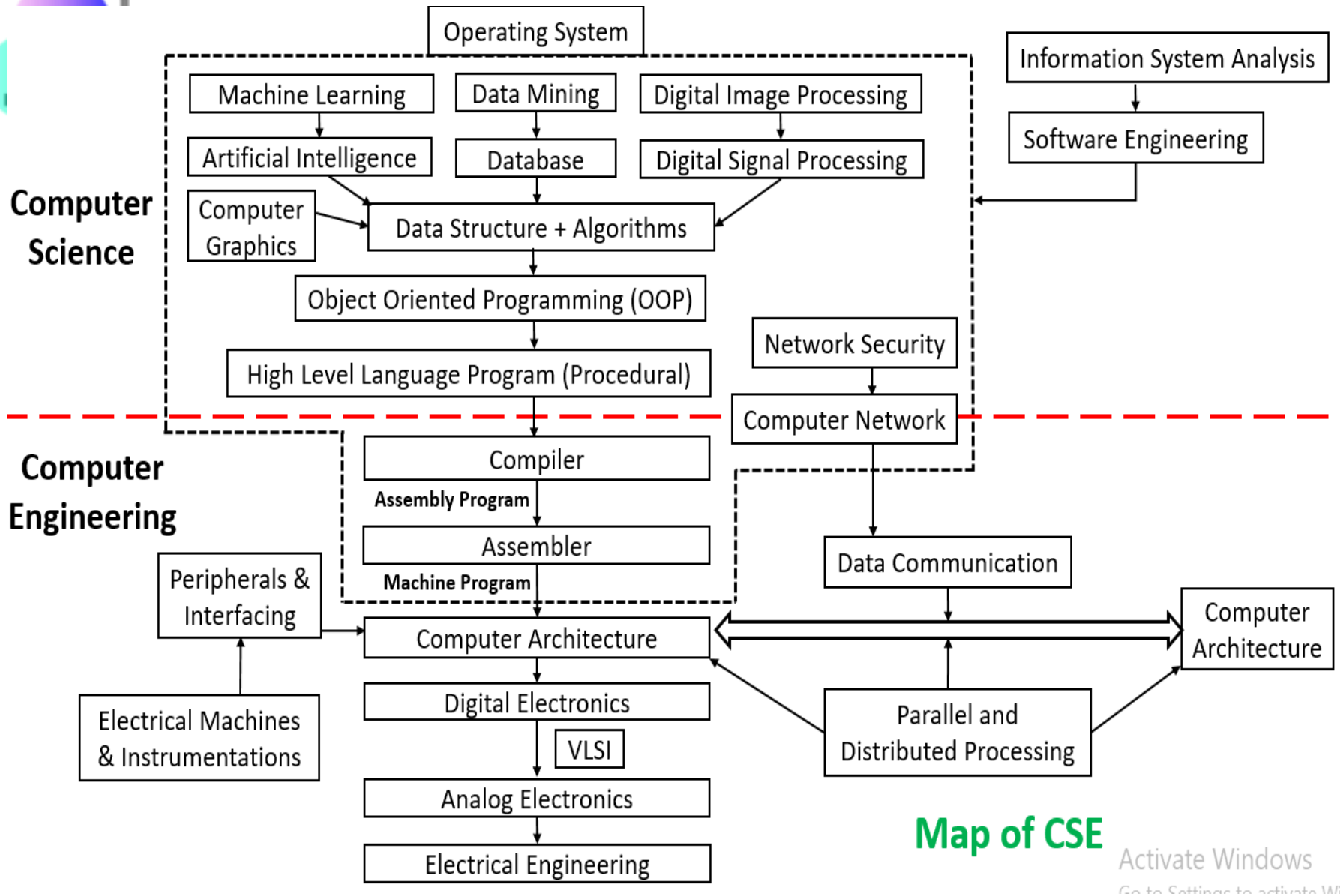
Computer Fundamentals and Ethics
Course No: CSE 1100

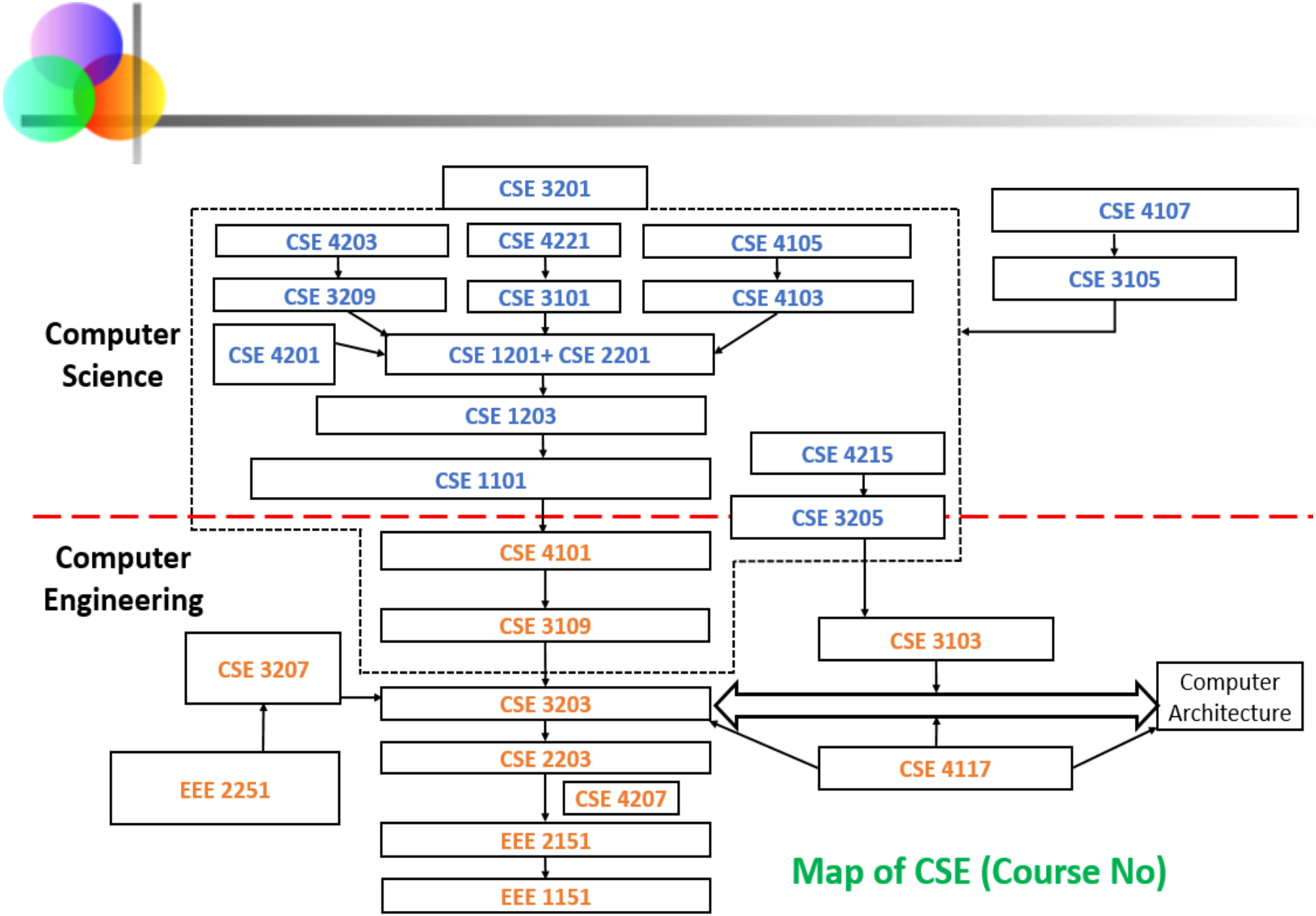
Topic : Software (Part-1)

