1. **What is Object Oriented Programming?**

Object Oriented Programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. There are many object-oriented programming languages including JavaScript, C++, Java, and Python.

1. **What are the differences between Procedure Oriented Programming (POP) & Object Oriented Programming (OOP)?**

|  |  |  |
| --- | --- | --- |
|  | Procedure Oriented Programming | Object Oriented Programming |
| 1. | Program is divided into small parts called functions. | Program is divided into parts called objects. |
| 2. | Importance is not given to data but to functions as well as sequence of actions to be done. | Importance is given to the data rather than procedures or functions because it works as a real world. |
| 3. | Follows Top Down approach. | OOP follows Bottom Up approach. |
| 4. | It does not have any access specifier. | OOP has access specifiers named Public, Private, Protected, etc |
| 5. | Data can move freely from function to function in the system. | Objects can move and communicate with each other through member functions. |
| 6. | To add new data and function in POP is not so easy. | OOP provides an easy way to add new data and function. |
| 7. | Most function uses Global data for sharing that can be accessed freely from function to function in the system. | In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data. |
| 8. | It does not have any proper way for hiding data so it is less secure. | OOP provides Data Hiding so provides more security. |
| 9. | Overloading is not possible. | In OOP, overloading is possible in the form of Function Overloading and Operator Overloading. |
| 10. | Example of Procedure Oriented Programming are : C, VB, FORTRAN, Pascal. | Example of Object Oriented Programming are : C++, JAVA, VB.NET, C#.NET. |

1. **Write down different Principles (or features) of object oriented programming.**

Principles (or features) of object oriented programming:

1. Encapsulation

2. Data abstraction

3. Polymorphism

4. Inheritance

5. Dynamic binding

6. Message passing

**Encapsulation:** Wrapping of data and functions together as a single unit is known as encapsulation. By default data is not accessible to outside world and they are only accessible through the functions which are wrapped in a class. prevention of data direct access by the program is called data hiding or information hiding

**Data abstraction:** Abstraction refers to the act of representing essential features without including the back ground details or explanation. Classes use the concept of abstraction and are defined as a list of attributes such as size, weight, cost and functions to operate on these attributes. They encapsulate all essential properties of the object that are to be created. The attributes are called as data members as they hold data and the functions which operate on these data are called as member functions. Class use the concept of data abstraction so they are called abstract data type (ADT)

**Polymorphism:** Polymorphism comes from the Greek words “poly” and “morphism”. “poly” means many and “morphism” means form i.e.. many forms. Polymorphism means the ability to take more than one form. For example, an operation have different behavior in different instances. The behavior depends upon the type of the data used in the operation. Different ways to achieving polymorphism in C++ program: 1) Function overloading 2) Operator overloading

**Inheritance:** Inheritance is the process by which one object can acquire the properties of another. Inheritance is the most promising concept of OOP, which helps realize the goal of constructing software from reusable parts, rather than hand coding every system from scratch. Inheritance not only supports reuse across systems, but also directly facilitates extensibility within a system. Inheritance coupled with polymorphism and dynamic binding minimizes the amount of existing code to be modified while enhancing a system. When the class child, inherits the class parent, the class child is referred to as derived class (sub class) and the class parent as a base class (super class). In this case, the class child has two parts: a derived part and an incremental part. The derived part is inherited from the class parent. The incremental part is the new code written specifically for the class child.

**Dynamic binding:** Binding refers to linking of procedure call to the code to be executed in response to the call. Dynamic binding(or late binding) means the code associated with a given procedure call in not known until the time of call at run time.

**Message passing:** An object oriented program consists of set of object that communicate with each other. Objects communicates with each other by sending and receiving information. A message for an object is a request for execution of a procedure and therefore invoke the function that is called for an object and generates result

1. **What the benefits are of object oriented programming (OOPs)?**

Benefits of object oriented programming (OOPs):

¬ Reusability: In OOP‟ s programs functions and modules that are written by a user can be reused by other users without any modification.

