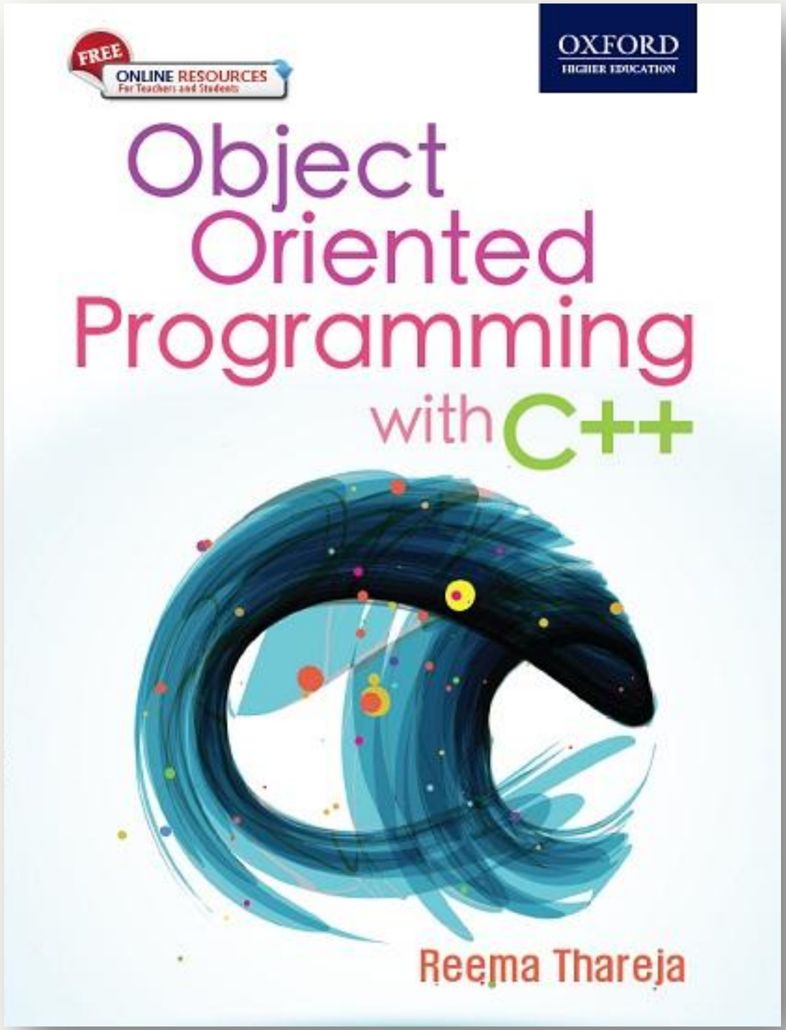

FREE
ONLINE RESOURCES
For Teachers and Students
OXFORD
HIGHER EDUCATION
Object Oriented Programming with C++
Reema Thareja

OXFORD
HIGHER EDUCATION

Object Oriented Programming with C++

Chapter One

Introduction to Object Oriented Programming

INTRODUCTION

- A programming language is a language specifically designed to express computations that can be performed by the computer.
- Programming languages are used to create programs that control the behaviour of a system, to express algorithms, or can be used as a mode of human communication.
- The term ‘programming language’ usually refers to high-level languages such as **BASIC, C, C++, COBOL, FORTRAN, ADA, and PASCAL,** to name a few.
- While high-level programming languages are easy for us to read and understand, the computer understands the machine language that consists only of numbers.

INTRODUCTION

- The determination of the language is dependent on the following factors:
- The type of computer (microcontroller, microprocessor, etc.) on which the program has to be executed.
- The type of program (system program, application program, etc.).
- The expertise of the programmer. That is, the proficiency level of a programmer in a particular language.

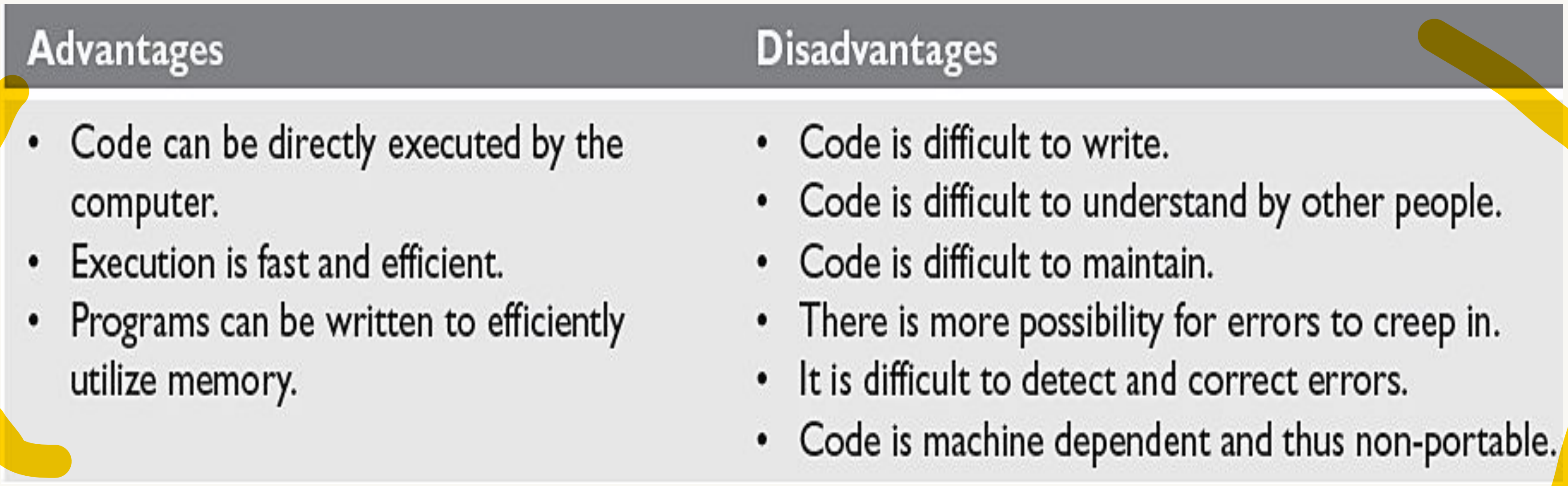
INTRODUCTION

- For example, **FORTRAN** is particularly good for processing numerical data but it does not lend itself very well to organizing large programs. Pascal can be used for writing well-structured and readable programs but it is not as flexible as C language. **C++** goes one step ahead of C by incorporating powerful object oriented features but it is complex and difficult to learn

First Generation: Machine Language

- The first program was programmed using the machine language. This is the lowest level of programming language and is the only language that a computer understands.
- All the commands and data values are expressed using 0s and 1s, corresponding to the off and on electrical states in a computer.
- In the 1950s, each computer had its own native language, and programmers had primitive systems for combining numbers to represent instructions such as *add and subtract* . In machine language, all instructions, memory locations, numbers, and characters are represented in strings of ones and zeroes.