Emrana Kabir Hashi

Assistant professor

Department of CSE, RUET




CSE 3201
CSE 4107
CSE 4203
CSE 4221
CSE 4105
CSE 3105
CSE 3209
CSE 3101
CSE 4103
Computer Science
CSE 4201
CSE 1201+ CSE 2201
CSE 1203
CSE 4215
CSE 1101
CSE 3205
CSE 4101
Computer Engineering
CSE 3109
CSE 3103
CSE 3207
Computer Architecture
CSE 3203
CSE 2203
CSE 4117
EEE 2251
CSE 4207
EEE 2151
EEE 1151
Map of CSE (Course No)

Programming Language

Programming Language

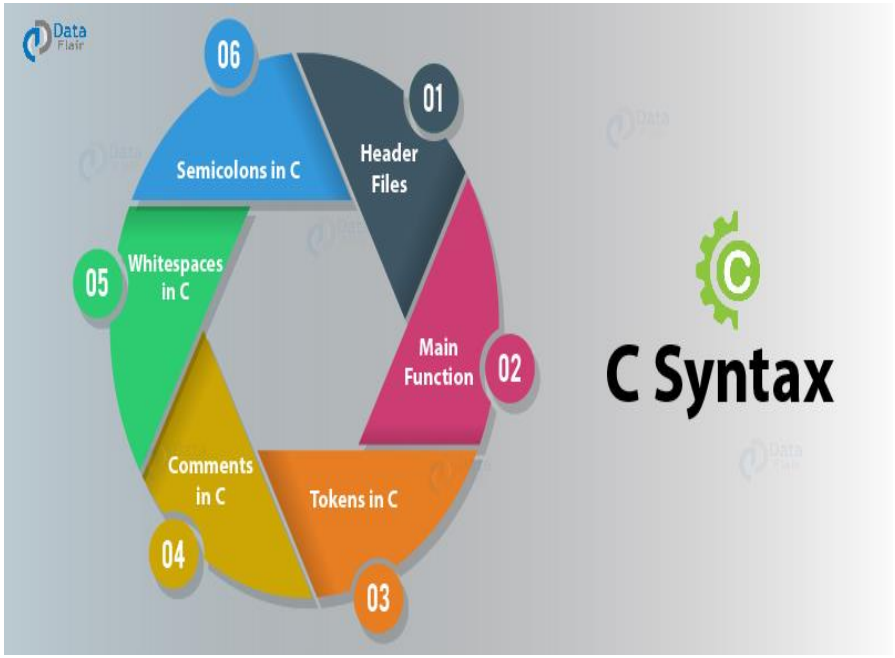
- A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.
- Coding schemes used to write both systems and application software.
- A programming language is an artificial language that can be used to control the behavior of a machine, particularly a computer.
- Programming languages, like human languages, are defined through the use of syntactic and semantic rules, to determine structure and meaning respectively.
- Programming languages are used to facilitate communication about the task of organizing and manipulating information, and to express algorithms precisely.

Programming Language

- In computer science, the syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be correctly structured statements or expressions in that language.

- The C basic syntax consists of header files, main function, and program code.

Programming Language




```
including header files
#include<stdio.h>
int main()                    main() function must be there
{
 int i ;
 // Asking user for value     Single line comment
 printf("Enter a value") ;
 scanf("%d", &i) ;
 getch() ;                    semicolon after each statement
 return 0;
}
program enclosed within curly braces
```

Programming Language

- In computer science, the term semantics refers to the meaning of language constructs, as opposed to their form .

Semantic Errors:

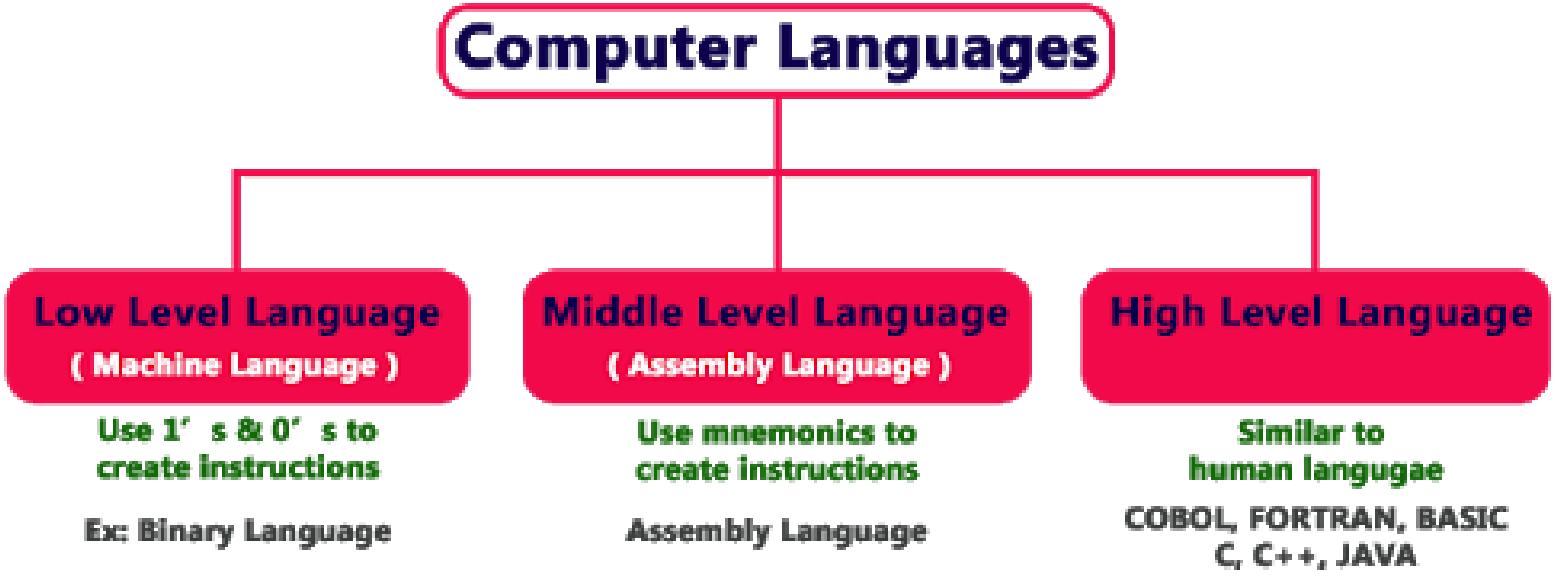
Errors recognized by semantic analyzer are as follows:

- Type mismatch
- Undeclared variables
- Reserved identifier misuse

Non-computational languages

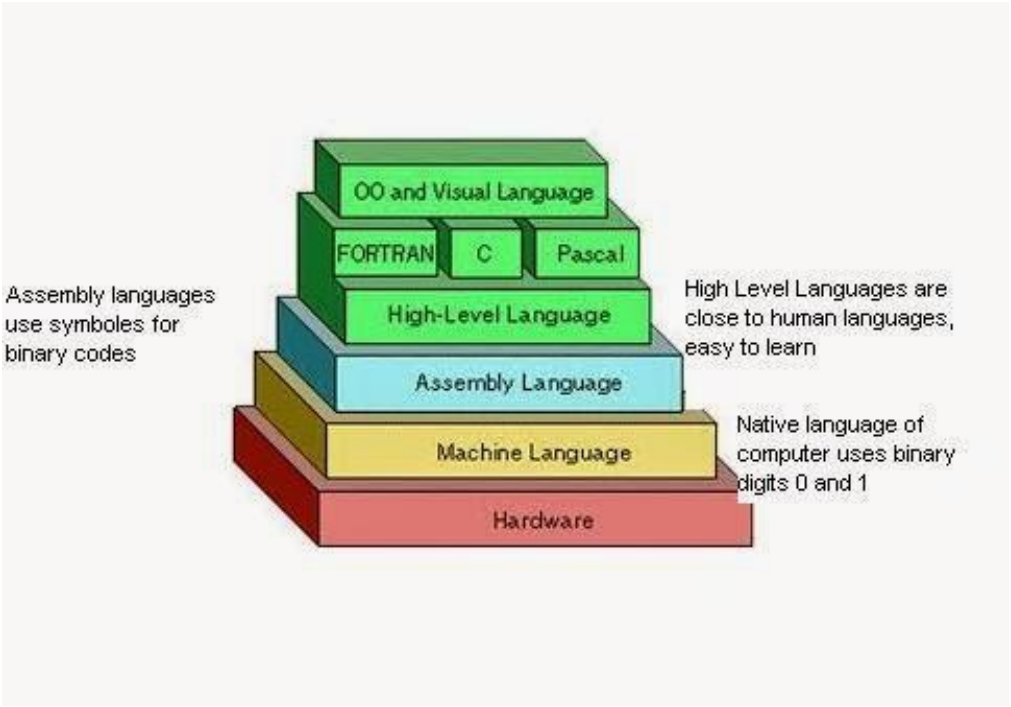
- Non-computational languages, such as markup languages like HTML or formal grammars are usually not considered programming languages.
- Often a programming language is embedded in the non-computational language.

Computer Languages



Machine language

- 1st generation programming language.
- Machine languages are the only languages understood by computers.
- Considered a low-level language because it involves basic coding using the binary symbols 1 and 0



Machine language

- While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.
- For example, an x86/IA-32 processor can execute the following binary instruction as expressed in machine language:

  Binary: 10110000 01100001 (Hexadecimal: 0xb061)

- It does not needs any translator program.

Machine language

Machine language

- **Advantage:**
  - The only advantage is that program of machine language run very fast because no translation program is required for the CPU
- **Disadvantages:**
  - It is very difficult to program in machine language. The programmer has to know details of hardware to write program.
  - The programmer has to remember a lot of codes to write a program which results in program errors.
  - It is difficult to debug the program.

Assembly language

- 2nd generation language
- An assembly language is a low-level language for programming computers.
- It implements a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture.
- Replaced binary digits with mnemonics (e.g., “ADD”) programmers could more easily understand

Assembly language

Assembly language

```
name "add"
mov al, 5                ; bin=00000101b
mov bl, 10           ; hex=0ah or bin=00001010b
add bl, al           ; 5 + 10 = 15 (decimal) or hex=0fh
                          or  bin=00001111b
```

Assembly language

- **Advantage:**
  - Assembly Language is easier to understand and saves a lot of time and effort of the programmer.
  - It is easier to correct errors and modify program instructions.
  - Assembly Language has the same efficiency of execution as the machine level language. Because this is one-to-one translator between assembly language program and its corresponding machine language program.
- **Disadvantages:**
  - One of the major disadvantages is that assembly language is machine dependent. A program written for one computer might not run in other computers with different hardware configuration.

High level language

- High-level languages are relatively easy to learn because the instructions bear a close resemblance to everyday language, and because the programmer does not require a detailed knowledge of the internal workings of the computer.
- Each instruction in a high-level language is equivalent to several machine-code instructions, therefore it is more compact than equivalent low-level programs.
- High-level languages are used to solve problems and are often described as problem-oriented languages

High level language

- **Examples of HLL:**
  - BASIC was designed to be easily learnt by first-time programmers;
  - COBOL is used to write programs solving business problems;
  - FORTRAN is used for programs solving scientific and mathematical problems.
  - With the increasing popularity of windows-based systems, the next generation of programming languages was designed to facilitate the development of GUI interfaces;
  - For example, Visual Basic wraps the BASIC language in a graphical programming environment.
  - Support for object-oriented programming has also become more common, for example in C++ and Java.

High level language

Example (C program to add 2 numbers):

```
#include<stdio.h>                    //header files
Void main()
{
 int a, b, c;                  // declaration of 3 variables
 printf("Enter two numbers:\n");
 Scanf("%d", &a);                 // read 1st number
 Scanf("%d", &b);                // read 2nd number
 c=a+b;                     // compute the sum
 printf("Sum of 2 numbers is %d", c); //print sum
}
```

High level language

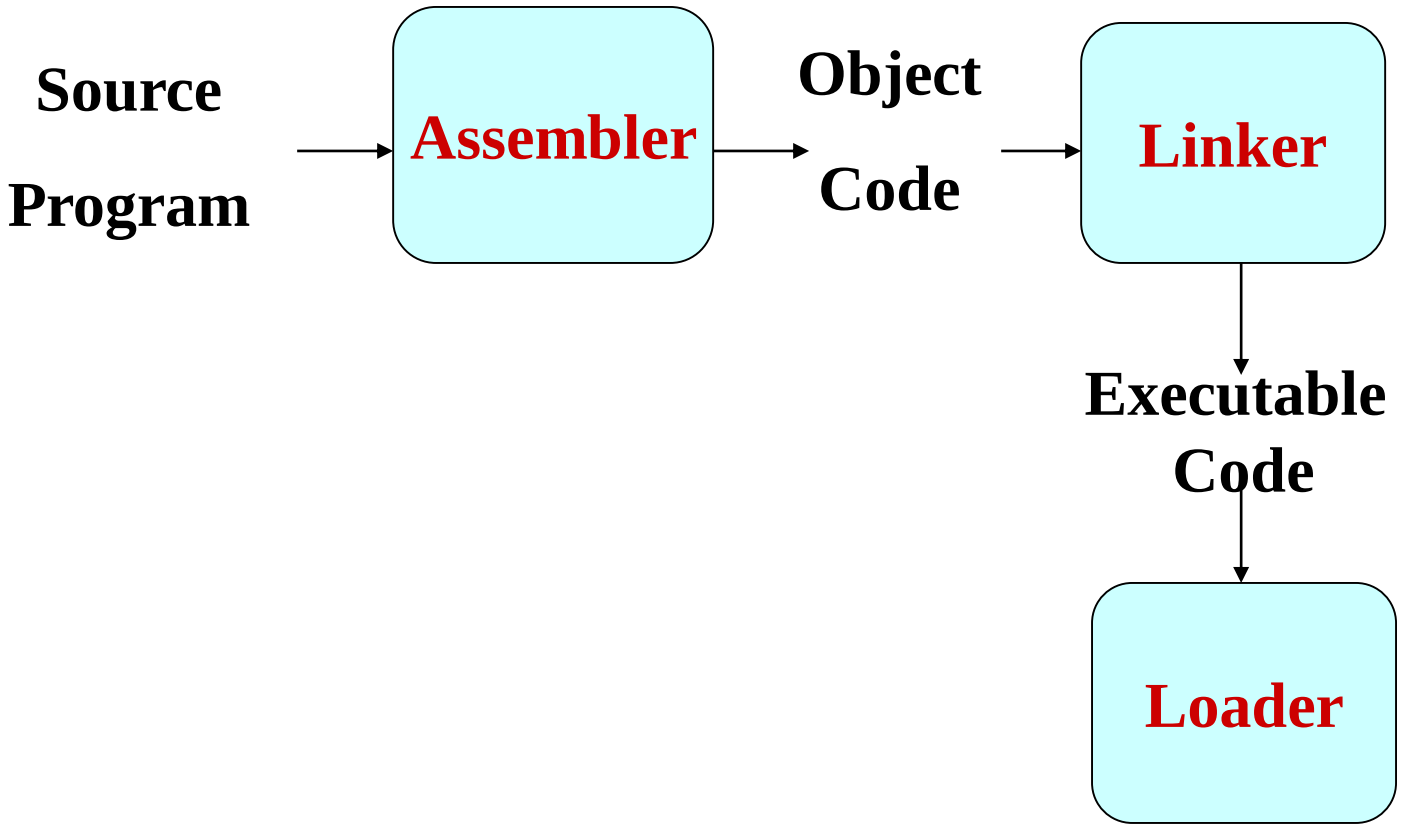
- **Advantage:**
  - Higher level languages have a major advantage over machine and assembly languages that higher level languages are easy to learn and use. It is because that they are similar to the languages used by us in our day to day life.