¬ Inheritance: Through this we can eliminate redundant code and extend the use of existing classes. ¬ Data Hiding: The programmer can hide the data and functions in a class from other classes. It helps the programmer to build the secure programs.

¬ Reduced complexity of a problem: The given problem can be viewed as a collection of different objects. Each object is responsible for a specific task. The problem is solved by interfacing the objects. This technique reduces the complexity of the program design.

¬ Easy to Maintain and Upgrade: OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones. Software complexity can be easily managed.

Message Passing: The technique of message communication between objects makes the interface¬ with external systems easier.

¬ Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.

A **class** is an abstract blueprint used to create more specific, concrete objects. Classes often represent broad categories, like Car or Dog that share **attributes**. These classes define what attributes an instance of this type will have, like color, but not the value of those attributes for a specific object.

Classes can also contain functions, called **methods** available only to objects of that type. These functions are defined within the class and perform some action helpful to that specific type of object.

For example, our Car class may have a method repaint that changes the color attribute of our car. This function is only helpful to objects of type Car, so we declare it within the Car class thus making it a method.

Class templates are used as a blueprint to create individual **objects**. These represent specific examples of the abstract class, like myCar or goldenRetriever. Each object can have unique values to the properties defined in the class.

For example, say we created a class, Car, to contain all the properties a car must have, color, brand, and model. We then create an instance of a Car type object, myCar to represent my specific car.

We could then set the value of the properties defined in the class to describe my car, without affecting other objects or the class template.

We can then reuse this class to represent any number of cars.

to represent any number of cars.

**Objects**

Objects are **instances of classes** created with specific data.

**BASIC STRUCTURE OF C++ LANGUAGE:** The program written in C++ language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. Documentation section

2. Linking section

3. Definition section

4. Global declaration section & class declarations

5. Member function definition

6. Main function section

main()

{

Declaration section

Executable section

}

1. DOCUMENTATION SECTION : comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /\* and \*/ . Whatever is written between these two are called comments.
2. LINKING SECTION : This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.

e.g. #include <iostream>

using namespace std;

¬ directive causes the preprocessor to add the contents of the iostream file to the program. It contains declarations for cout and cin

¬ cout is a predefined object that represents the standard output stream. The operator << is an insertion operator, causes the string in double quotes to be displayed on the screen.

The statement cin>>n; is an input statement and causes the program to wait for the user to type in a number. The number keyed is placed on the variable “n”. The identifier cin is a predefined object in C++ that corresponds to the standard input stream. The operator >> is known as extraction operator. It extracts the value from the keyboard and assigns it to the value variable on its right.

3. DEFINITION SECTION : It is used to declare some constants and assign them some value. e.g. #define MAX 25 Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

4. GLOBAL DECLARATION SECTION : Here the variables and class definations which are used through out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program).

A CLASS is a collection of data and functions that act or manipulate the data. The data components of a class are called data members and function components of a class are called member functions A class can also termed as a blue print or prototype that defines the variable or functions common to all objects of certain kind. It is a user defined data type

1. SUB PROGRAM OR FUNCTION SECTION : This has all the sub programs or the functions which our program needs.

void display()

{

cout<<”C++ is better that C”;

}

**SIMPLE C++ PROGRAM:**

#include<iostream>

using namespace std;

void display(){

cout<<”C++ is better that C”;

}

int main() {

display();

return 0;

}

6. MAIN FUNCTION SECTION : It tells the compiler where to start the execution from main() { point from execution starts } main function has two sections

1. declaration section : In this the variables and their data types are declared.

2. Executable section or instruction section : This has the part of program which actually performs the task we need.

//namespace: namespace is used to define a scope that could hold global identifiers. ex:-namespace scope for c++ standard library. A classes, functions and templates are declared within the namespace named std using namespace std.

