**Structured Programming**

**Overview**

**Structured programming** is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines in contrast to using simple tests and jumps such as the go to statement, which can lead to “**spaghetti code**” that is potentially difficult to follow and maintain.

**Discussion**

One of the most important concepts of programming is the ability to control a program so that different lines of code are executed or that some lines of code are executed many times. The mechanisms that allow us to control the flow of execution are called **control structures**. Flowcharting is a method of documenting (charting) the flow (or paths) that a program would execute. There are three main categories of control structures:

* **Sequence** – Very boring. Simply do one instruction then the next and the next. Just do them in a given sequence or in the order listed. Most lines of code are this.
* **Selection** – This is where you select or choose between two or more flows. The choice is decided by asking some sort of question. The answer determines the path (or which lines of code) will be executed.
* **Iteration** – Also known as repetition, it allows some code (one to many lines) to be executed (or repeated) several times. The code might not be executed at all (repeat it zero times), executed a fixed number of times or executed indefinitely until some condition has been met. Also known as looping because the flowcharting shows the flow looping back to repeat the task.

A fourth category describes unstructured code.

* **Branching** – An uncontrolled structure that allows the flow of execution to jump to a different part of the program. This category is rarely used in modular structured programming.

All high-level programming languages have control structures. All languages have the first three categories of control structures (sequence, selection, and iteration). Most have the if then else structure (which belongs to the selection category) and the while structure (which belongs to the iteration category). After these two basic structures, there are usually language variations.

### **Advantages of structured programming**

The primary advantages of structured programming are:

1. It encourages top-down implementation, which improves both readability and maintainability of code.
2. It promotes code reuse, since even internal modules can be extracted and made independent, residents in libraries, described in directories and referenced by many other applications.
3. It's widely agreed that development time and code quality are improved through structured programming.

These advantages are normally seen as compelling, even decisive, and nearly all modern software development employs structured programming.

### **Disadvantages of structured programming**

The biggest disadvantage of structured programming is a reduction in execution efficiency, followed by greater memory usage. Both these problems arise from the introduction of calls to a module or process, which then returns to the caller when it's done. System parameters and system resources are saved on a [stack](https://whatis.techtarget.com/definition/stack) (a queue organized as LIFO, or last-in-first-out) and popped when needed. The more program logic is decomposed, meaning the more modules are involved, the greater the overhead associated with the module interface. All structured programming languages are at risk to "over-structuring" and loss of efficiency.

Structured programming can also be applied incorrectly if the type of structure selected isn't right for the task at hand. The best-known example is the solving of math problems. RPL is an efficient way to state and solve a math problem because it eliminates the need to explicitly state execution order and eliminates recursion in code. However, if that problem were to be posed in structured programming procedural or object form, the resulting code would be much less efficient than the RPL version.

**2.**[**Object-Oriented Programming**](https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/)**:** Object-Oriented Programming, as name suggests, is a different approach to programming that brings together data and functions that execute on them. It basically supports encapsulation, abstraction, inheritance, polymorphism, etc. It also includes data hiding feature therefore it is more secure. This model is based on real life entities that focuses on by whom task is to be done rather than focusing on what to do.

**Example :** [JAVA](https://www.geeksforgeeks.org/java/), [C#](https://www.geeksforgeeks.org/csharp-programming-language/), [C++](https://www.geeksforgeeks.org/c-plus-plus/), etc.

**Difference between Structured Programming and Object-Oriented Programming :**

|  |  |
| --- | --- |
| **Structured Programming** | **Object-Oriented Programming** |
| It is a subset of procedural programming. | It relies on concept of objects that contain data and code. |
| Programs are divided into small programs or functions. | Programs are divided into objects or entities. |
| It is all about facilitating creation of programs with readable code and reusable components. | It is all about creating objects that usually contain both functions and data. |
| Its main aim is to improve and increase quality, clarity, and development time of computer program. | Its main aim is to improve and increase both quality and productivity of system analysis and design. |
| It simply focuses on functions and processes that usually work on data. | It simply focuses on representing both structure and behavior of information system into tiny or small modules that generally combines data and process both. |
| It is a method of organizing, managing and coding programs that can give or provide much easier modification and understanding. | It is a method in which set of objects can vary dynamically and can execute just by acting and reading to each other. |
| In this, methods are written globally and code lines are processed one by one i.e., Run sequentially. | In this, method works dynamically, make calls as per need of code for certain time. |
| It generally follows “Top-Down Approach”. | It generally follows “Bottom-Up Approach”. |
| It provides less flexibility and abstraction as compared to object-oriented programming. | It provides more flexibility and abstraction as compared to structured programming. |
| It is more difficult to modify structured program and reuse code as compared to object-oriented programs. | It is less difficult to modify object-oriented programs and reuse code as compared to structured programs. |
| It gives more importance of code. | It gives more importance to data. |

### Object-Oriented Programming In C++

Object-oriented programming revolves around data. The main programming unit of OOP is the object. An object is a representation of a real-time entity and consists of data and methods or functions that operate on data. This way, data, and functions are closely bound and data security is ensured.