Assembler

- Is used to translate assembly language statements into the target computer's machine code.
- The assembler performs a more or less isomorphic translation (a one-to-one mapping) from mnemonic statements into machine instructions and data.
- Fundamental functions
  - translating mnemonic operation codes to their machine language equivalents
  - assigning machine addresses to symbolic labels

Assembler

Role of Assembler

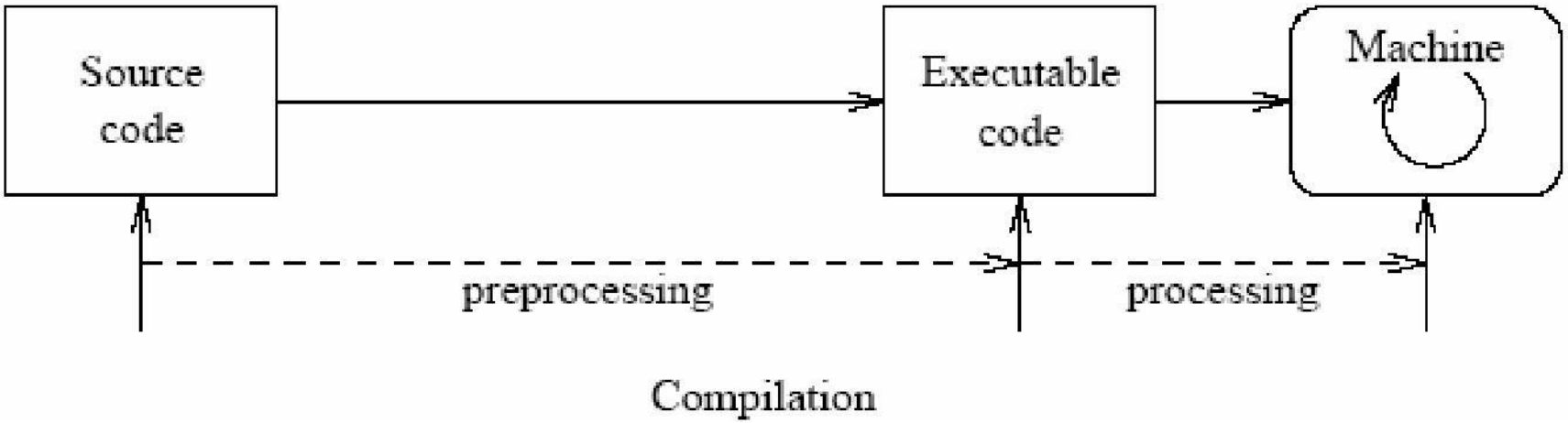


Compiler

Compiler:

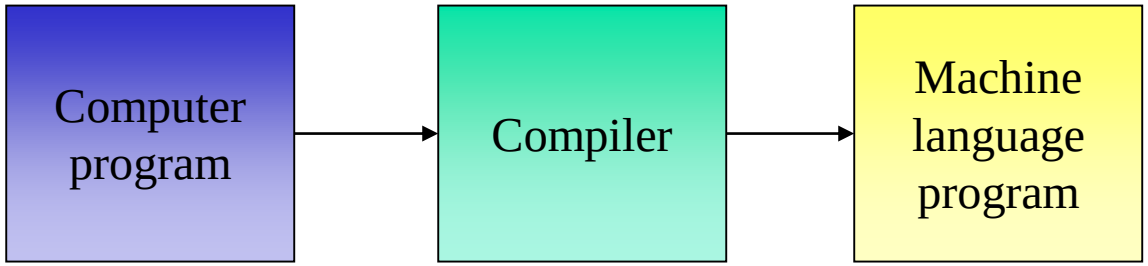
- A compiler is a program that reads a program in one language – the source language and translates into an equivalent program in another language – the target language.
- A compiler is a special program that processes statements written in a particular programming language called as source code and converts them into machine language or “machine code” that a computer’s processor uses.
- Compiler translates high level language programs directly into machine language program. This process is called compilation.