**What Is an Object-oriented programming?**

**Object-oriented programming**, or **OOP**, is an approach to problem solving where all computations are carried out using objects. An **object** is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You could also expect a person to be able to do something, such as walking or driving. This would be considered a method of the person.

Code in object-oriented programming is organized around objects. Once you have your objects, they can interact with each other to make something happen. Let's say you want to have a program where a person gets into a car and drives it from A to B. You would start by describing the objects, such as a person and car. That includes methods: a person knows how to drive a car, and a car knows what it is like to be driven. Once you have your objects, you bring them together so the person can get into the car and drive.

**Classes and Objects**

A **class** is a blueprint of an object. You can think of a class as a concept, and the object is the embodiment of that concept. You need to have a class before you can create an object. So, let's say you want to use a person in your program. You want to be able to describe the person and have the person do something. A class called 'person' would provide a blueprint for what a person looks like and what a person can do. To actually use a person in your program, you need to create an object. You use the person class to create an object of the type 'person.' Now you can describe this person and have it do something.

Classes are very useful in programming. Consider the example of where you don't want to use just one person but 100 people. Rather than describing each one in detail from scratch, you can use the same person class to create 100 objects of the type 'person.' You still have to give each one a name and other properties, but the basic structure of what a person looks like is the same.

**Methods and Functions**

Once you have created objects, you want them to be able to do something. This is where methods come in. A **method** in object-oriented programming is a procedure associated with a class. A method defines the behavior of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform. The association between method and class is called binding. Consider the example of an object of the type 'person,' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

A **function** is a combination of instructions that are combined to achieve some result. A function typically requires some input (called arguments) and returns some results. For example, consider the example of driving a car. To determine the mileage, you need to perform a calculation using the distance driven and the amount of fuel used. You could write a function to do this calculation. The arguments going into the function would be distance and fuel consumption, and the result would be mileage. Anytime you want to determine the mileage, you simply call the function to perform the calculation.

How does this differ from a method? A function is independent and not associated with a class. You can use this function anywhere in your code, and you don't need to have an object to use it.

Now, what if you were to associate the function with an object of the type 'car?' For example, you want to be able display the mileage of the car on the dashboard. In this case, the mileage calculation has become a method because it is a procedure associated with the car's class. Every time you create a new object of the type 'car' using the car class, this method will be part of the object. The action the car is now able to perform is to calculate mileage. It is the same calculation as performed by the stand-alone function but is now bound to the car.

**https://beginnersbook.com/2017/08/first-cpp-program/**

**Hello World Program in C++**

/\*

\* Multiple line

\* comment

\*/

#include<iostream>

//Single line comment

using namespace std;

//This is where the execution of program begins

int main()

{

// displays Hello World! on screen

cout<<"Hello World!";

return 0;

}

**Output:**

Hello World!

1. **Comments** – You can see two types of comments in the above program

// This is a single line comment

/\* This is a multiple line comment

\* suitable for long comments

\*/

Comments as the names suggests are just a text written by programmer during code development. Comment doesn’t affect your program logic in any way, you can write whatever you want in comments but it should be related to the code and have some meaning so that when someone else look into your code, the person should understand what you did in the code by just reading your comment.

For example:

/\* This function adds two integer numbers

\* and returns the result as an integer value

\*/

int sum(int num1, int num2) {

return num1+num2;

}

Now if someone reads my comment he or she can understand what I did there just by reading my comment. This improves readability of your code and when you are working on a project with your team mates, this becomes essential aspect.

**2. #include<iostream>** – This statements tells the compiler to include iostream file. This file contains pre defined input/output functions that we can use in our program.

**3. using namespace std;** – A namespace is like a region, where we have functions, variables etc and their scope is limited to that particular region. Here std is a namespace name, this tells the compiler to look into that particular region for all the variables, functions, etc. I will not discuss this in detail here as it may confuse you. I have covered this topic in a separate tutorial with examples. Just follow the tutorial in the given sequence and you would be fine.

