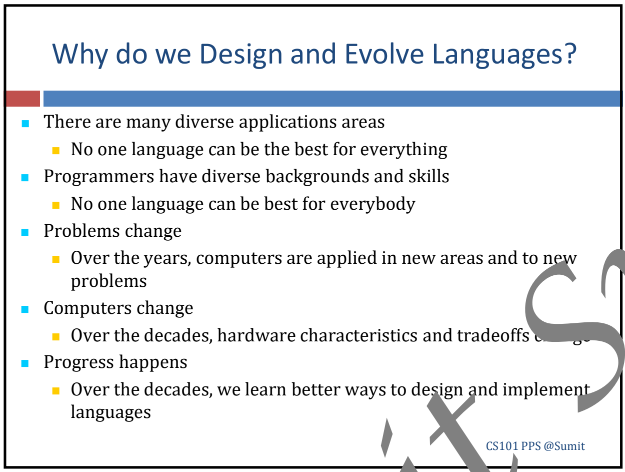
Why Study Programming Languages?

Helps you to:

- Increased capacity to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of the significance of implementation
- Increased ability to design new languages
- choose best language for task
- design better program interfaces (and languages)
- Overall advancement of computing

CS101 PPS @Sumit

8

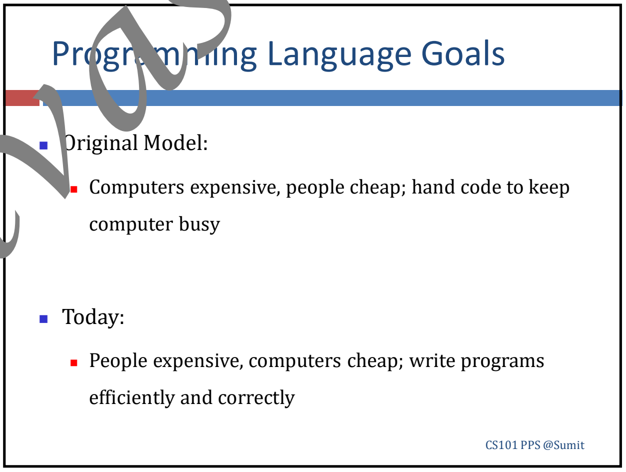


Why do we Design and Evolve Languages?

- There are many diverse applications areas
  - No one language can be the best for everything
- Programmers have diverse backgrounds and skills
  - No one language can be best for everybody
- Problems change
  - Over the years, computers are applied in new areas and to new problems
- Computers change
  - Over the decades, hardware characteristics and tradeoffs change
- Progress happens
  - Over the decades, we learn better ways to design and implement languages

CS101 PPS @Sumit

9

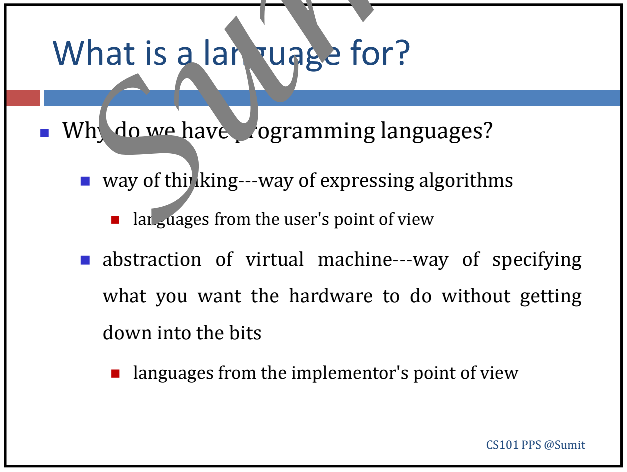


Programming Language Goals

- Original Model:
  - Computers expensive, people cheap; hand code to keep computer busy
- Today:
  - People expensive, computers cheap; write programs efficiently and correctly

CS101 PPS @Sumit

10

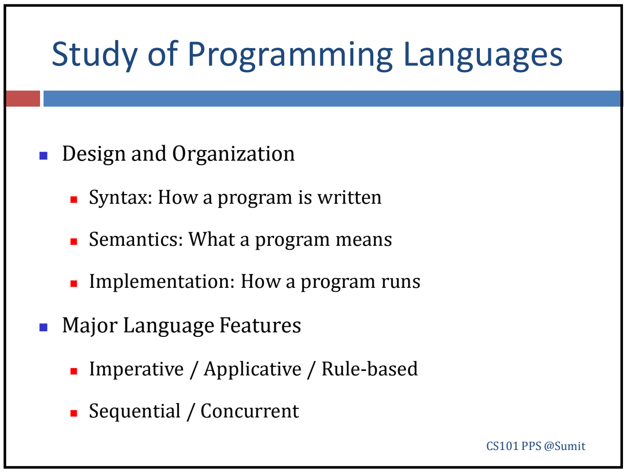


What is a language for?

- Why do we have programming languages?
  - way of thinking---way of expressing algorithms
    - languages from the user's point of view
  - abstraction of virtual machine---way of specifying what you want the hardware to do without getting down into the bits
    - languages from the implementor's point of view

CS101 PPS @Sumit

11



Study of Programming Languages

- Design and Organization
  - Syntax: How a program is written
  - Semantics: What a program means
  - Implementation: How a program runs
- Major Language Features
  - Imperative / Applicative / Rule-based
  - Sequential / Concurrent

CS101 PPS @Sumit

12

Historical Environment

13

Historical Environment

- **Mainframe Era**
  - Batch environments (through early 60's and 70's)
    - Programs submitted to operator as a pile of punch cards; programs were typically run over night and output put in programmer's bin

14

Historical Environment

- **Mainframe Era**
  - Interactive environments
    - Multiple teletypes and CRT's hooked up to single mainframe
    - Time-sharing OS (Multics) gave users time slices
    - Lead to compilers with read-eval-print loops

15

Historical Environment

- **Personal Computing Era**
  - Small, cheap, powerful
  - Single user, single-threaded OS (at first any way)
  - Windows interfaces replaced line input
  - Wide availability lead to inter-computer communications and distributed systems

16

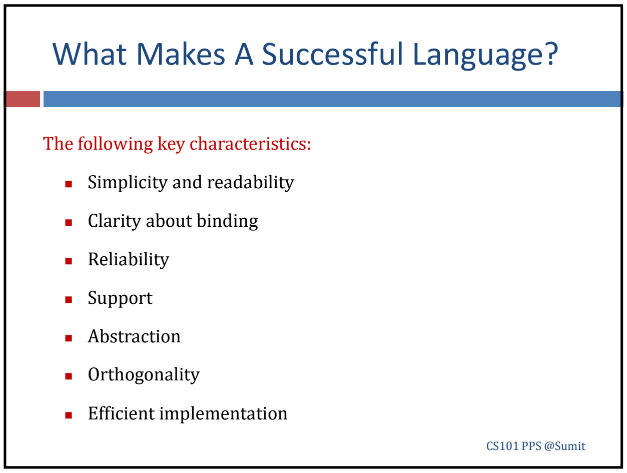
Historical Environment

- **Networking Era**
  - Local area networks for printing, file sharing, application sharing
  - Global network
    - First called ARPANET, now called Internet
    - Composed of a collection of protocols: FTP, Email (SMTP), HTTP (HMTL), URL