Compiler

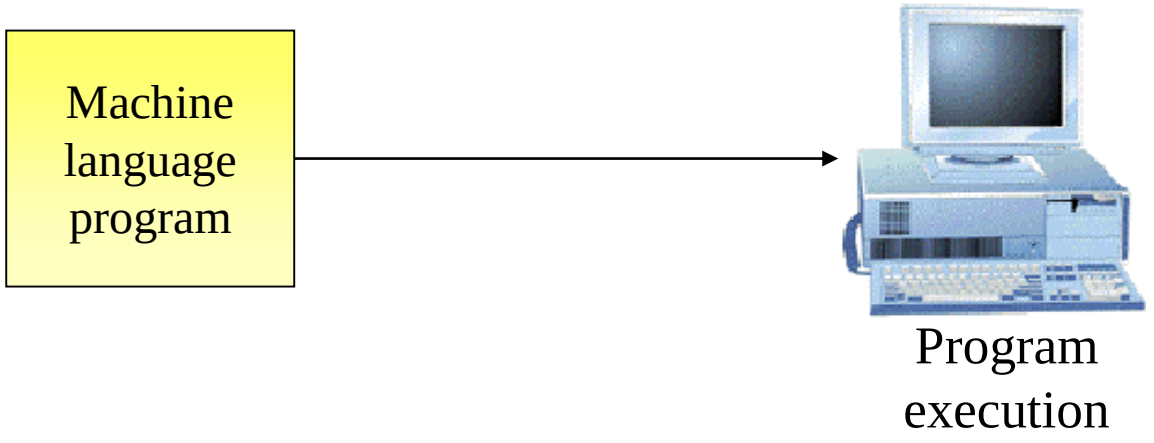

Source code
Executable code
Machine
preprocessing
processing
Compilation

Compiler

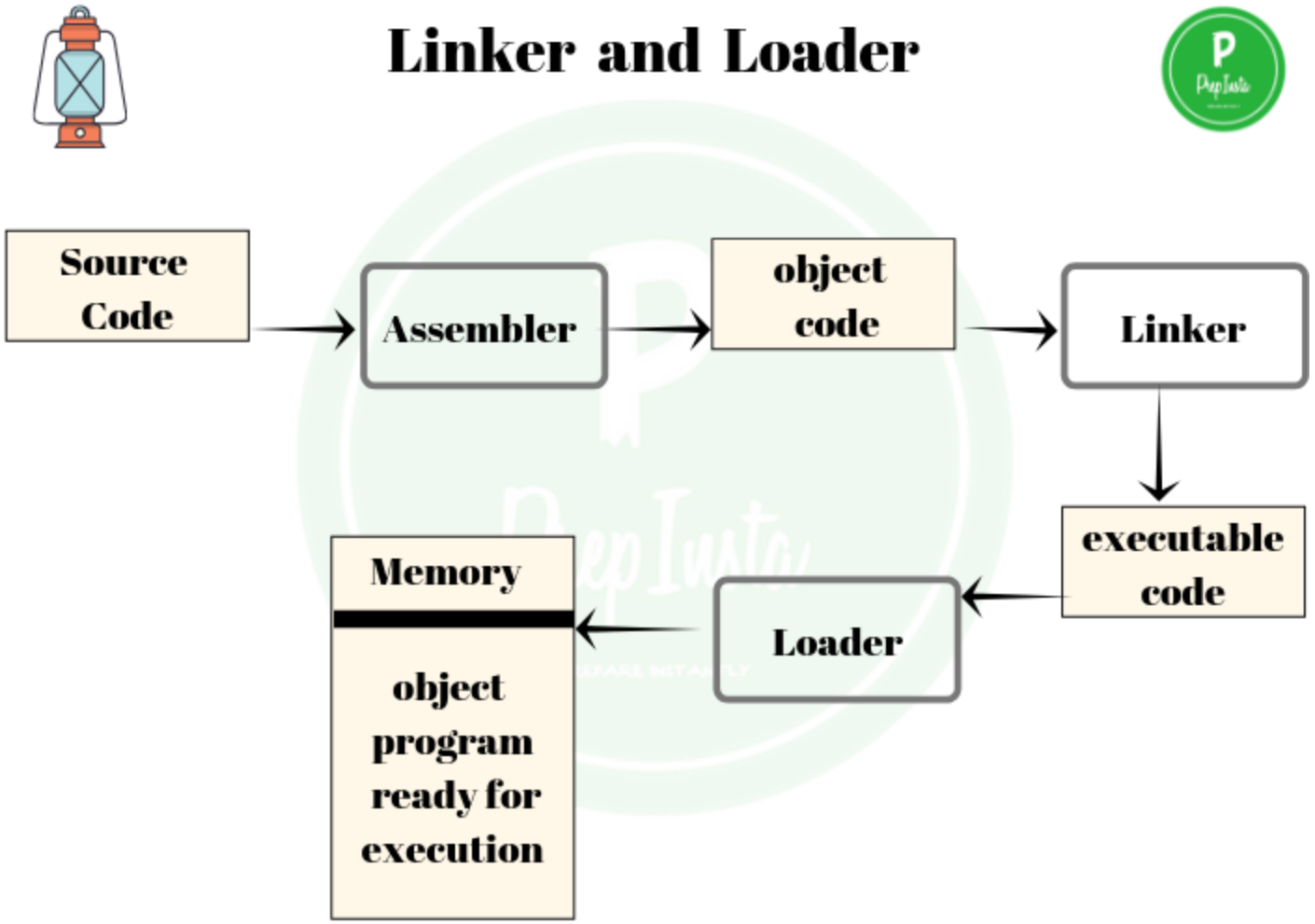
Step 1: Translate program



Step 2: Execute program



Linker and Loader



Linker

- A linker or link editor is a program that takes one or more objects generated by compilers and assembles them into a single executable program.
- Linkers can take objects from a collection called a library. The objects are program modules containing machine code and information for the linker.
- The linker takes care of arranging the objects in a program's address space.

Loader

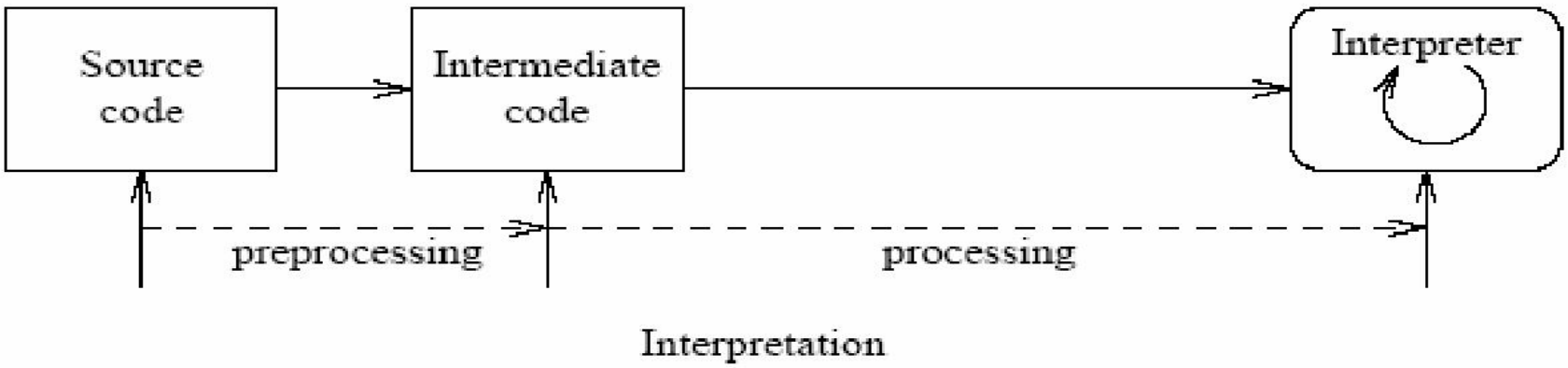
- A loader is the part of an operating system that is responsible for loading programs into memory, preparing them for execution and then executing them.
- To run an executable file, the loader must copy all the instructions into memory and direct the CPU to begin execution with the first instruction.
- As the program executes, it takes input data from source(s) and sends results to output devices.
- In Unix, the loader is the handler for the system call execve().

Interpreter

- An interpreter is a computer program that translates and executes instructions written in a computer programming language line-by-line, unit by unit etc.,
- An interpreter needs to be able to analyze, or parse, instructions written in the source language.
- Example: LISP systems