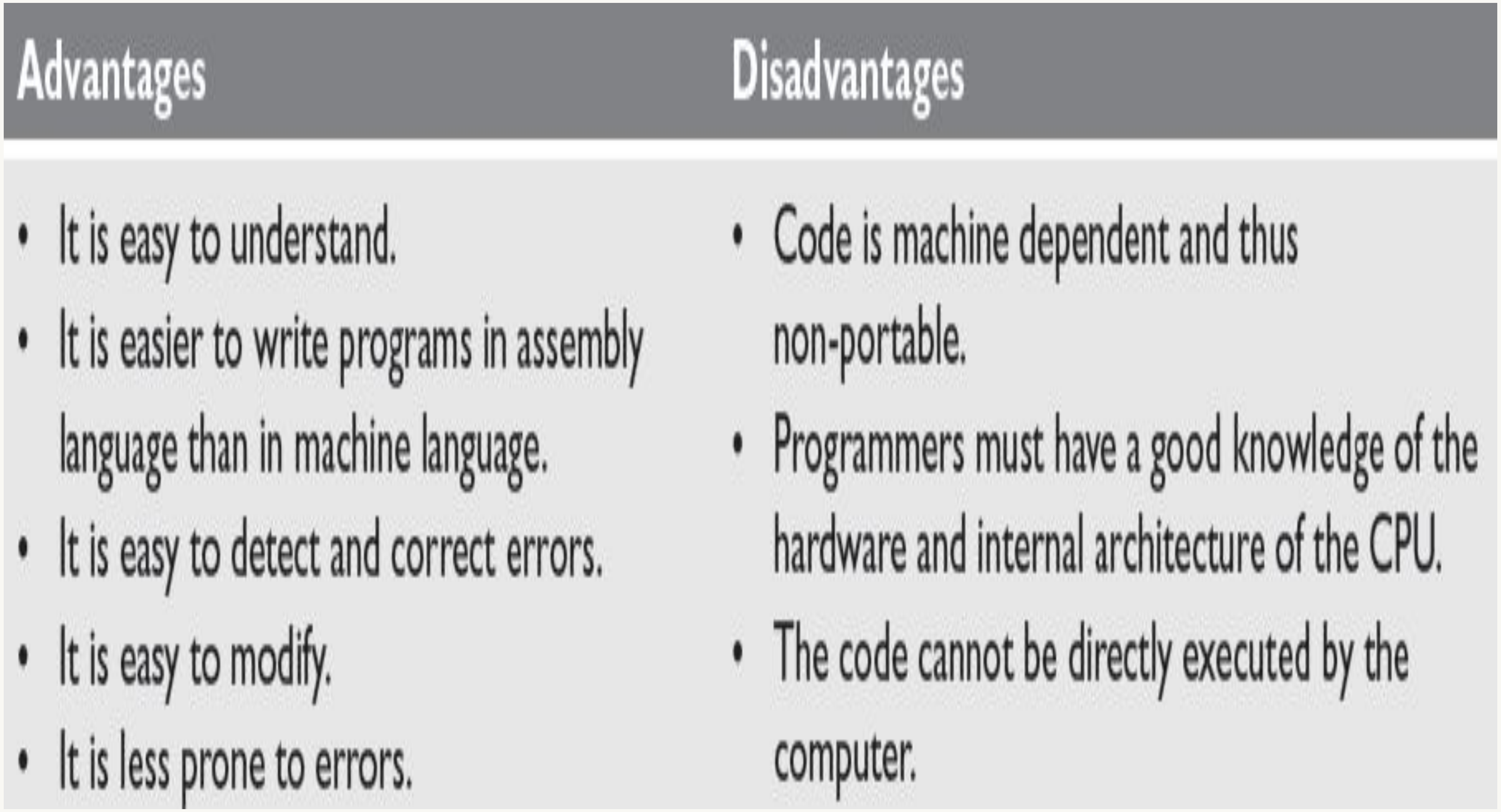
- In machine language, all instructions, memory locations, numbers, and characters are represented in strings of 0s and 1s. Although machine language programs are typically displayed with the *binary* numbers represented in *octal (base 8) or hexadecimal (base 16) number systems, these programs are* not easy for humans to read, write, or debug.



Advantages	Disadvantages
• Code can be directly executed by the computer. • Execution is fast and efficient. • Programs can be written to efficiently utilize memory.	• Code is difficult to write. • Code is difficult to understand by other people. • Code is difficult to maintain. • There is more possibility for errors to creep in. • It is difficult to detect and correct errors. • Code is machine dependent and thus non-portable.

Second Generation—Assembly Language

- 2GLs comprise the assembly languages. Assembly languages are symbolic programming languages that use symbolic notations to represent machine language instructions.
- These languages are closely connected to machine language and the internal architecture of the computer system on which they are used. Since it is close to machine language, assembly language is also a low-level language
- It uses symbolic codes, also known as mnemonic codes, which are easy-to-remember abbreviations, rather than numbers.
- Examples of these codes include ADD for add, CMP for compare, and MUL for multiply.



Advantages	Disadvantages
• It is easy to understand.	• Code is machine dependent and thus non-portable.
• It is easier to write programs in assembly language than in machine language.	• Programmers must have a good knowledge of the hardware and internal architecture of the CPU.
• It is easy to detect and correct errors.	• The code cannot be directly executed by the computer.
• It is easy to modify.	
• It is less prone to errors.	

Second Generation—Assembly Language

- Assembly language programs consist of a series of individual statements or instructions to instruct the computer what to do. Basically, an assembly language statement consists of a label, an operation code, and one or more *operands* .
- Labels are used to identify and refer instructions in the program. The operation code (opcode) is a mnemonic that specifies the operation to be performed, such as *move* , *add* , *subtract* , *or compare* .

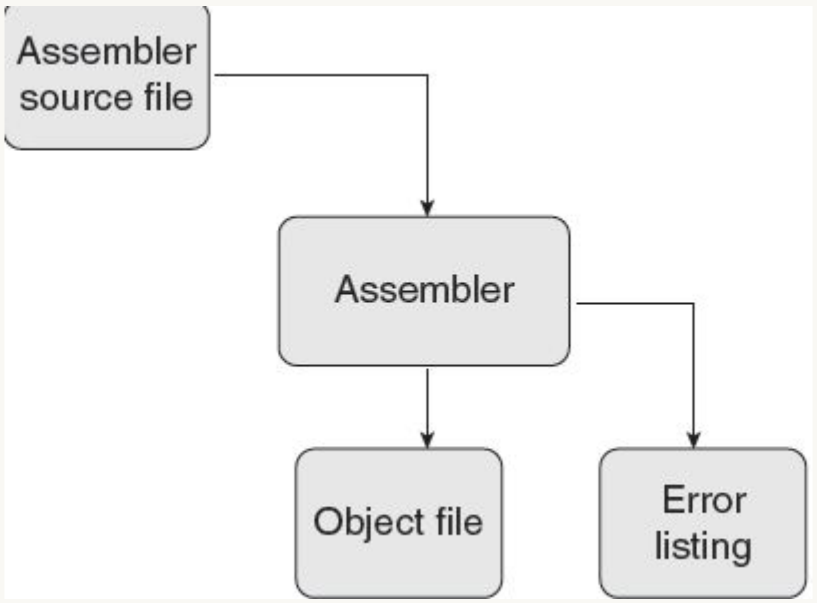
Assembler

- Since computers can execute only codes written in machine language, a special program, called the assembler, is required to convert the code written in assembly language into an equivalent code in machine language, which contains only 0s and 1s.
- There is a one-to-one correspondence between the assembly language code and the machine language code. However, if there

  is an error, the assembler gives a list of errors. The object file is

  created only when the assembly language code is free from errors. The object file can be executed as and when required.