17

Features of a Good Language

18



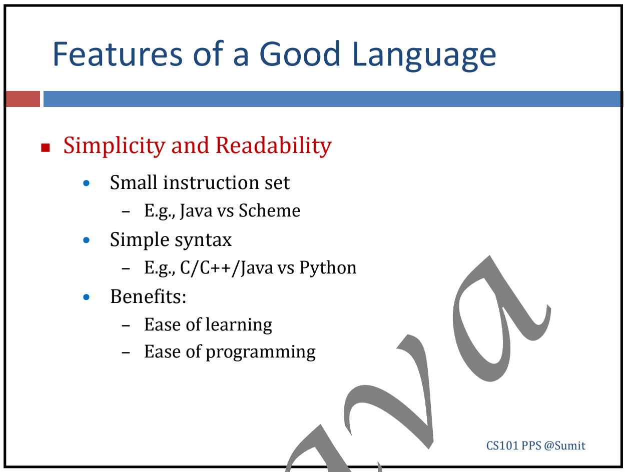
What Makes A Successful Language?

The following key characteristics:

- Simplicity and readability
- Clarity about binding
- Reliability
- Support
- Abstraction
- Orthogonality
- Efficient implementation

CS101 PPS @Sumit

19

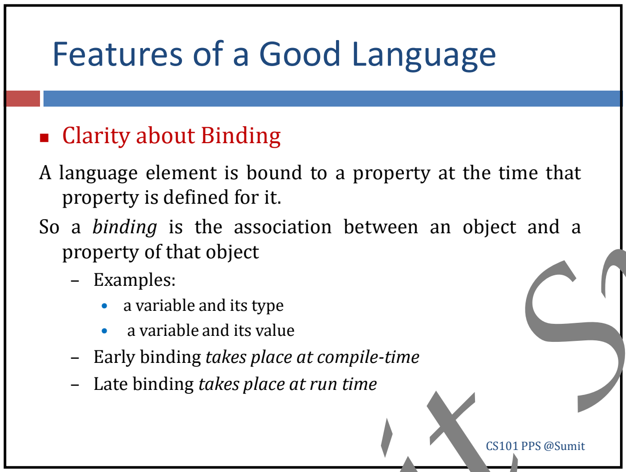


Features of a Good Language

- Simplicity and Readability
  - Small instruction set
    - E.g., Java vs Scheme
  - Simple syntax
    - E.g., C/C++/Java vs Python
  - Benefits:
    - Ease of learning
    - Ease of programming

CS101 PPS @Sumit

20



Features of a Good Language

- Clarity about Binding

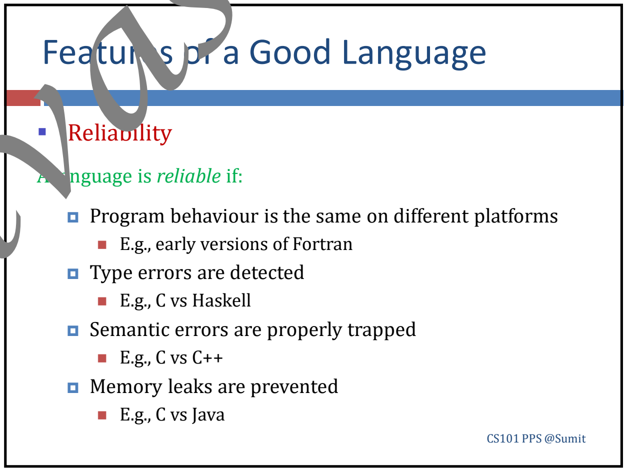
A language element is bound to a property at the time that property is defined for it.

So a *binding* is the association between an object and a property of that object

- Examples:
  - a variable and its type
  - a variable and its value
- Early binding *takes place at compile-time*
- Late binding *takes place at run time*

CS101 PPS @Sumit

21



Features of a Good Language

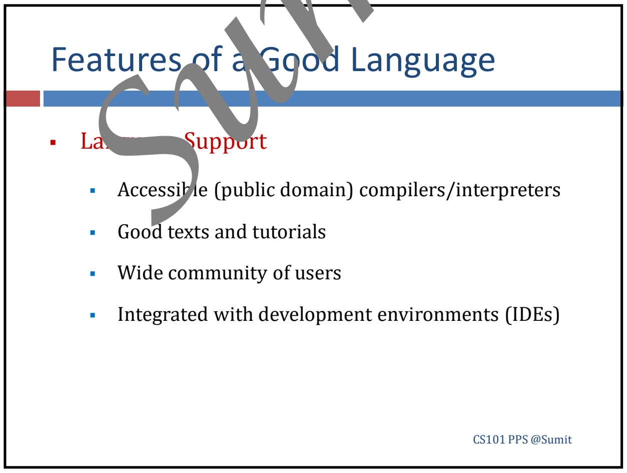
- Reliability

A language is *reliable* if:

- Program behaviour is the same on different platforms
  - E.g., early versions of Fortran
- Type errors are detected
  - E.g., C vs Haskell
- Semantic errors are properly trapped
  - E.g., C vs C++
- Memory leaks are prevented
  - E.g., C vs Java

CS101 PPS @Sumit

22

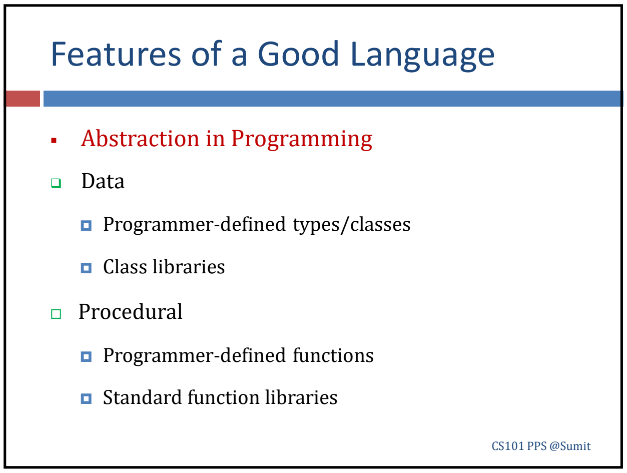


Features of a Good Language

- Language Support
  - Accessible (public domain) compilers/interpreters
  - Good texts and tutorials
  - Wide community of users
  - Integrated with development environments (IDEs)