Interpreter

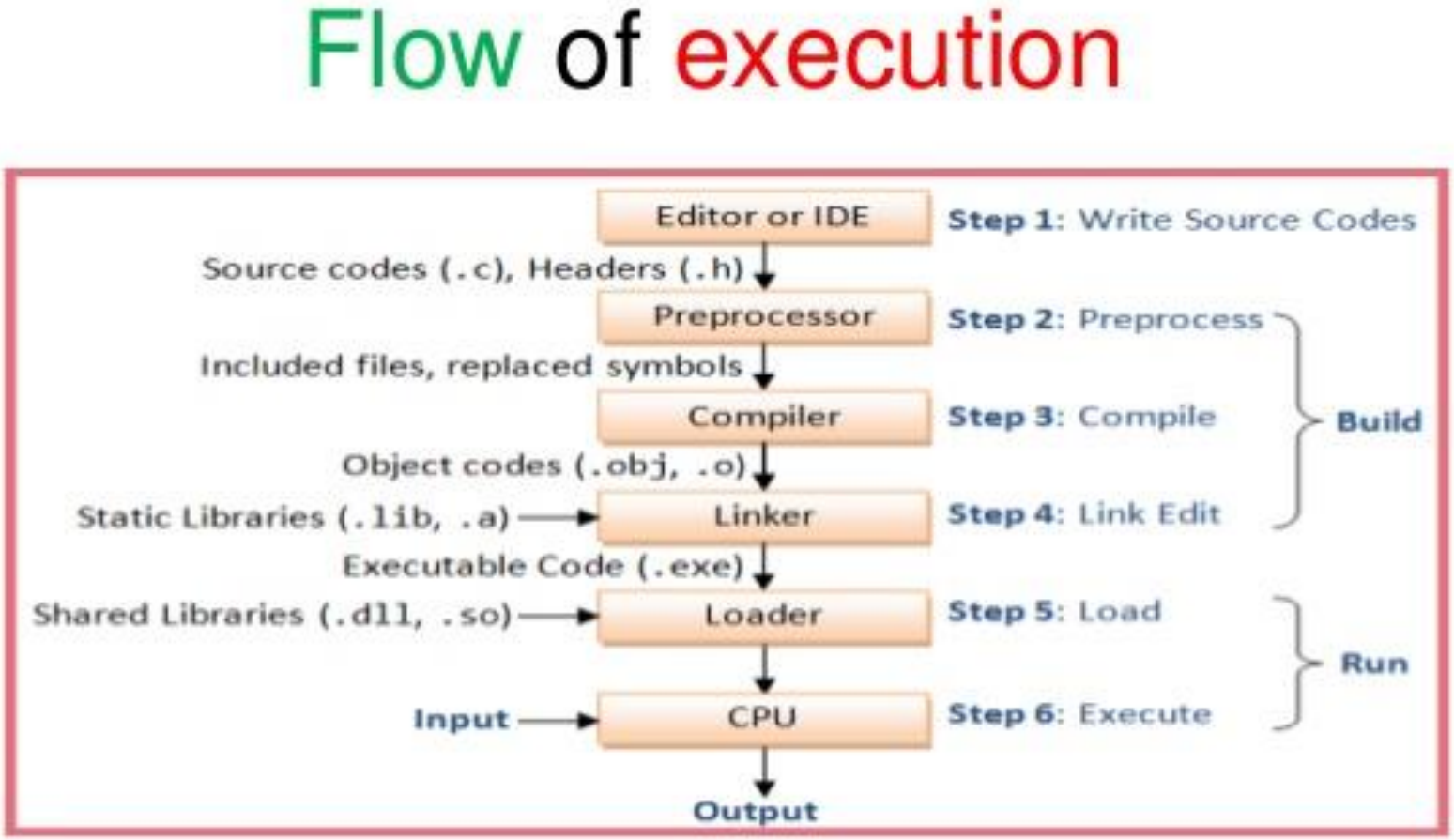
- An interpreter translates high-level instructions into an intermediate form, which it then executes.
- Compiled programs generally run faster than interpreted programs.
- The advantage of an interpreter, however, is that it does not need to go through the compilation stage during which machine instructions are generated.
- This process can be time-consuming if the program is long.



Interpreter vs Compiler

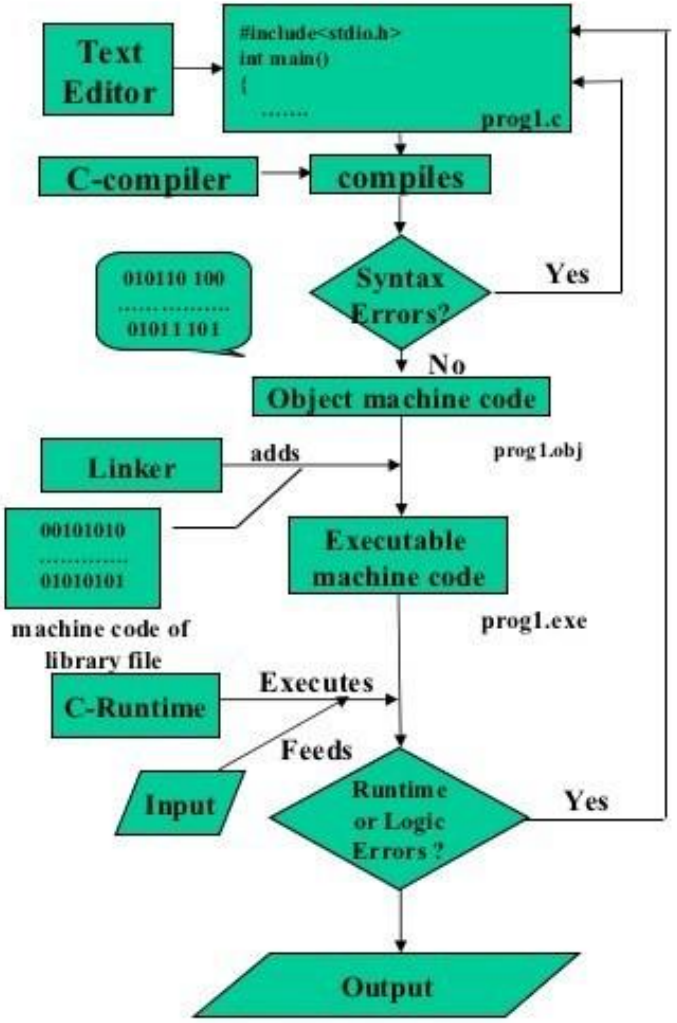
Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby, Basic use interpreters.	Programming language like C, C++,Java use compilers.

Flow of execution



Flow of execution

Executing a C program



Translators are system software used to convert high-level language program into machine-language code.

Compiler : Coverts the entire source program at a time into object code file, and saves it in secondary storage permanently. The same object machine code file will be executed several times, whenever needed.

Interpreter : Each statement of source program is translated into machine code and executed immediately. Translation and execution of each and every statement is repeated till the end of the program. No object code is saved. Translation is repeated for every execution of the source program.

Object oriented programming

- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.
- An object can be defined as a data field that has unique attributes and behavior.
- Additional benefits of OOP include code reusability, scalability and efficiency.
- The first step in OOP is to collect all of the objects a programmer wants to manipulate and identify how they relate to each other -- an exercise often known as data modeling.