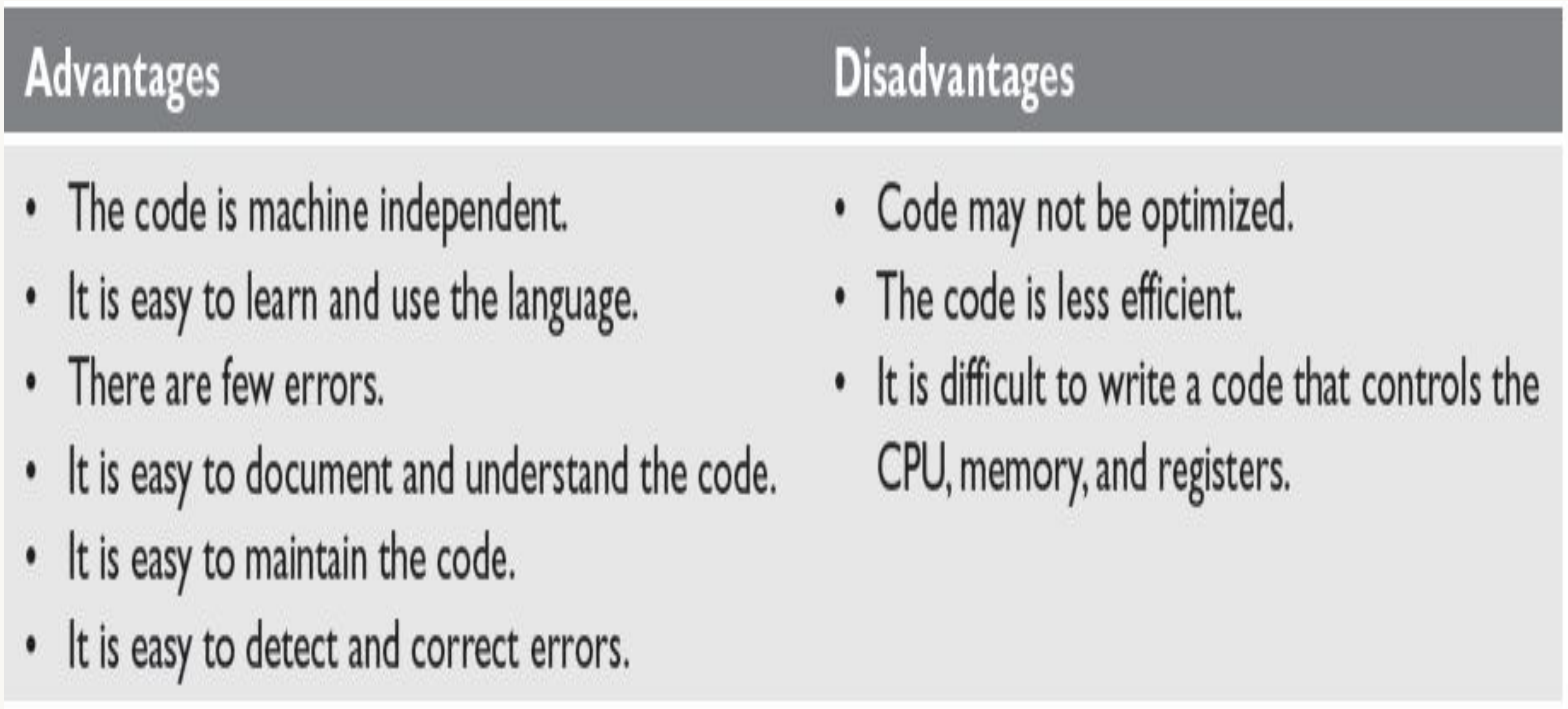
Assembler


Assembler source file
Assembler
Object file
Error listing

Third Generation: High-level Language

- The third generation was introduced to make the languages more programmer friendly .
- A statement written in a high-level programming language will expand into several machine language instructions.
- A translator is needed to translate the instructions written in a high-level language into the computer-executable machine language. Such translators are commonly known as interpreters and compilers. Each high-level language has many compilers, and there is one for each type of computer.

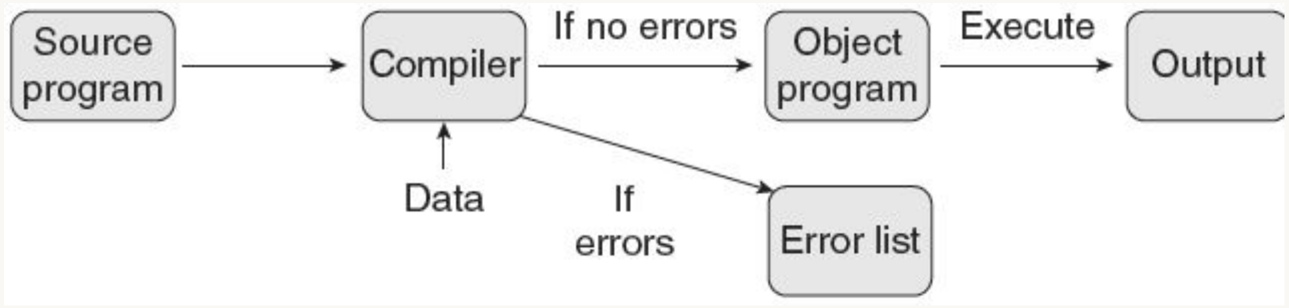
Third Generation: High-level Language contd.



Advantages	Disadvantages
• The code is machine independent.	• Code may not be optimized.
• It is easy to learn and use the language.	• The code is less efficient.
• There are few errors.	• It is difficult to write a code that controls the CPU, memory, and registers.
• It is easy to document and understand the code.	
• It is easy to maintain the code.	
• It is easy to detect and correct errors.	

Compiler

- A compiler is a special type of program that transforms the source code written in a programming language (the *source language*) *into machine language, which uses only two digits—0 and 1 (the target language)*.
- *The resultant code in 0s and 1s is known as the object code.*
- *The object code is* used to create an executable program.



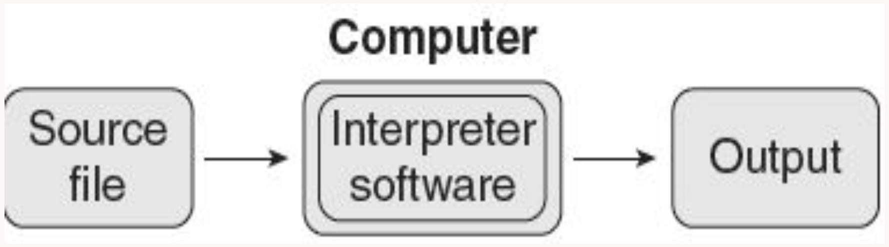
Compile Time Errors

- If the source code contains errors, then the compiler will not be able to do its intended task. Errors that limit the compiler in understanding a program are called syntax errors.
- Examples of syntax errors are spelling mistakes, typing mistakes, illegal characters, and use of undefined variables. The other type of error is the logical error, which occurs when the program does not function accurately.
- Logical errors are much harder to locate and correct than syntax errors.
- Whenever errors are detected in the source code, the compiler generates a list of error messages indicating the type of error and the line in which the error has occurred.