CS101 PPS @Sumit

23



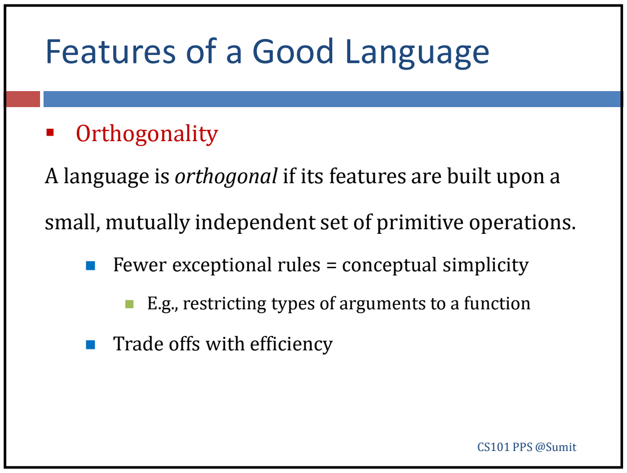
Features of a Good Language

- Abstraction in Programming

- Data
  - Programmer-defined types/classes
  - Class libraries
- Procedural
  - Programmer-defined functions
  - Standard function libraries

CS101 PPS @Sumit

24



Features of a Good Language

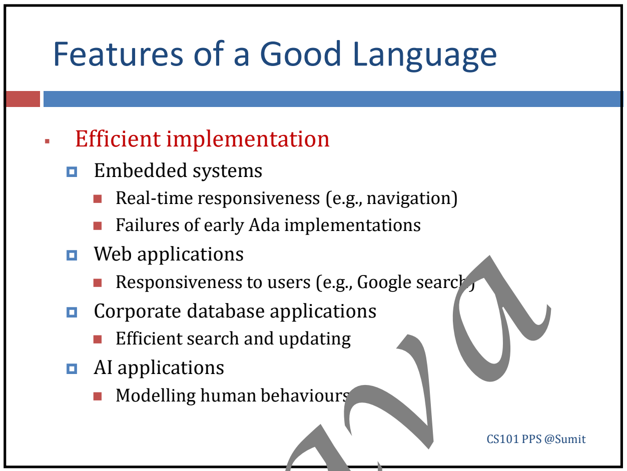
- Orthogonality

A language is *orthogonal* if its features are built upon a small, mutually independent set of primitive operations.

- Fewer exceptional rules = conceptual simplicity
    - E.g., restricting types of arguments to a function
- Trade offs with efficiency

CS101 PPS @Sumit

25

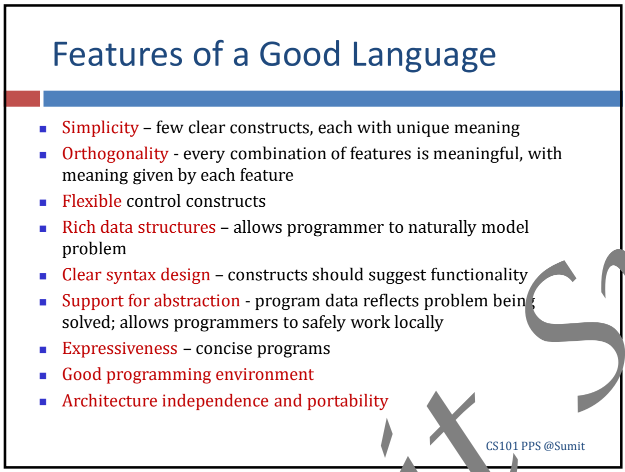


Features of a Good Language

- Efficient implementation
    - Embedded systems
        - Real-time responsiveness (e.g., navigation)
        - Failures of early Ada implementations
    - Web applications
        - Responsiveness to users (e.g., Google search)
    - Corporate database applications
        - Efficient search and updating
    - AI applications
        - Modelling human behaviours

CS101 PPS @Sumit

26

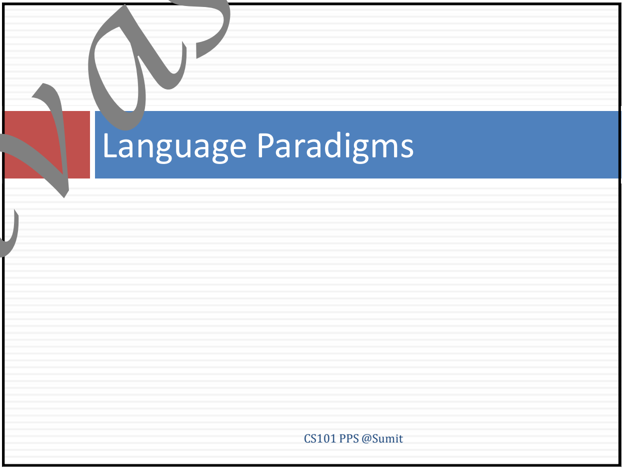


Features of a Good Language

- Simplicity – few clear constructs, each with unique meaning
- Orthogonality - every combination of features is meaningful, with meaning given by each feature
- Flexible control constructs
- Rich data structures – allows programmer to naturally model problem
- Clear syntax design – constructs should suggest functionality
- Support for abstraction - program data reflects problem being solved; allows programmers to safely work locally
- Expressiveness – concise programs
- Good programming environment
- Architecture independence and portability