In OOP, everything is represented as an object and when programs are executed, the objects interact with each other by passing messages. An object need not know the implementation details of another object for communicating.

**Apart from objects, OOP supports various features which are listed below:**

* **Classes**
* **Encapsulation**
* **Abstraction**
* **Inheritance**
* **Polymorphism**

Using OOP, we write programs using classes and objects by utilizing the above features. A programming language is said to be a true object-oriented programming language if everything it represents is using an object. Smalltalk is one language which is a pure object-oriented programming language.

On the other hand, programming languages like C++, and Java are said to be partial object-oriented programming languages.

### Why C++ Is Partial OOP?

C++ language was designed with the main intention of using object-oriented features to C language.

Although C++ language supports the features of OOP like Classes, objects, inheritance, encapsulation, abstraction, and polymorphism, there are few reasons because of which C++ is classified as a partial object-oriented programming language.

**We present a few of these reasons below:**

**#1) Creation of class/objects is Optional**

In C++, the main function is mandatory and is always outside the class. Hence, we can have only one main function in the program and can do without classes and objects.

This is the first violation of Pure OOP language where everything is represented as an object.

**#2) Use of Global Variables**

C++ has a concept of global variables that are declared outside the program and can be accessed by any other entity of the program. This violates encapsulation. Though C++ supports encapsulation with respect to classes and objects, it doesn’t take care of it in case of global variables.

**#3) Presence of a Friend Function**

C++ supports a friend class or function that can be used to access private and protected members of other classes by making them a friend. This is yet another feature of C++ that violates OOP paradigm.

To conclude, although C++ supports all the OOP features mentioned above, it also provides features that can act as a workaround for these features, so that we can do without them. This makes C++ a partial Object-oriented programming language.

### OOP Features

**Here we will introduce various OOP features that are used for programming.**

#### **Classes & Objects**

An object is a basic unit in object-oriented programming. An object contains data and methods or functions that operate on that data. Objects take up space in memory.

A class, on the other hand, is a blueprint of the object. Conversely, an object can be defined as an instance of a class. A class contains a skeleton of the object and does not take any space in the memory.

Let us take an **Example** of a car object. A car object named “Maruti” can have data such as color; make, model, speed limit, etc. and functions like accelerate. We define another object “ford”. This can have similar data and functions like that of the previous object plus some more additions.

Similarly, we can have numerous objects of different names having similar data and functions and some minor variations.

Thus instead of defining these similar data and functions in these different objects, we define a blueprint of these objects which is a class called Car. Each of the objects above will be instances of this class car.

#### **Abstraction**

Abstraction is the process of hiding irrelevant information from the user. **For Example**, when we are driving the car, first we start the engine by inserting a key. We are not aware of the process that goes on in the background for starting the engine.

Using abstraction in programming, we can hide unnecessary details from the user. By using abstraction in our application, the end user is not affected even if we change the internal implementation.

#### **Encapsulation**

Encapsulation is the process using which data and the methods or functions operating on them are bundled together. By doing this, data is not easily accessible to the outside world. In OOP we achieve encapsulation by making data members as private and having public functions to access these data members.

#### **Inheritance**

Using inheritance object of one class can inherit or acquire the properties of the object of another class. Inheritance provides reusability of code.

As such we can design a new class by acquiring the properties and functionality of another class and in this process, we need not modify the functionality of the parent class. We only add new functionality to the class.

#### **Polymorphism**

Polymorphism means many forms.

Polymorphism is an important feature of OOP and is usually implemented as operator overloading or function overloading. Operator overloading is a process in which an operator behaves differently in different situations. Similarly, in function overloading, the same function behaves differently in different situations.

#### **Dynamic Binding**

OOP supports dynamic binding in which function call is resolved at runtime. This means that the code to be executed as a result of a function call is decided at runtime. Virtual functions are an example of dynamic binding.

#### **Message Passing**

In OOP, objects communicate with each other using messages. When objects communicate, information is passed back and forth between the objects. A message generally consists of the object name, method name and actual data that is to be sent to another object.

### Advantages Of OOP

**Let us discuss some of the advantages of OOP.**

**#1) Reusability**

OOP allows the existing code to be reused through inheritance. We can easily acquire the existing functionality and improve on it without having to rewrite the code again. This results in less bloated code.

**#2) Modularity**

As we modularize the program in OOP, it’s easy to modify or troubleshoot the program if a problem occurs or new feature or enhancement is to be added. Modularization also helps in code clarity and makes it more readable.

**#3) Flexibility**

OOP helps us with flexible programming using the polymorphism feature. As polymorphism takes many forms, we can have operators or functions that will work with many objects and thus save us from writing different functions for each object.

**#4) Maintainability**

Maintaining code is easier as it is easy to add new classes, objects, etc without much restructuring or changes.

**#5) Data and Information Hiding**

OOP aids us in data hiding thereby keeping information safe from leaking. Only the data that is required for the smooth functioning of the program are exposed to the user by hiding intrinsic details.