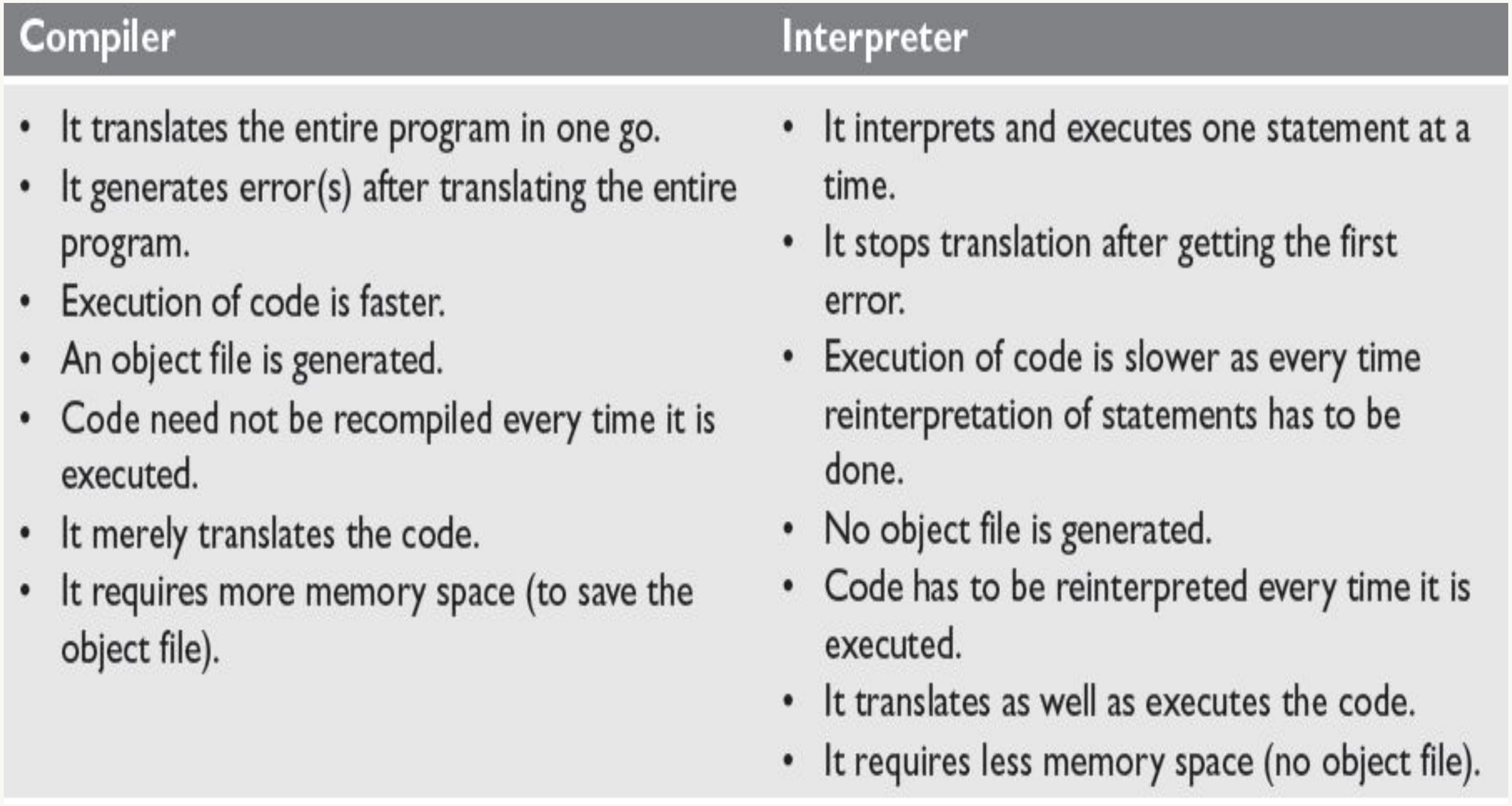
Interpreter

- The interpreter executes instructions written in a high-level language.
- A program written in a high-level language can be executed in any of the two ways—by compiling the program or by passing the program through an interpreter.
- Interpreter translates the instructions into an intermediate

  form, which it then executes. The interpreter takes one statement of high-level code, translates it into the machine level code, executes it, and then takes the next statement and repeats the process until the entire program is translated.



Compiler vs Interpreter

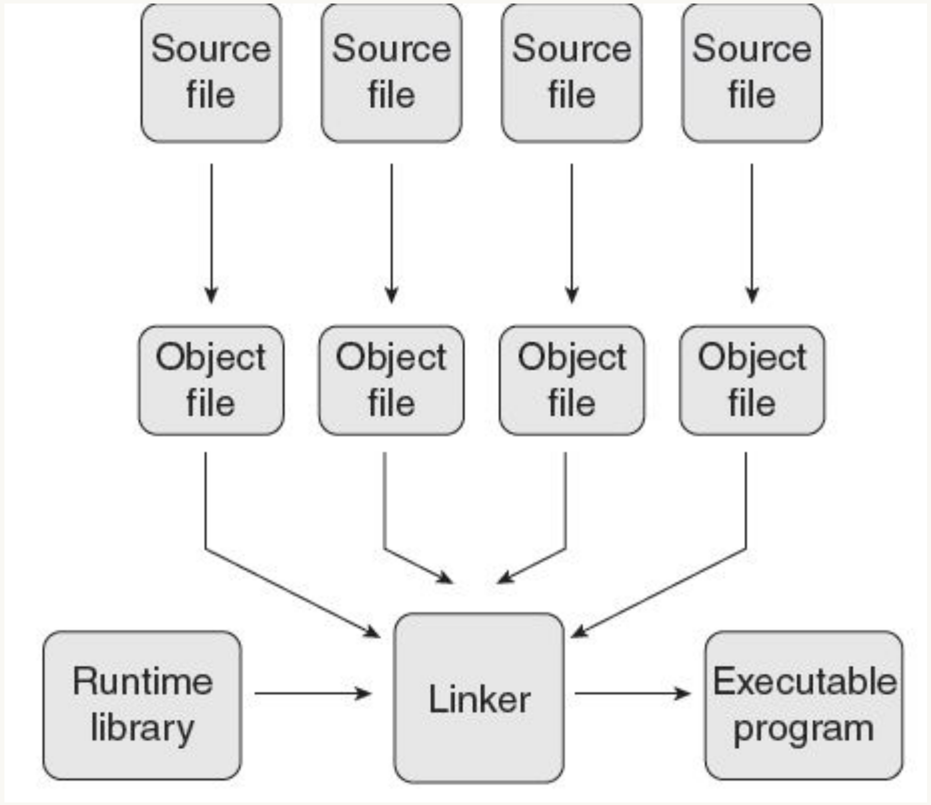


Compiler	Interpreter
• It translates the entire program in one go. • It generates error(s) after translating the entire program. • Execution of code is faster. • An object file is generated. • Code need not be recompiled every time it is executed. • It merely translates the code. • It requires more memory space (to save the object file).	• It interprets and executes one statement at a time. • It stops translation after getting the first error. • Execution of code is slower as every time reinterpretation of statements has to be done. • No object file is generated. • Code has to be reinterpreted every time it is executed. • It translates as well as executes the code. • It requires less memory space (no object file).

Linker

- Program is divided into various(smaller) modules as it is easy to code, edit, debug, test, document, and maintain them. Moreover, a module written for one program can also be used for another program. When a module is compiled, an object file of that module is generated.
- Once the modules are coded and tested, the object files of all the modules are combined together to form the final executable file.
- Therefore, a linker, also called a link editor or binder, is a program that combines the object modules to form an executable program..

Linker contd.



Loader

- A loader is a special type of program that copies programs from a storage device to the main memory, where they can be executed.
- The functionality and complexity of most loaders are hidden from the users.

Fourth Generation: Very High-level Languages

- The instructions of the code are written in English-like sentences.
- They are non-procedural, so users concentrate on the ‘what’ instead of the ‘how’ aspect of the task
- The code written in a 4GL is easy to maintain.
- The code written in a 4GL enhances the productivity of programmers, as they have to type fewer lines of code to get something done. A programmer supposedly becomes 10 times more productive when he/she writes the code using a 4GL than using a 3GL.
- A typical example of a 4GL is the query language, which allows a user to request information from a database with precisely worded English-like sentences.