CS101 PPS @Sumit

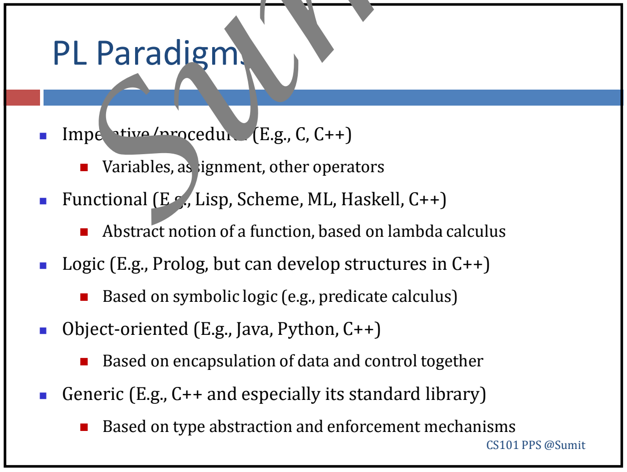
27



Language Paradigms

CS101 PPS @Sumit

28

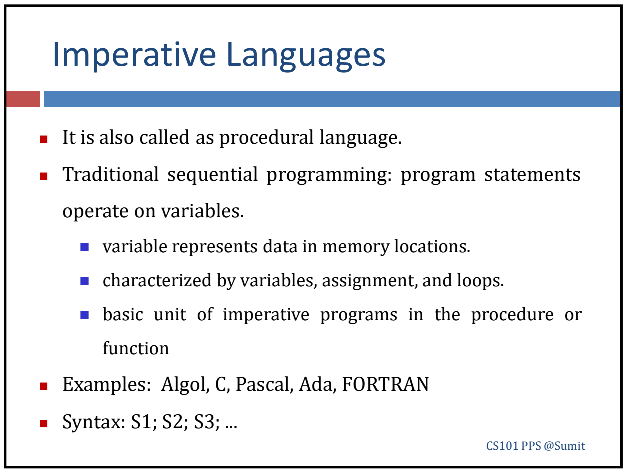


PL Paradigms

- Imperative (procedural) (E.g., C, C++)
    - Variables, assignment, other operators
- Functional (E.g., Lisp, Scheme, ML, Haskell, C++)
    - Abstract notion of a function, based on lambda calculus
- Logic (E.g., Prolog, but can develop structures in C++)
    - Based on symbolic logic (e.g., predicate calculus)
- Object-oriented (E.g., Java, Python, C++)
    - Based on encapsulation of data and control together
- Generic (E.g., C++ and especially its standard library)
    - Based on type abstraction and enforcement mechanisms

CS101 PPS @Sumit

29



Imperative Languages

- It is also called as procedural language.
- Traditional sequential programming: program statements operate on variables.
    - variable represents data in memory locations.
    - characterized by variables, assignment, and loops.
    - basic unit of imperative programs in the procedure or function
- Examples: Algol, C, Pascal, Ada, FORTRAN
- Syntax: S1; S2; S3; ...

CS101 PPS @Sumit

30

Object-oriented Languages

- Classes are complex data types grouped with operations (methods) for creating, examining, and modifying elements (objects); subclasses include (inherit) the objects and methods from superclasses

31

Object-oriented Languages

- Classes are complex data types grouped with operations (methods) for creating, examining, and modifying elements (objects); subclasses include (inherit) the objects and methods from superclasses.
- Computation is based on objects sending messages (methods applied to arguments) to other objects
- Syntax: Varies
- Example languages: Java, C++, Smalltalk

32

Applicative (Functional) languages

- Programs as functions that take arguments and return values; arguments and returned values may be functions
- Programming consists of building the function that computes the answer; function application and composition main method of computation
- Syntax: P1(P2(P3 X))
- Example languages: ML, LISP, Scheme, Haskell, Miranda

33

Logic Programming

- Rule-based languages
  - Programs as sets of basic rules for decomposing problem
  - Computation by deduction: search, unification and backtracking main components
  - Syntax: Answer :- specification rule
  - Example languages: (Prolog, Datalog, BNF Parsing)

program is *declarative*, it specifies what must be true but not how to compute it.

34

Logic Programming

- program is *declarative*, it specifies what must be true but not how to compute it.
  - ◘ logic inference the basic control
  - ◘ no sequential operation
  - ◘ non-deterministic: may have many solutions or none

35

More Language Paradigms (1)

❑ Declarative: state what needs computing, not how to compute it (algorithm).

- Many 4GL, like SQL and Mathematica share this property.
- Prolog is also declarative

36

More Language Paradigms (1)

Concurrent or Parallel: Programming to utilize multiple CPU or multiple threads of execution.

- Requires attention to task management, synchronization, and data conflict
- sequence of execution may not be predictable.
- parallel features are often added to existing programming languages.

❑ Examples: threads in Java, C#, and other languages. MPI (Message Passing Interface) library for cluster and grid computing.

37

Programming Language Implementation

- Develop layers of machines, each more primitive than the previous
- Translate between successive layers
- End at basic layer
- Ultimately hardware machine at bottom

38

The Onion Model of Computers

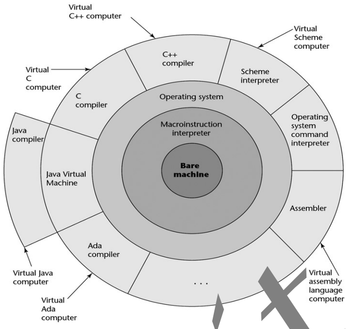


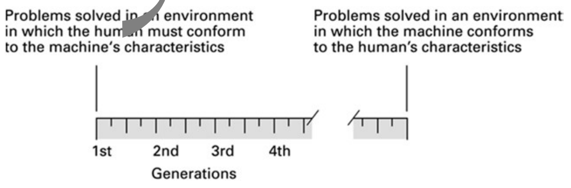
Figure 1.2
Layered interface of virtual computers, provided by a typical computer system

39

Generations of PL

40

Generations of PL



Problems solved in an environment in which the human must conform to the machine's characteristics
Problems solved in an environment in which the machine conforms to the human's characteristics
1st
2nd
3rd
4th
Generations

41

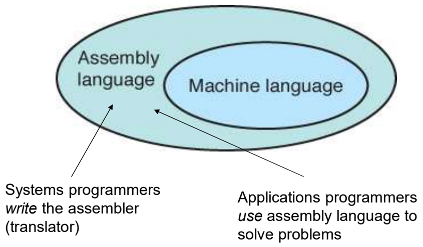
First Generation PL

- The first-generation languages are also called machine languages/ 1G language.
- This language is machine-dependent.
- The machine language statements are written in binary code (0/1 form) because the computer can understand only binary language.
- Advantages :
  1. Fast & efficient as statements are directly written in binary language.
  2. No translator is required.
- Disadvantages :
  1. Difficult to learn binary codes.
  2. Difficult to understand – both programs & where the error occurred.

42

Assembly/Machine

Programmers divide into two groups: application programmers and systems programmers


Assembly language
Machine language
Systems programmers *write* the assembler (translator)
Applications programmers *use* assembly language to solve problems

43

Second Generation PL

- The second-generation languages are also called assembler languages/ 2G languages.
- Assembly language contains human-readable notations that can be further converted to machine language using an assembler.
- Advantages :
  1. It is easier to understand if compared to machine language.
  2. Modifications are easy.
  3. Correction & location of errors are easy.
- Disadvantages :
  1. Assembler is required.
  2. This language is architecture /machine-dependent, with a different instruction set for different machines.

44

Third Generation PL (HLL)

- The third generation is also called procedural language /3 GL.
- It consists of the use of a series of English-like words that humans can understand easily, to write instructions.
- It's also called High-Level Programming Language.
- For execution, a program in this language needs to be translated into machine language using a Compiler/ Interpreter.
- Examples of this type of language are C, C++, PASCAL, FORTRAN, COBOL, etc.

45

Third Generation PL

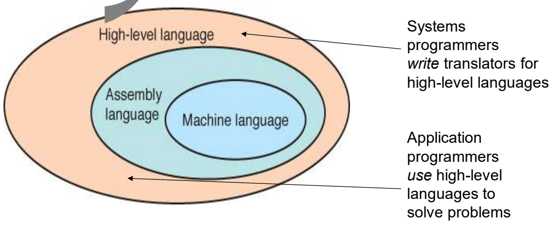
- Advantages
  1. It is easy to develop, learn and understand the program.
  2. As the program written in these languages are less prone to errors they are easy to maintain.
  3. The program written in these languages can be developed in very less time as compared to the first and second generation language.
- Disadvantages
  1. Compiler/ interpreter is needed.
  2. Different compilers are needed for different machines.