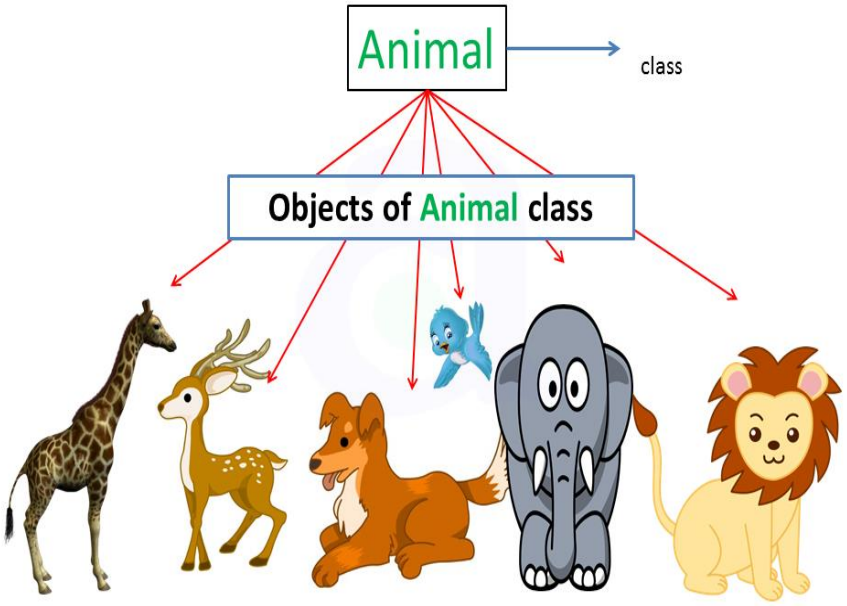
Object oriented programming

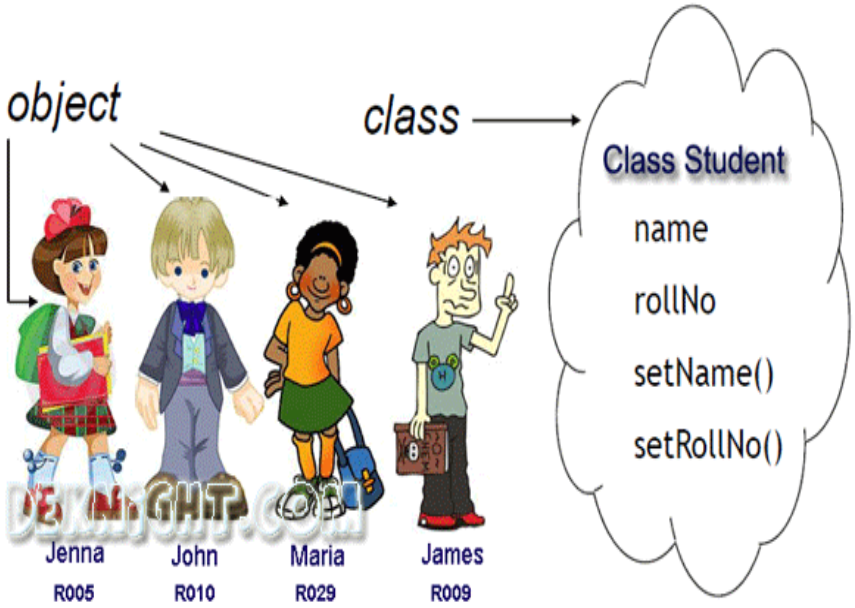
- Class: A class describes the contents of the objects that belong to it: it describes an aggregate of data fields (called instance variables), and defines the operations (called methods).

- Object: An object is an element (or instance) of a class; objects have the behaviors of their class. The object is the actual component of programs, while the class specifies how instances are created and how they behave.

- Method: A method is an action which an object is able to perform.

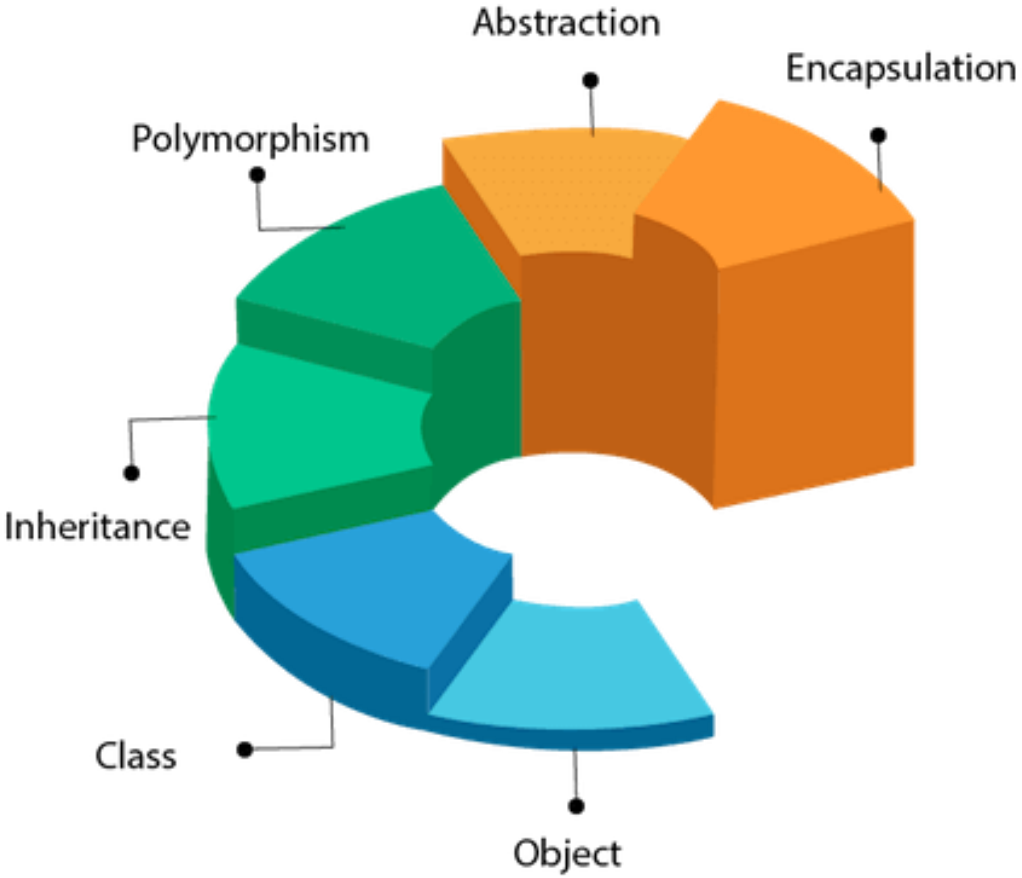
Object oriented programming





Object oriented programming

OOPs (Object-Oriented Programming System)