Fifth Generation Programming Language

- Fifth-generation programming languages (5GLs) are centered on solving problems using the constraints given to a program rather than using an algorithm written by a programmer.
- Most constraint-based and logic programming languages and some declarative languages form a part of the 5GLS.
- These languages are widely used in artificial intelligence research. Another aspect of a 5GL is that it contains visual tools to help develop a program.
- Typical examples of 5GLs include Prolog,OPS5, Mercury, and Visual Basic.

PROGRAMMING PARADIGMS

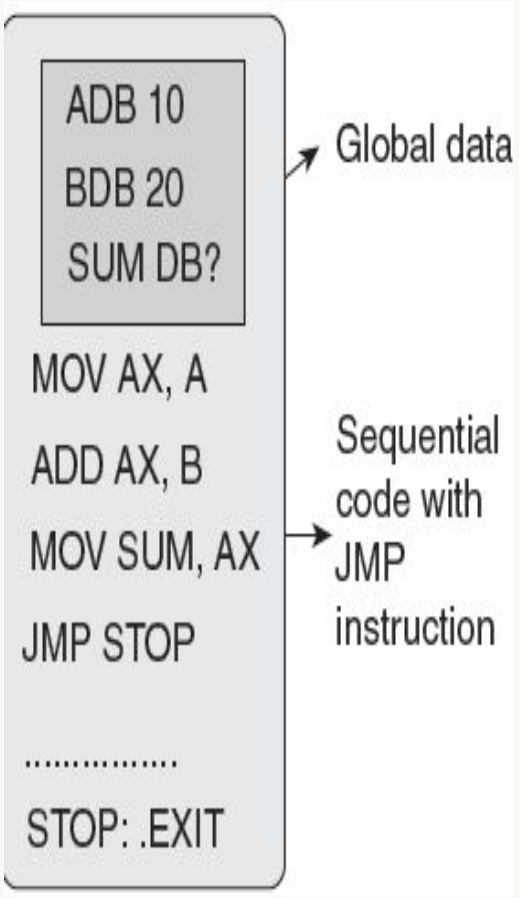
- A programming paradigm is a fundamental style of programming that defines how the structure and basic elements of a computer program will be built. The style of writing programs and the set of capabilities and limitations that a particular programming language has depends on the programming paradigm it supports.
- While some programming languages strictly follow a single paradigm, others may draw concepts from more than one.
- These paradigms, in sequence of their application, can be classified as follows:-
- Monolithic programming—emphasizes on finding a solution

PROGRAMMING PARADIGMS contd.

- Procedural programming—lays stress on algorithms
- Structured programming—focuses on modules
- Object-oriented programming—emphasizes on classes and objects
- Logic-oriented programming—focuses on goals usually expressed in predicate calculus
- Rule-oriented programming—makes use of ‘if-then-else’ rules for computation
- Constraint-oriented programming—utilizes invariant relationships to solve a problem

Monolithic Programming

- Programs written using monolithic programming languages such as assembly language and BASIC consist of global data and sequential code. The global data can be accessed and modified (knowingly or mistakenly) from any part of the program, thereby, posing a serious threat to its integrity.
- A sequential code is one in which all instructions are executed in the specified sequence. In order to change the sequence of instructions, jump statements or 'Go To' statements are used monolithic programs have just one program module as such programming languages do not support the concept of subroutines.



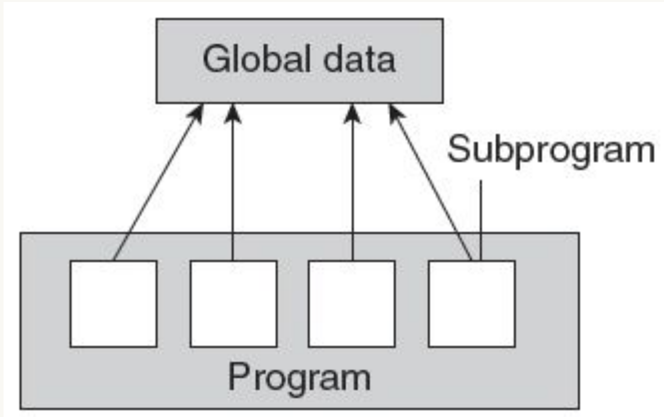
```
ADB 10
BDB 20
SUM DB?
MOV AX, A
ADD AX, B
MOV SUM, AX
JMP STOP

...............

STOP: .EXIT
```

Procedural Programming

- A program is divided into *n number of subroutines* that access global data
- A subroutine that needs the service provided by another subroutine can call that subroutine.

Advantages

- The only goal is to write correct programs.
- Programs were easier to write as compared to monolithic programming.



Procedural Programming contd.

Disadvantages

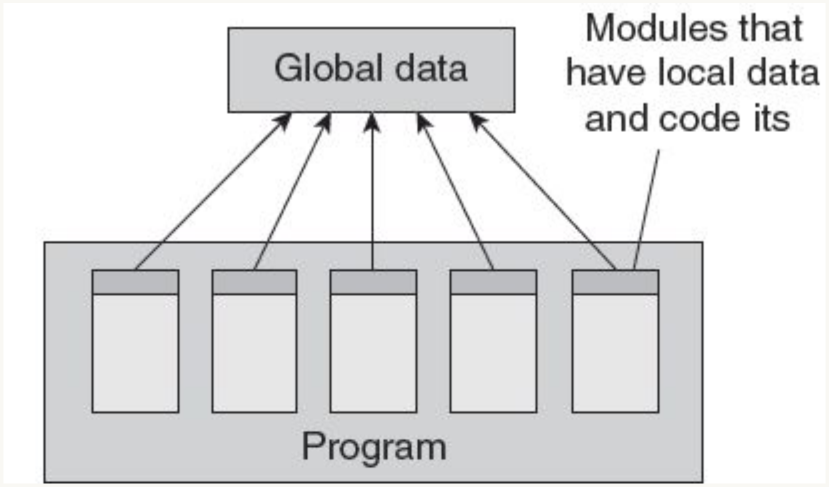
- Writing programs is complex.
- No concept of reusability.
- Requires more time and effort to write programs.
- Programs are difficult to maintain.
- Global data is shared and therefore may get altered (mistakenly).

Structured Programming

- Structured programming, also referred to as modular programming, employs a top-down approach in which the overall program structure is broken down into separate modules.
- This allows the code to be loaded into memory more efficiently and also be reused in other programs. Modules are coded separately and once a module is written and tested individually, it is then integrated with other modules to form the overall program structure.

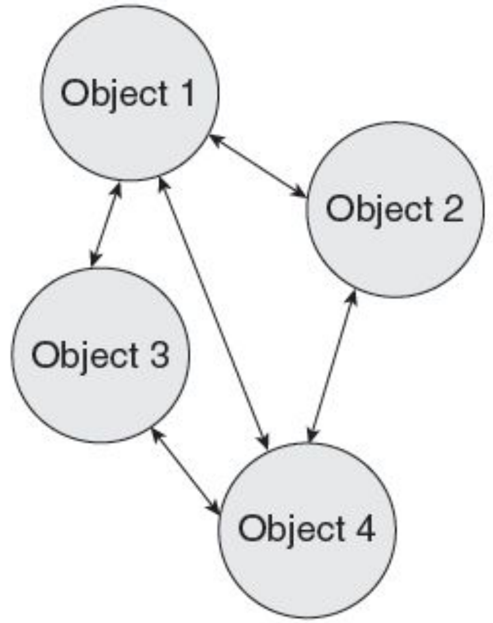
Structured Programming contd.