46

Third Generation PL

High-level Languages

English-like statements made programming easier:


High-level language
Assembly language
Machine language
Systems programmers *write* translators for high-level languages
Application programmers *use* high-level languages to solve problems

47

Fourth Generation PL (Very HLL)

- The fourth-generation language is also called a non – procedural language/ 4GL.
- The languages of this generation were considered as very high-level programming languages required a lot of time and effort that affected the productivity of a programmer.
- It were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.
- It enables users to access the database. Examples: SQL, Foxpro, Focus, CSS, Coldfusion etc.
- These languages are also human-friendly to understand.

48

Fourth Generation PL (Very HLL)

Advantages:

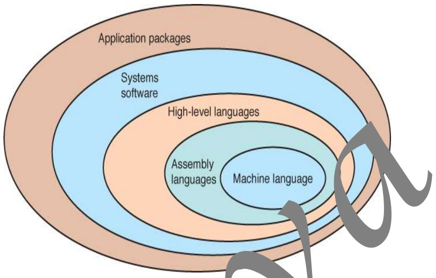
1. These programming languages allow the efficient use of data by implementing the various database.
2. They require less time, cost and effort to develop different types of software applications.
3. The program developed in these languages are highly portable as compared to the programs developed in the languages of other generation.

- Disadvantages :

1. Memory consumption is high.
2. Has poor control over Hardware.
3. Less flexible.

49

Forth Generation PL



50

Fifth Generation PL (AI Language)

- The fifth-generation languages are also called 5GL.
- It is based on the concept of artificial intelligence.
- It uses the concept that rather than solving a problem algorithmically.
- An application can be built to solve it based on some constraints, i.e., we make computers learn to solve any problem.
-
- Parallel Processing & superconductors are used for this type of language to make real artificial intelligence.
- Examples: PROLOG, LISP, Mercury, OPS5 etc.

51

Fifth Generation PL (AI Language)

- Advantages :

1. Machines can make decisions.
2. Programmer effort reduces to solve a problem.
3. Easier than 3GL or 4GL to learn and use.

- Disadvantages :

1. Complex and long code.
2. More resources are required & they are expensive too.

52

Sixth Generation PL

- Sixth generation programming language (6GPL) is a very high-level programming language with extreme abstraction from the hardware.
- It usually consists of a set of human-readable instructions that must be analyzed by a command interpreter.
- Such languages may be domain-specific or general-purpose and often apply natural language processing in order to function.
- It is based on No code and Visual Development.

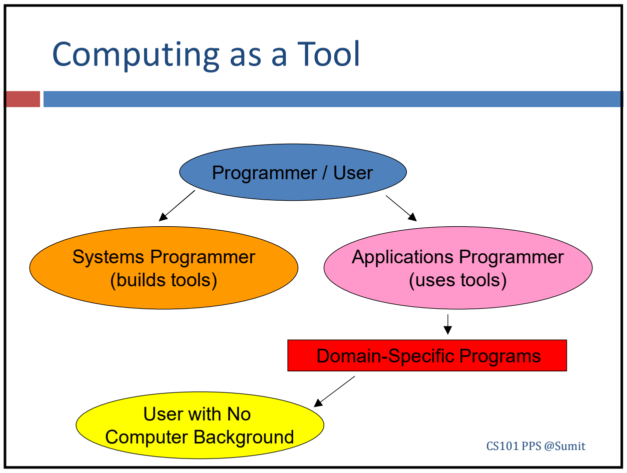
53

Sixth Generation PL

- The following program written in X++ asks a person to enter their username and password.
- *WRITE username and REQUEST user to FILL IN username.*
- *WRITE password and REQUEST user to FILL IN password.*
- *IF username and password are FILLED IN, LOG IN to system.*
- *User SHALL FILL IN username as text; THEN, press ENTER to GO TO password; then, FILL IN password.*

- WRITE tells the system to write text on the screen. WRITE username outputs: username REQUEST user to FILL IN username tells system to ask a person to type their username.

54



Computing as a Tool

Programmer / User

Systems Programmer (builds tools)

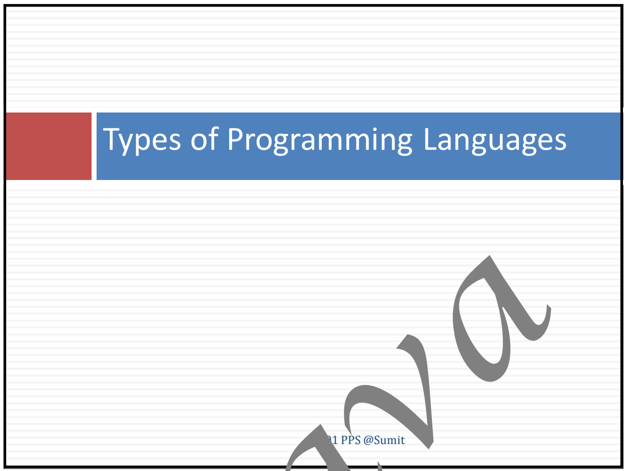
Applications Programmer (uses tools)

Domain-Specific Programs

User with No Computer Background

CS101 PPS @Sumit

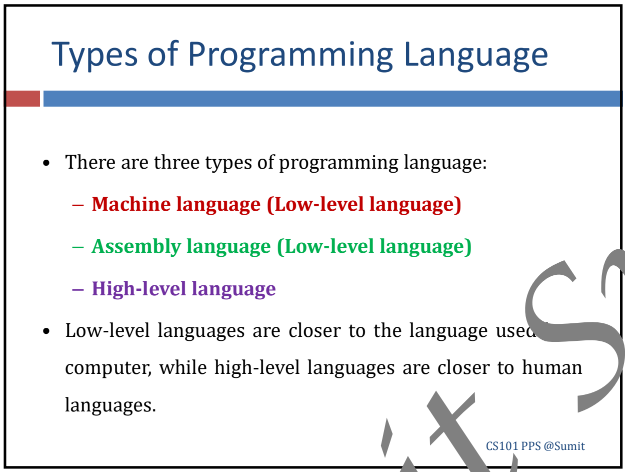
55



Types of Programming Languages

CS101 PPS @Sumit

56

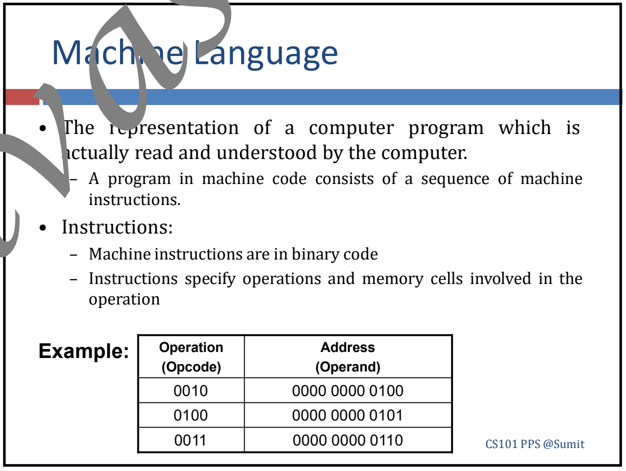


Types of Programming Language

- There are three types of programming language:
  - **Machine language (Low-level language)**
  - **Assembly language (Low-level language)**
  - **High-level language**
- Low-level languages are closer to the language used by a computer, while high-level languages are closer to human languages.