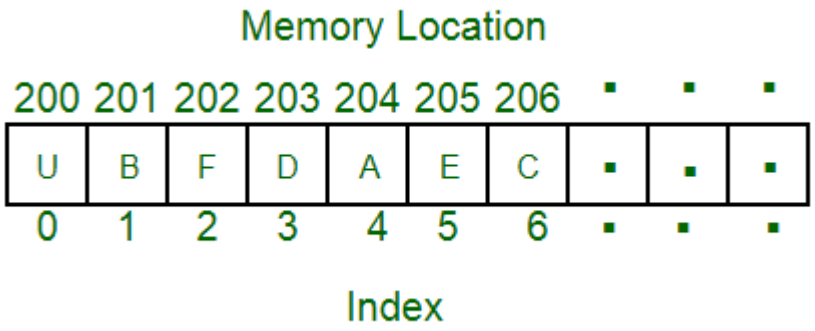
Object oriented programming

Object-oriented programming languages

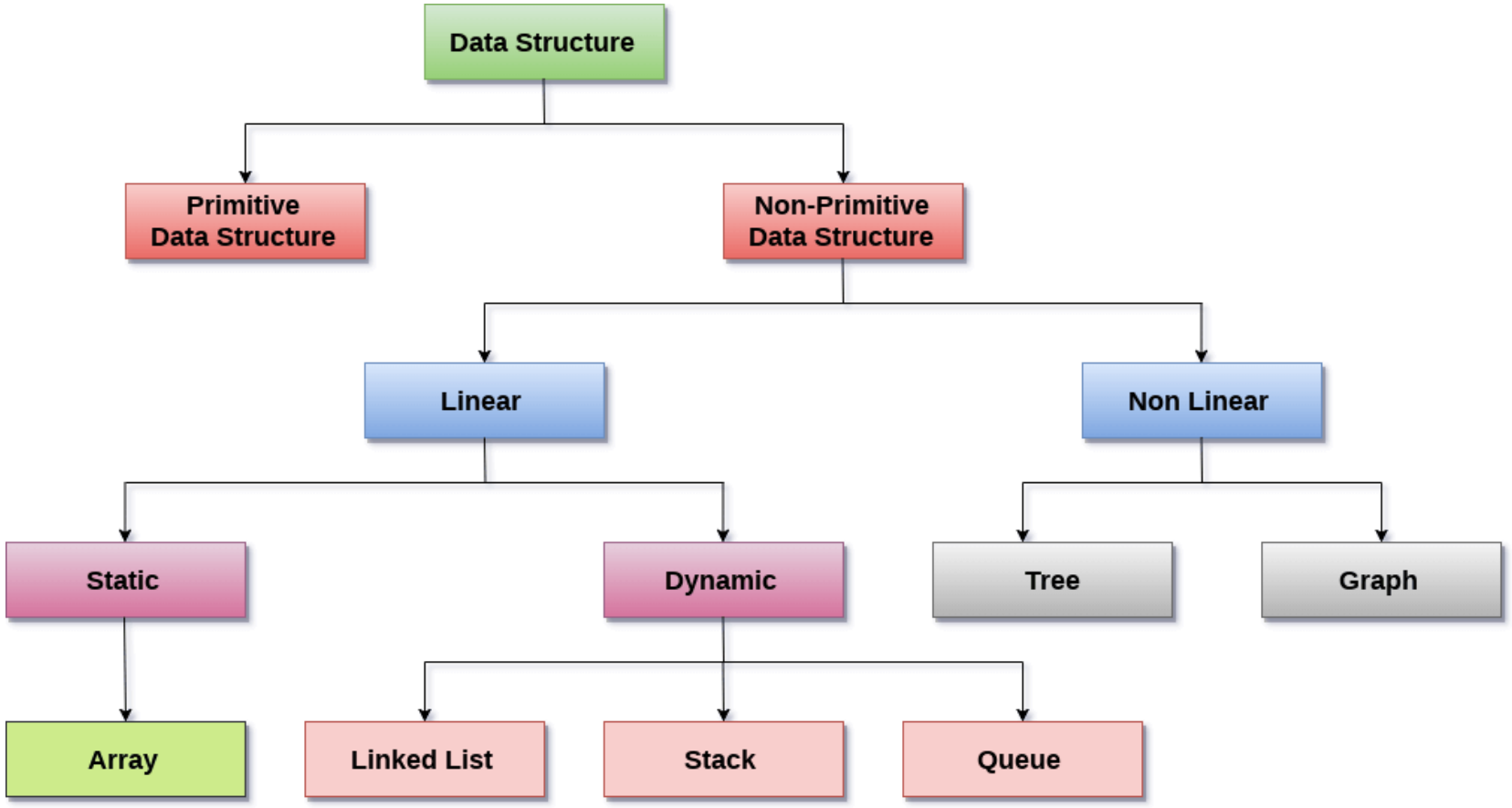
- While Simula is credited as the first object-oriented programming language, the most popular OOP languages are:
- Java
- JavaScript
- Python
- C++
- Visual Basic .NET
- Ruby
- PHP

Data Structures

- A **data structure** is a particular way of organizing data in a computer so that it can be used effectively.
- For example, we can store a list of items having the same data-type using the *array* data structure.

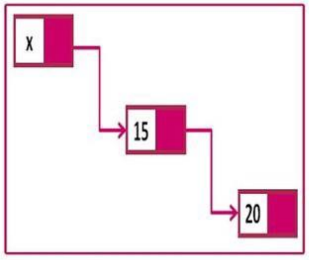


Data Structures

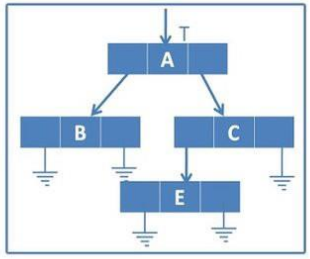


Data Structures

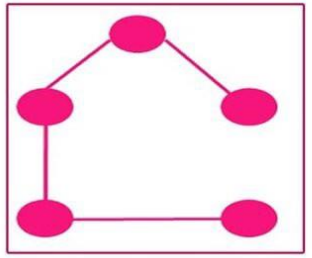
Sorting



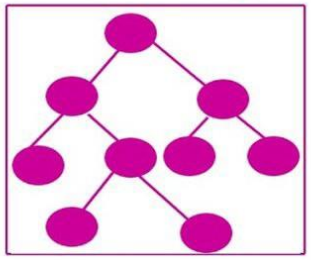
Link list



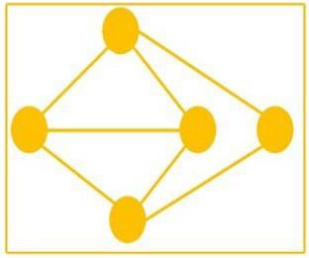
list



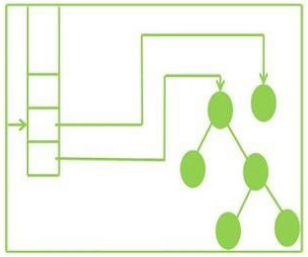
spanning tree



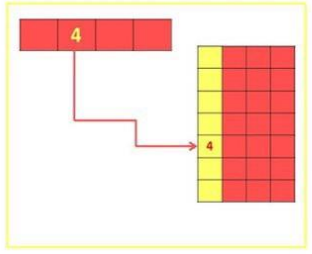
Tree



Graph



Stack



Hashing

By...navinkumardhoprephotography.com

Data Structures

Topics:

- Array
- Linked List
- Stack
- Queue
- Binary Tree
- Binary Search Tree
- Heap
- Hashing
- Graph
- Matrix
- Advanced Data Structure

Algorithms

- In mathematics and computer science, an **algorithm** is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.
- Algorithms are always unambiguous and are used as specifications for performing calculations, data processing, automated reasoning, and other tasks.

Algorithms

Typical steps in the development of algorithms:

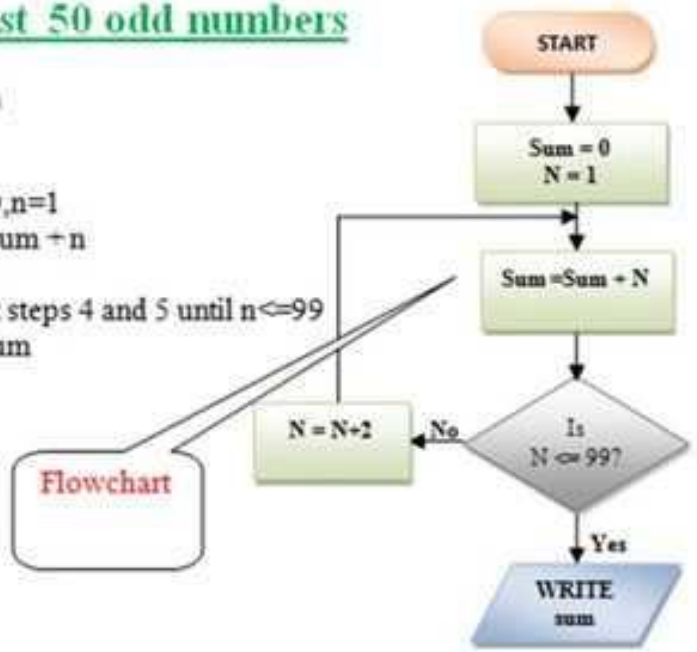
- Problem definition
- Development of a model
- Specification of the algorithm
- Designing an algorithm
- Checking the correctness of the algorithm
- Analysis of algorithm
- Implementation of algorithm
- Program testing
- Documentation preparation

Algorithms

Sum of first 50 odd numbers

Algorithm

1. Start
2. Sum=0,n=1
3. Sum=sum + n
4. n=n+2
5. Repeat steps 4 and 5 until n<=99
6. Print sum
7. Stop



Algorithms

Topics :

- Analysis of Algorithms
- Searching and Sorting
- Greedy Algorithms
- Dynamic Programming
- Pattern Searching
- Other String Algorithms
- Backtracking
- Divide and Conquer
- Geometric Algorithms
- Mathematical Algorithms
- Bit Algorithms
- Graph Algorithms
- Randomized Algorithms
- Branch and Bound
- Quizzes on Algorithms

Thank You