- Not data-centered
- Global data is shared and therefore may get inadvertently modified
- Main focus on functions



Object Oriented Programming

- It treats data as a critical element in the program development and restricts its flow freely around the system.
- The object oriented paradigm is task based (as it considers operations) as well as data-based (as these operations are grouped with the relevant data).
- Every object contains some data and the operations, methods, or functions that operate on that data. While some objects



Objects of a program interact by sending messages to each other

Figure 1.9 Object oriented paradigm

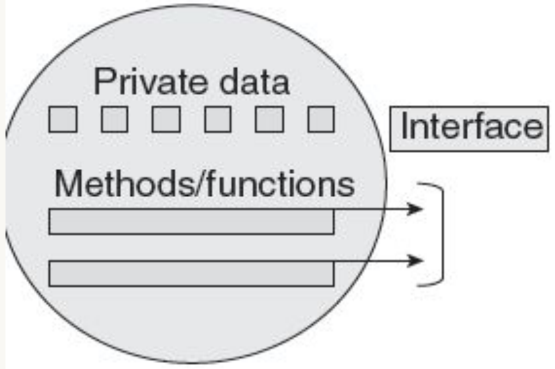


Figure 1.10 Object

Object Oriented Programming contd.

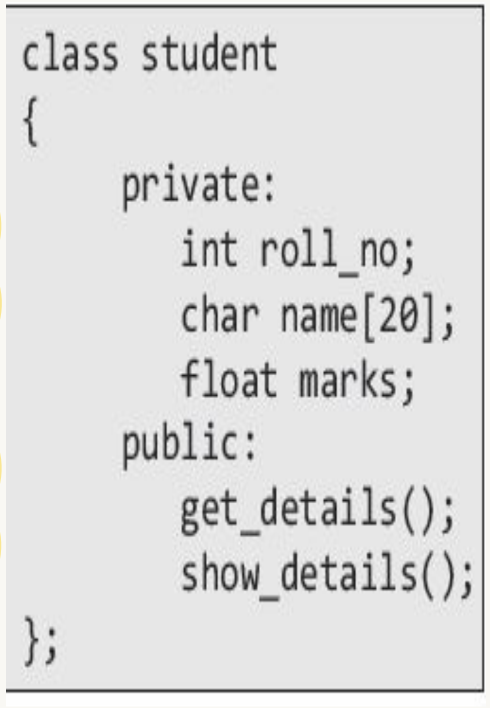
- May contain only basic data types such as characters, integers, floating types, the other object, the other objects on the other hand may incorporate complex data types such as trees or graphs.
- Programs that need the object will access the object’s methods through a specific interface. The interface specifies how to send a message to the object, that is, a request for a certain operation to be performed.
- The programs are data centered.
- Programs are divided in terms of objects and not procedures.
- Functions that operate on data are tied together with the data.

Object Oriented Programming contd.

- Data is hidden and not accessible by external functions.
- New data and functions can be easily added as and when required.
- Follows a bottom-up approach for problem solving.

Classes

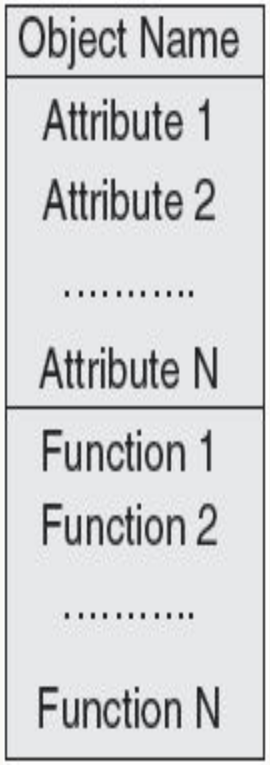
- OOP, being specifically designed to solve real world problems, allows its users to create user defined data types in the form of classes.
- A class provides a template or a blueprint that describes the structure and behaviour of a set of similar objects.
- Once we have the definition for a class, a specific instance of the class can be easily created.



```
class student
{
        private:
            int roll_no;
            char name[20];
            float marks;
        public:
            get_details();
            show_details();
};
```

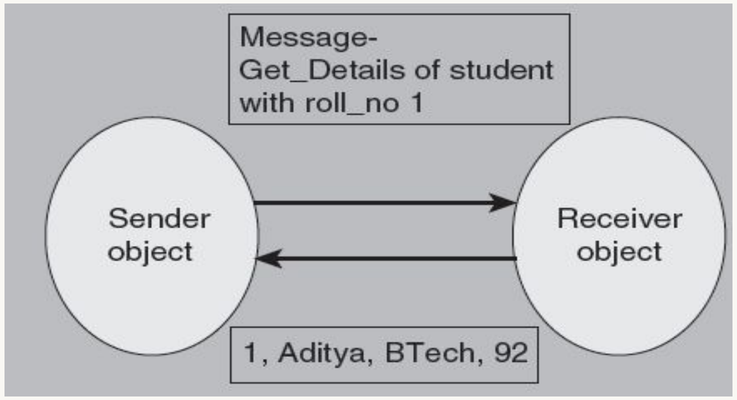
Objects

- We have taken an example of student class and have mentioned that a class is used to create instances, known as objects. Therefore, if student is a class, then all the 60 students
- In a course (assuming there are maximum 60 students in a particular course) are the objects of the student class. Therefore, all students such as Aditya, Chaitanya, Deepti, and Esha are objects of the class.
- Hence, a class can have multiple instances.Every object contains some data and procedures. They are also called methods.



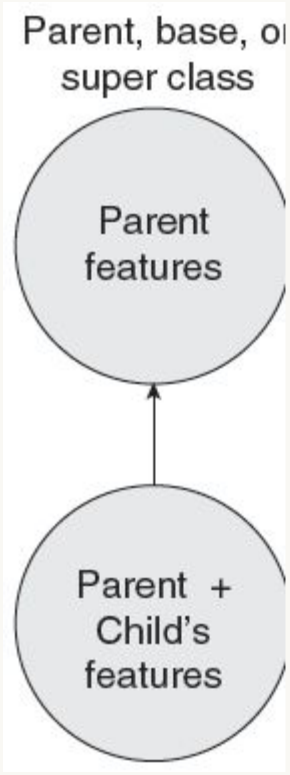
Method and Message Passing

- A method is a function associated with a class. It defines the operations that the object can execute when it receives a message. In object oriented language, only methods of the class can access and manipulate the data stored in an instance of the class (or object).
- Two objects can communicate with each other through messages. An object asks another object to invoke one of its methods by sending it a message.



Inheritance

- Inheritance is a concept of OOP in which a new class is created from an existing class. The new class, often known as a sub-class, contains the attributes and methods of the parent class (the existing class from which the new class is created).
- The new class, known as sub-class or derived class, inherits the attributes and behaviour of the pre-existing class, which is referred to as super-class or parent class (refer to Fig.).
- The inheritance relationship of sub- and super classes generates a hierarchy. Therefore, inheritance relation is also called 'is-a' relation.
- The main advantage of inheritance is the ability to reuse the code.



Polymorphism: Static Binding and Dynamic Binding

- Polymorphism, refers to having several different forms.
- It is a concept that enables the programmers to assign a different meaning or usage to a variable, function, or an object in different contexts.
- Dynamic binding or late binding, also known as run time polymorphism, is a feature that enables programmers to associate a function call with a code at the execution (or run) time.
- For example, if we have a function print_result() in class student, then both the inherited classes, undergraduate and postgraduate students will also have the same function

Polymorphism: Static Binding and Dynamic Binding contd.