CS101 PPS @Sumit

57



Machine Language

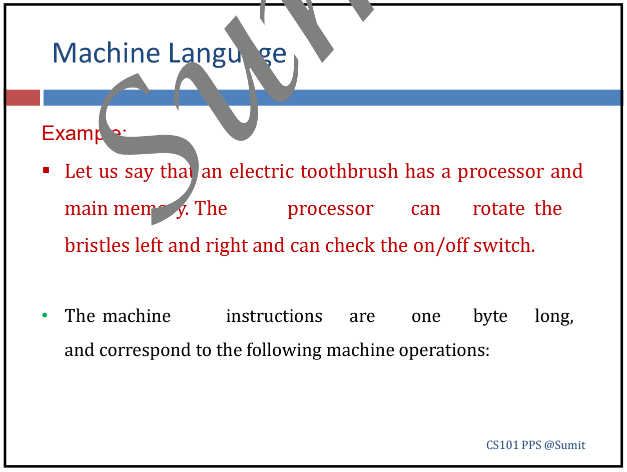
- The representation of a computer program which is actually read and understood by the computer.
  - A program in machine code consists of a sequence of machine instructions.
- Instructions:
  - Machine instructions are in binary code
  - Instructions specify operations and memory cells involved in the operation

Example:

Operation (Opcode)	Address (Operand)
0010	0000 0000 0100
0100	0000 0000 0101
0011	0000 0000 0110

CS101 PPS @Sumit

58



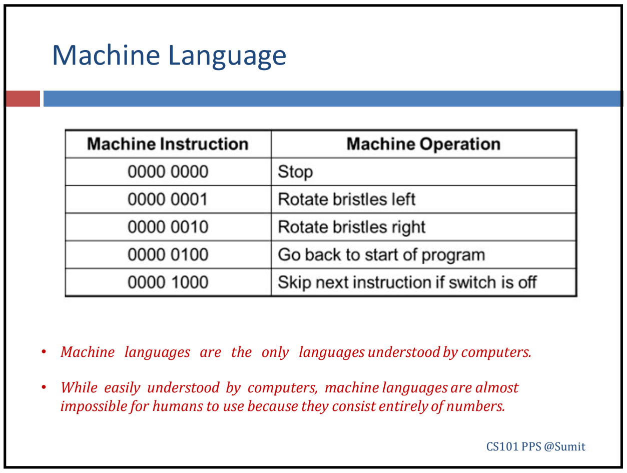
Machine Language

Example:

- Let us say that an electric toothbrush has a processor and main memory. The processor can rotate the bristles left and right and can check the on/off switch.
- The machine instructions are one byte long, and correspond to the following machine operations:

CS101 PPS @Sumit

59



Machine Language

Machine Instruction	Machine Operation
0000 0000	Stop
0000 0001	Rotate bristles left
0000 0010	Rotate bristles right
0000 0100	Go back to start of program
0000 1000	Skip next instruction if switch is off

- *Machine languages are the only languages understood by computers.*
- *While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.*

CS101 PPS @Sumit

60

Machine Language

Advantages of Machine Language

- Programs written in machine language are very fast to execute as instructions written in Machine language are directly understood by CPU and no translation program is required.

Limitations of Machine Language

- Machine dependent.
- Difficult to program
- Error prone.

61

Assembly Language

- A program written in assembly language consists of a series of instructions mnemonics that correspond to a stream of executable instructions, when translated by an assembler, that can be loaded into memory and executed.
- Assembly languages use keywords and symbols, much like English, to form a programming language but at the same time introduce a new problem.
- The problem is that the computer doesn't understand the assembly code, so we need a way to convert it to machine code, which the computer does understand.
- Assembly language programs are translated into machine language by a program called an assembler.

62

Assembly Language

- A symbolic representation of the machine language of a specific processor.
- Is converted to machine code by an assembler.
- Usually, each line of assembly code produces one machine instruction (One-to-one correspondence).
- Programming in assembly language is slow and error-prone but is more efficient in terms of hardware performance.
- Mnemonic representation of the instructions and data

63

Assembly Language

- **Example:**

Machine language :

10110000 01100001

Assembly language :

mov a1, #061h

Meaning:

Move the hexadecimal value 61 (97 decimal) into the processor register named "a1".

64

Assembly Language

- **Example: translate the following statement to assembly language and machine code.**
- **x=y*(y+z);**
- Assume x,y and z are stored in memory locations 0,1 and 2 and there are general purpose registers called A,B,C...etc

Assembly Language	**Machine Code**	
MOV A,[1]	**3e 00 01 ;**	**A=y**
MOV B,[2]	**3f 00 02 ;**	**B=z**
ADD A,B	**8c ;**	**A=A+B;**
MULT A,B	**9f ;**	**A=A*B**
MOV [0],A	**4e 00 00 ;**	**x=A**

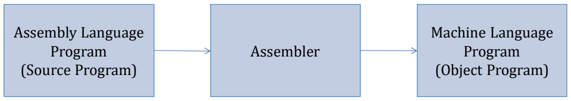
65

Assembler

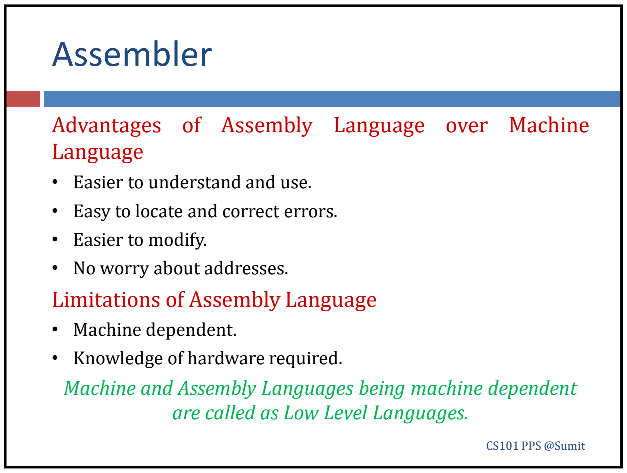
□ The translator program that translates an assembly code into machine code is called an Assembler.