**4. int main()** – As the name suggests this is the main function of our program and the execution of program begins with this function, the int here is the return type which indicates to the compiler that this function will return a integer value. That is the main reason we have a return 0 statement at the end of main function.

**5. cout << “Hello World!”;** – The cout object belongs to the iostream file and the purpose of this object is to display the content between double quotes as it is on the screen. This object can also display the value of variables on screen(don’t worry, we will see that in the coming tutorials).

**6. return 0;** – This statement returns value 0 from the main() function which indicates that the execution of main function is successful. The value 1 represents failed execution.

# Variables in C++

A variable is a name which is associated with a value that can be changed. For example when I write int num=20; here variable name is num which is associated with value 20, int is a data type that represents that this variable can hold integer values. We will cover the data types in the next tutorial. In this tutorial, we will discuss about variables.

**Syntax of declaring a variable in C++**

data\_type variable1\_name = value1, variable2\_name = value2;

**For example:**

int num1=20, num2=100;

We can also write it like this:

int num1,num2;

num1=20;

num2=100;

**Types of variables**

Variables can be categorised based on their data type. For example, in the above example we have seen integer types variables. Following are the types of variables available in C++.

**int**: These type of of variables holds integer value.

**char**: holds character value like ‘c’, ‘F’, ‘B’, ‘p’, ‘q’ etc.

**bool**: holds boolean value true or false.

**double**: double-precision floating point value.

**float**: Single-precision floating point value.

## Types of variables based on their scope

Before going further lets discuss what is scope first. When we discussed the [Hello World Program](https://beginnersbook.com/2017/08/first-cpp-program/), we have seen the curly braces in the program like this:

int main {

//Some code

}

Any variable declared inside these curly braces have scope limited within these curly braces, if you declare a variable in main() function and try to use that variable outside main() function then you will get compilation error.

Now that we have understood what is scope. Lets move on to the types of variables based on the scope.

1. Global variable  
2. Local variable

### Global Variable

A variable declared outside of any function (including main as well) is called global variable. Global variables have their scope throughout the program, they can be accessed anywhere in the program, in the main, in the user defined function, anywhere.

Lets take an example to understand it:

#### Global variable example

Here we have a global variable myVar, that is declared outside of main. We have accessed the variable twice in the main() function without any issues.

#include <iostream>

using namespace std;

// This is a global variable

char myVar = 'A';

int main()

{

cout <<"Value of myVar: "<< myVar<<endl;

myVar='Z';

cout <<"Value of myVar: "<< myVar;

return 0;

}

**Output:**

Value of myVar: A

Value of myVar: Z

### Local variable

Local variables are declared inside the braces of any user defined function, main function, loops or any control statements(if, if-else etc) and have their scope limited inside those braces.

#### Local variable example

#include <iostream>

using namespace std;

char myFuncn() {

// This is a local variable

char myVar = 'A';

}

int main()

{

cout <<"Value of myVar: "<< myVar<<endl;

myVar='Z';

cout <<"Value of myVar: "<< myVar;

return 0;

}

**Output:**  
Compile time error, because we are trying to access the variable myVar outside of its scope. The scope of myVar is limited to the body of function myFuncn(), inside those braces.

**Can global and local variable have same name in C++?**

**Lets see an example having same name global and local variable.**

#include <iostream>

using namespace std;

// This is a global variable

char myVar = 'A';

char myFuncn() {

// This is a local variable

char myVar = 'B';

return myVar;

}

int main()

{

cout <<"Funcn call: "<< myFuncn()<<endl;

cout <<"Value of myVar: "<< myVar<<endl;

myVar='Z';

cout <<"Funcn call: "<< myFuncn()<<endl;

cout <<"Value of myVar: "<< myVar<<endl;

return 0;

}

**Output:**

Funcn call: B

Value of myVar: A

Funcn call: B

Value of myVar: Z

**As you can see that when I changed the value of myVar in the main function, it only changed the value of global variable myVar because local variable myVar scope is limited to the function myFuncn().**