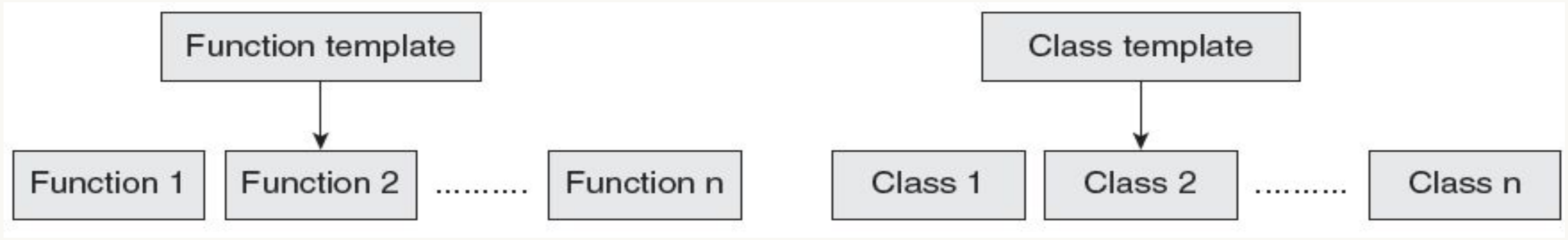
- implemented in their respective classes. Now, when there is a call to print_result(), ptr_to_student-> print_result(); .
- Then, the decision regarding which version to call—the one in undergraduate class or the one in postgraduate class—will be taken at the execution time.

Containership

- The ability of a class to contain object(s) of one or more classes as member data. For example,
- Class One can have an object of class Two as its data member. This would allow the object of class One to call the public functions of class Two .
- Here, class One becomes the container, whereas class Two becomes the contained class.
- Containership is also called composition because as in our example, class One is composed of class Two . In OOP, containership represents a ‘has-a’ relationship

Genericity

- To reduce code duplication and generate short, simpler code, C++ supports the use of generic codes (or templates) to define the same code for multiple data types.
- This means that a C++ function or a class can perform similar operations on different data types like integers, fl oat, and double. This means that a generic function can be invoked with arguments of any compatible type.
- Generic programs, therefore, act as a model of function or class that can be used to generate functions or classes. During program compilation, C++ compiler generates one or more functions or classes based on the specified template.



Delegation

- In delegation, more than one object is involved in handling a request. The object that receives the request for a service, delegates it to another object called its delegate. The property of delegation emphasizes on the ideology that a complex object is made of several simpler objects.
- For example, our body is made up of brain, heart, hands, eyes, ears, etc., the functioning of the whole body as a system rests on correct functioning of the parts it is composed of. Similarly, a car has a wheel, brake, gears, etc. to control it.
- Delegation differs from inheritance in that two classes that participate in inheritance share an 'is-a' relationship; however, in delegate, they have a 'has-a' relationship

Data Abstraction and Encapsulation

- Data abstraction refers to the process by which data and functions are defined in such a way that only essential details are revealed and the implementation details are hidden. The main focus of data abstraction is to separate the interface and the implementation of a program.
- In OOP languages, classes provide public methods to the outside world to provide the functionality of the object or to manipulate the object's data. Any entity outside the world does not know about the implementation details of the class or that method.

Data Abstraction and Encapsulation contd.

- Data encapsulation, also called data hiding, is the technique of packing data and functions into a single component (class) to hide implementation details of a class from the users.
- Users are allowed to execute only a restricted set of operations (class methods) on the data members of the class.
- Therefore, encapsulation organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object

MERITS OF OOP LANGUAGE

- Elimination of redundant code through inheritance (by extending existing classes).
- Higher productivity and reduced development time due to reusability of the existing modules.
- Secure programs as data cannot be modified or accessed by any code outside the class.
- Real world objects in the problem domain can be easily mapped objects in the program.
- A program can be easily divided into parts based on objects.
- The data centered design approach captures more details of a model in a form that can be easily implemented.

MERITS OF OOP LANGUAGE contd.

- Programs designed using OOP are expandable as they can be easily upgraded from small to large systems.
- Message passing between objects simplifies the interface descriptions with external systems. Software complexity becomes easily manageable.
- With polymorphism, behavior of functions, operators, or objects may vary depending upon the circumstances.
- Data abstraction and encapsulation hides implementation details from the external world and provides it a clearly defined interface.
- OOP enables programmers to write easily extendable and maintainable programs.
- OOP supports code reusability to a great extent.

DEMERITS OF OOP LANGUAGE

- Programs written using object oriented languages have greater processing overhead as they demand more resources.
- Requires more skills to learn and implement the concepts.
- Beneficial only for large and complicated programs.
- Even an easy to use software when developed using OOP is hard to be build.
- OOP cannot work with existing systems.
- Programmers must have a good command in software engineering and programming methodology.