□ One to one translation : One AL instruction is mapped to one ML instruction.

□ AL instructions are CPU specific.


Assembly Language Program (Source Program)
Assembler
Machine Language Program (Object Program)

66



Assembler

Advantages of Assembly Language over Machine Language

- Easier to understand and use.
- Easy to locate and correct errors.
- Easier to modify.
- No worry about addresses.

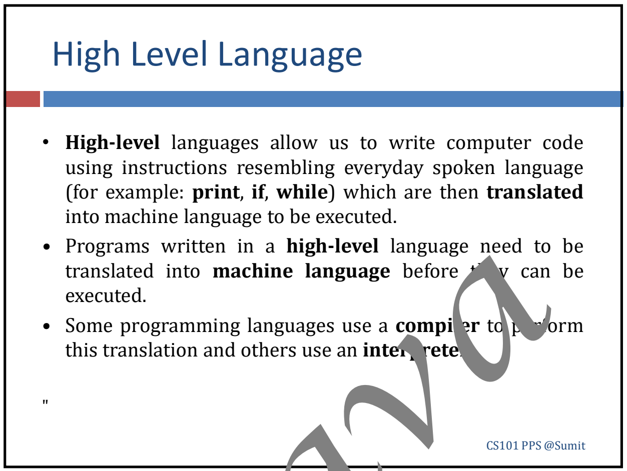
Limitations of Assembly Language

- Machine dependent.
- Knowledge of hardware required.

Machine and Assembly Languages being machine dependent are called as Low Level Languages.

CS101 PPS @Sumit

67



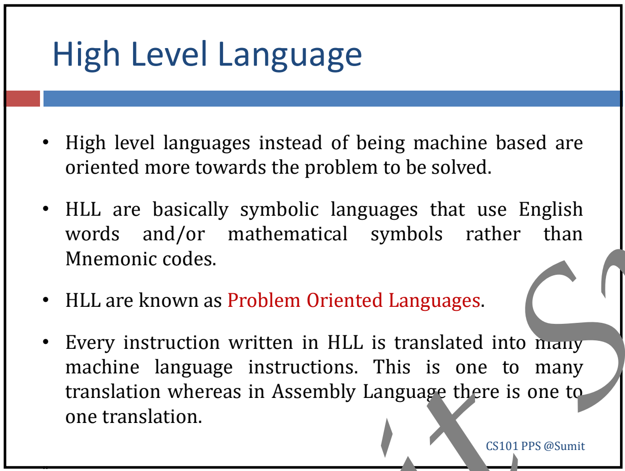
High Level Language

- **High-level** languages allow us to write computer code using instructions resembling everyday spoken language (for example: **print**, **if**, **while**) which are then **translated** into machine language to be executed.
- Programs written in a **high-level** language need to be translated into **machine language** before they can be executed.
- Some programming languages use a **compiler** to perform this translation and others use an **interpreter**.

"

CS101 PPS @Sumit

68

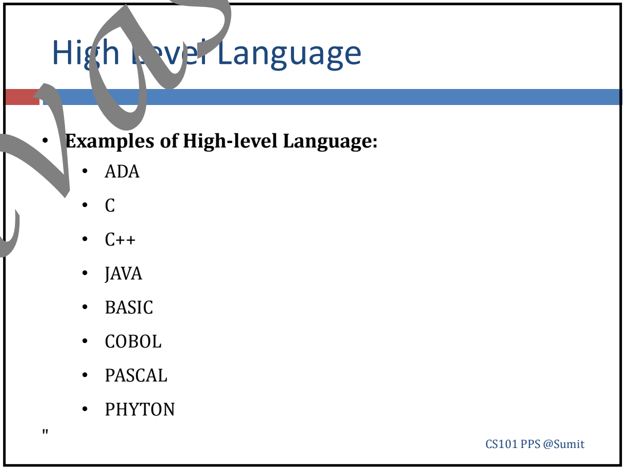


High Level Language

- High level languages instead of being machine based are oriented more towards the problem to be solved.
- HLL are basically symbolic languages that use English words and/or mathematical symbols rather than Mnemonic codes.
- HLL are known as Problem Oriented Languages.
- Every instruction written in HLL is translated into many machine language instructions. This is one to many translation whereas in Assembly Language there is one to one translation.

CS101 PPS @Sumit

69



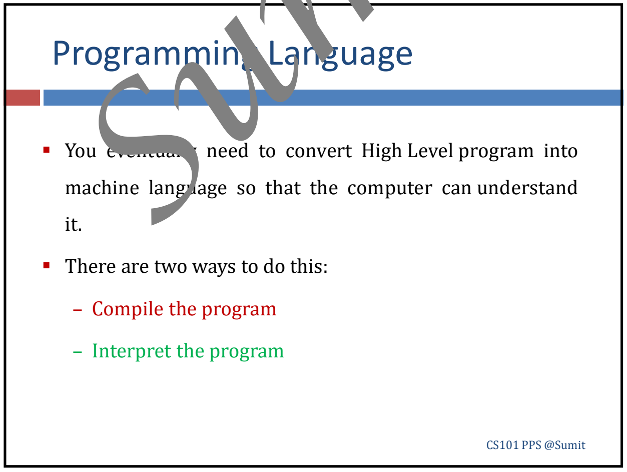
High Level Language

- **Examples of High-level Language:**
  - ADA
  - C
  - C++
  - JAVA
  - BASIC
  - COBOL
  - PASCAL
  - PHYTON

"

CS101 PPS @Sumit

70

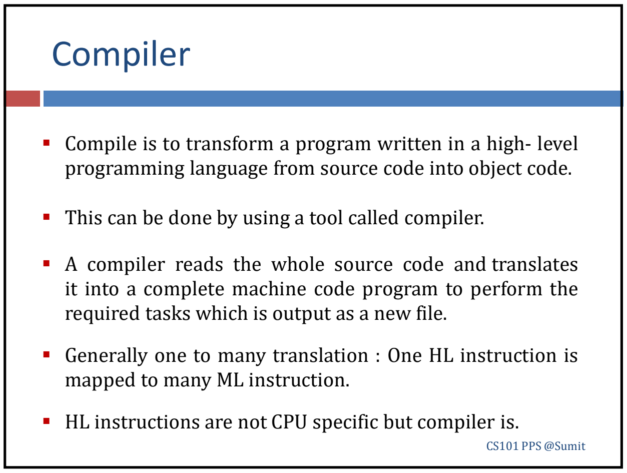


Programming Language

- You eventually need to convert High Level program into machine language so that the computer can understand it.
- There are two ways to do this:
  - Compile the program
  - Interpret the program

CS101 PPS @Sumit

71



Compiler

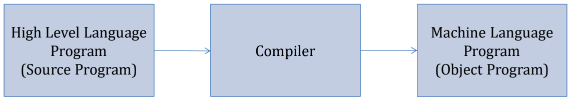
- Compile is to transform a program written in a high- level programming language from source code into object code.
- This can be done by using a tool called compiler.
- A compiler reads the whole source code and translates it into a complete machine code program to perform the required tasks which is output as a new file.
- Generally one to many translation : One HL instruction is mapped to many ML instruction.
- HL instructions are not CPU specific but compiler is.