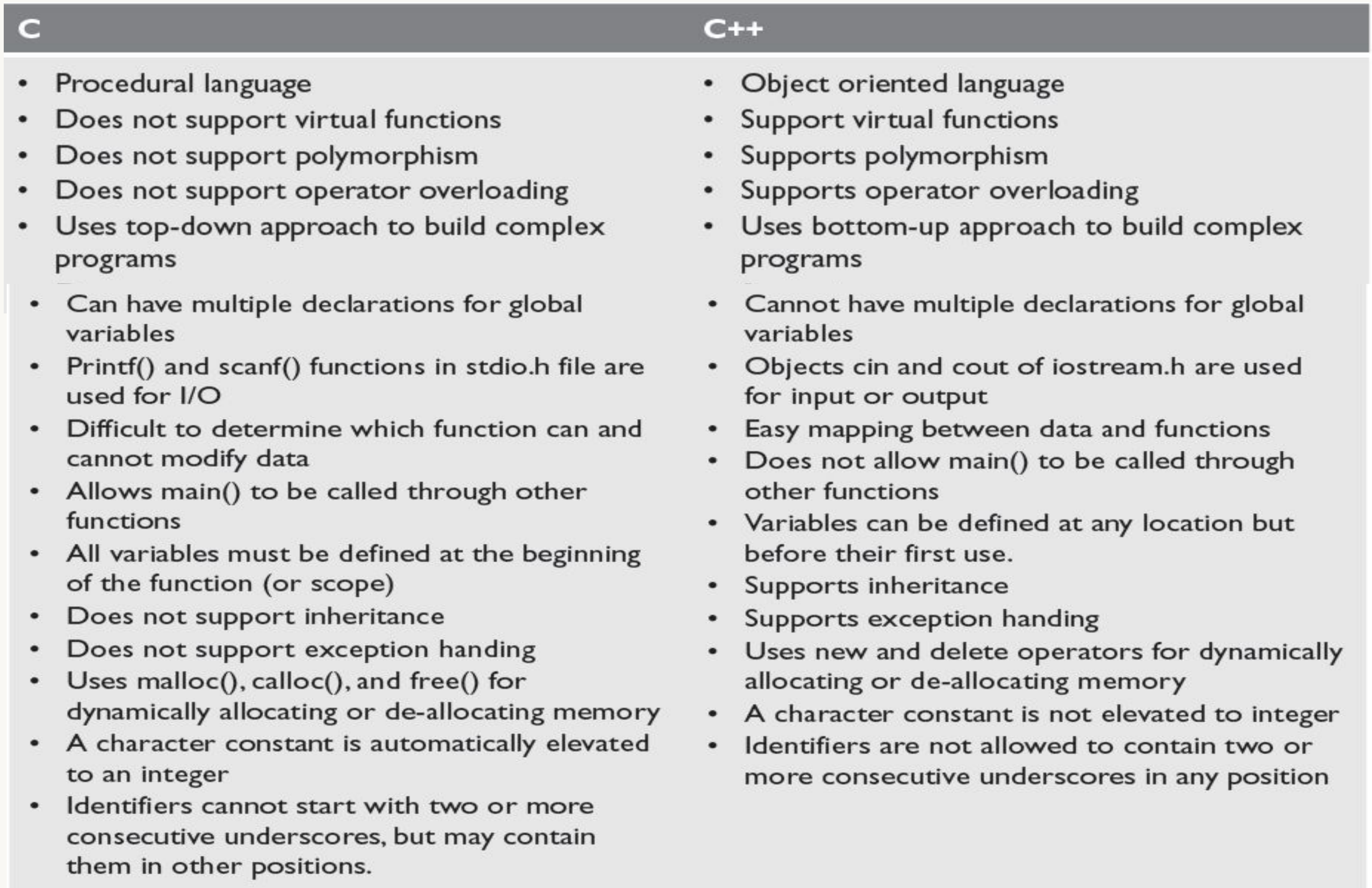
APPLICATIONS OF OOP LANGUAGE

- Designing user interfaces such as work screens, menus, windows, and so on
- Real-time systems
- Simulation and modelling
- Compiler design
- Client server system
- Object oriented databases
- Object oriented distributed database
- Artificial intelligence—expert systems and neural networks

APPLICATIONS OF OOP LANGUAGE

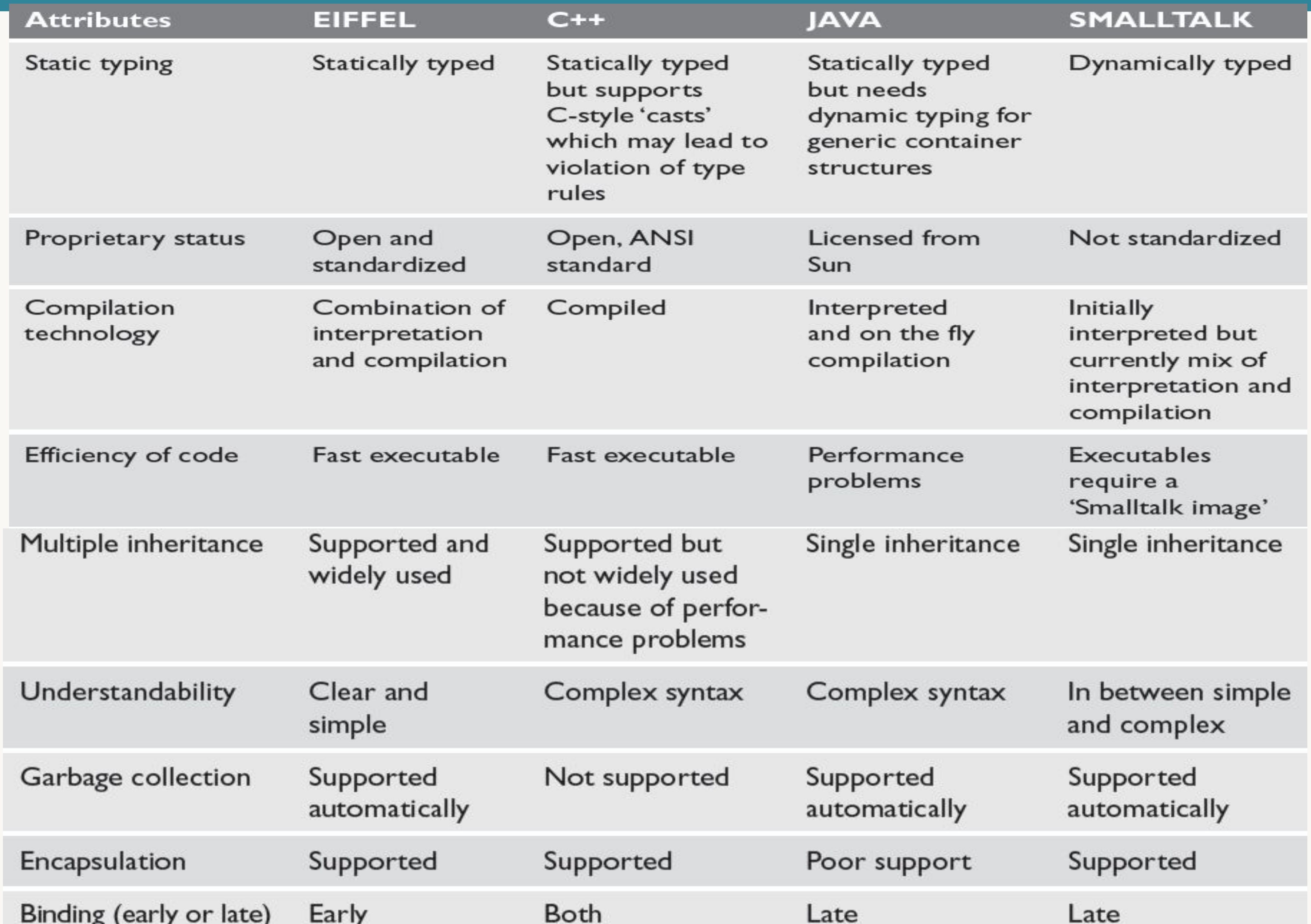
- Parallel programming
- Decision control systems
- Office automation systems
- Networks for programming routers, firewalls, and other devices
- Computer-aided design (CAD) systems
- Computer-aided manufacturing (CAM) systems
- Computer animation
- Developing computer games
- Hypertext and hypermedia

Differences between C and C++



C	C++
• Procedural language	• Object oriented language
• Does not support virtual functions	• Support virtual functions
• Does not support polymorphism	• Supports polymorphism
• Does not support operator overloading	• Supports operator overloading
• Uses top-down approach to build complex programs	• Uses bottom-up approach to build complex programs
• Can have multiple declarations for global variables	• Cannot have multiple declarations for global variables
• Printf() and scanf() functions in stdio.h file are used for I/O	• Objects cin and cout of iostream.h are used for input or output
• Difficult to determine which function can and cannot modify data	• Easy mapping between data and functions
• Allows main() to be called through other functions	• Does not allow main() to be called through other functions
• All variables must be defined at the beginning of the function (or scope)	• Variables can be defined at any location but before their first use.
• Does not support inheritance	• Supports inheritance
• Does not support exception handing	• Supports exception handing
• Uses malloc(), calloc(), and free() for dynamically allocating or de-allocating memory	• Uses new and delete operators for dynamically allocating or de-allocating memory
• A character constant is automatically elevated to an integer	• A character constant is not elevated to integer
• Identifiers cannot start with two or more consecutive underscores, but may contain them in other positions.	• Identifiers are not allowed to contain two or more consecutive underscores in any position

Comparison between commonly used object oriented languages



Attributes	EIFFEL	C++	JAVA	SMALLTALK
Static typing	Statically typed	Statically typed but supports C-style 'casts' which may lead to violation of type rules	Statically typed but needs dynamic typing for generic container structures	Dynamically typed
Proprietary status	Open and standardized	Open, ANSI standard	Licensed from Sun	Not standardized
Compilation technology	Combination of interpretation and compilation	Compiled	Interpreted and on the fly compilation	Initially interpreted but currently mix of interpretation and compilation
Efficiency of code	Fast executable	Fast executable	Performance problems	Executables require a 'Smalltalk image'
Multiple inheritance	Supported and widely used	Supported but not widely used because of perfor-mance problems	Single inheritance	Single inheritance
Understandability	Clear and simple	Complex syntax	Complex syntax	In between simple and complex
Garbage collection	Supported automatically	Not supported	Supported automatically	Supported automatically
Encapsulation	Supported	Supported	Poor support	Supported
Binding (early or late)	Early	Both	Late	Late