CS101 PPS @Sumit

72

Compiler

- The translator program that translates the instructions of HLL into Machine Language is called Compiler.


High Level Language Program (Source Program)
Compiler
Machine Language Program (Object Program)

CS101 PPS @Sumit

73

Interpreter

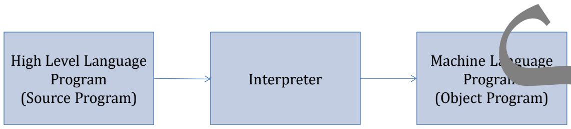
- Interpreter is a program that executes instructions written in a high-level language.
- An interpreter reads the source code one instruction or line at a time, converts this line into machine code and executes it.
- Example: JavaScript, VBScript, PHP, …

CS101 PPS @Sumit

74

Interpreter

- An Interpreter is a type of translator used for translating HLL into Machine Code.
- It takes one statement of HLL and translates it into a Machine instruction which is immediately executed.


High Level Language Program (Source Program)
Interpreter
Machine Language Program (Object Program)

CS101 PPS @Sumit

75

Difference between Interpreter and a Compiler

- In case of Compiler, whole source program is translated into equivalent machine language program. The object code thus obtained is permanently saved for future use. So, repeated compilation is not necessary whereas in Interpreter no object code is saved because translation and execution process alternate.
- **Advantage** of an Interpreter over Compiler is that it responses fast to changes in source program.
- Interpreters are easy to write and do not require large memory space.
- **Disadvantage** of interpreter over compiler is that interpreter is a time consuming translation method because each statement must be translated every time it is executed from source program.

CS101 PPS @Sumit

76

Language Processors

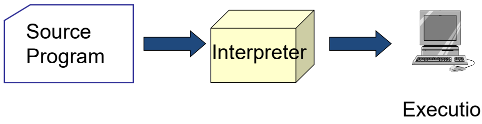
- Assemblers, Interpreters and Compilers are System Software that translate a source program into object program and are known as Language Processors.

CS101 PPS @Sumit

77

Language processing: Interpreted

- Interpreted: BASIC, Postscript, Scheme, Matlab
- The *interpreter* reads the source program and executes each command as it reads.
- The interpreter "knows" how to perform each instruction in the language.



CS101 PPS @Sumit

78

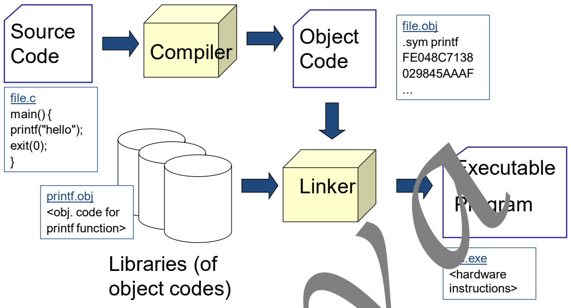
Language processing: Compiled

- Compiled: C/C++, Pascal, Fortran
- The *compiler* converts source code into machine language to create an *object code* file.
- A *linker* combines object code files and pre-compiled *libraries* to produce an executable program (machine language).

CS101 PPS @Sumit

79

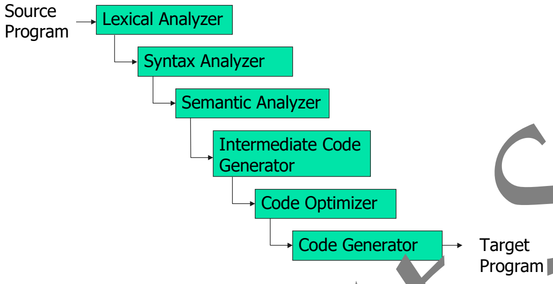
Language processing: Compiled



CS101 PPS @Sumit

80

Typical Phases of a Compiler



CS101 PPS @Sumit

81

High Level Language

- Advantages:
  - Machine independent.
  - Easy to learn and use.
  - Fewer errors.
  - Easier to maintain.

CS101 PPS @Sumit

82

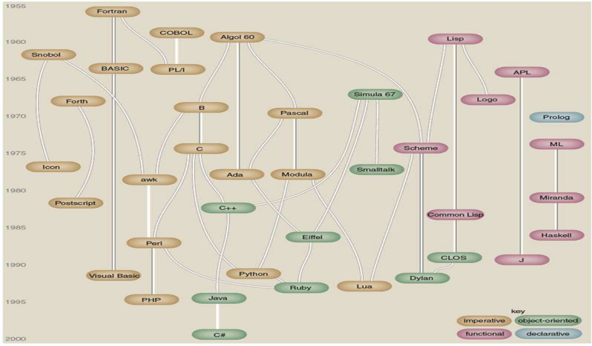
Comparison

	Machine Language	Assembly Language	High-level Languages
Time to execute	Since it is the basic language of the computer, it does not require any translation, and hence ensures better machine efficiency. This means the programs run faster.	A program called an 'assembler' is required to convert the program into machine language. Thus, it takes longer to execute than a machine language program.	A program called a compiler or interpreter is required to convert the program into machine language. Thus, it takes more time for a computer to execute.
Time to develop	Needs a lot of skill, as instructions are very lengthy and complex. Thus, it takes more time to program.	Simpler to use than machine language, though instruction codes must be memorized. It takes less time to develop programs as compared to machine language.	Easiest to use. Takes less time to develop programs and, hence, ensures better program efficiency.

CS101 PPS @Sumit

83

Programming Language



84

Implementation of Language

Implement a Language

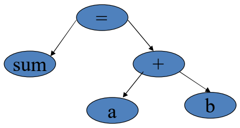
- Generally, the action of any translating program can be divided into three phases
  - Scanning
  - Parsing
  - Code generation

Implement a Language - Scanning

- Scanning process: a long string of characters is broken into tokens.
- Example: sum = a + b is broken into 5 tokens sum, =, a, +, b
- A token is the smallest meaningful unit of information.

Implement a Language - Parsing

- Parsing: the string of tokens is transformed into a syntactic structure.
- What happens in a compiler or interpreter is that the list of tokens is converted to a parse tree in memory via a complicated algorithm.



End of Today's Lecture

Doubts && Queries?

THANK YOU

A language that adopts the original simple and elegant ideas, while eliminating the complexity