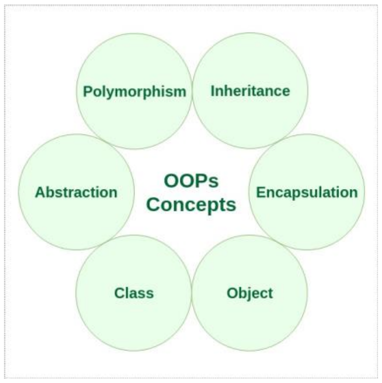
Unit-1
Object-Oriented Programming

Object-oriented programming – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Concepts of an Object Oriented Programming language:



Class : The building block of C++ that leads to Object-Oriented programming is a Class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user-defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.
- In the above example of class Car, the data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.

We can say that a **Class in C++** is a blue-print representing a group of objects which shares some common properties and behaviours.

Object: An Object is an identifiable entity with some characteristics and behaviour. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

```
class person
{
   char name[20];
   int id;
public:
   void getdetails(){}
};
```

```
int main()
{
  person p1; // p1 is a object
}
```

Object take up space in memory and have an associated address like a record in pascal or structure or union in C.
When a program is executed the objects interact by sending messages to one another.
Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

Encapsulation: In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance.
Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is.
Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section".

Encapsulation in C++
Methods
Variables
Class

Encapsulation in C++

Encapsulation also leads to *data abstraction or hiding*. As using encapsulation also hides the data.
In the above example, the data of any of the section like sales, finance or accounts are hidden from any other section.

Abstraction: Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

Polymorphism: The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

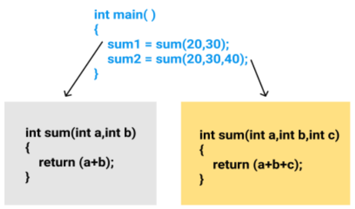
A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.

An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

C++ supports operator overloading and function overloading.

- **Operator Overloading**: The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.
- **Function Overloading*:*** Function overloading is using a single function name to perform different types of tasks.
  Polymorphism is extensively used in implementing inheritance.

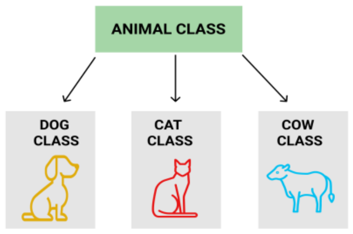
Example: Suppose we have to write a function to add some integers, some times there are 2 integers, some times there are 3 integers. We can write the Addition Method with the same name having different parameters, the concerned method will be called according to parameters.


```
int main( )
{
   sum1 = sum(20,30);
   sum2 = sum(20,30,40);
}
```
```
int sum(int a,int b)
{
   return (a+b);
}
```
```
int sum(int a,int b,int c)
{
   return (a+b+c);
}
```

Inheritance: The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

- **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class**:The class whose properties are inherited by sub class is called Base Class or Super class.
- **Reusability**: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.



Advantages of OOP:

Object Oriented Programming has several advantage -

- Through inheritance redundant code is eliminated and existing class can be extended according to requirement.
- Data hiding can be achieved with the help of the data abstraction and encapsulation .It helps the programmers to build secure programs.
- It is easy to partition the work in a project based on a objects.
- Object oriented system can be easily upgraded from small to large system.
- It provides message passing technique for communication between objects.
- It is helpful in reducing complexity of the programs.
- Object Oriented Programming supports re-usability of the code.

Introduction to C++:

C++ is an object oriented programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey,as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.
C++ is a superset of C, and that virtually any legal C program is a legal C++ program.
It is regarded as a **middle-level language**, as it comprises a combination of both high-level and low-level language features.
C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Applications of C++ :

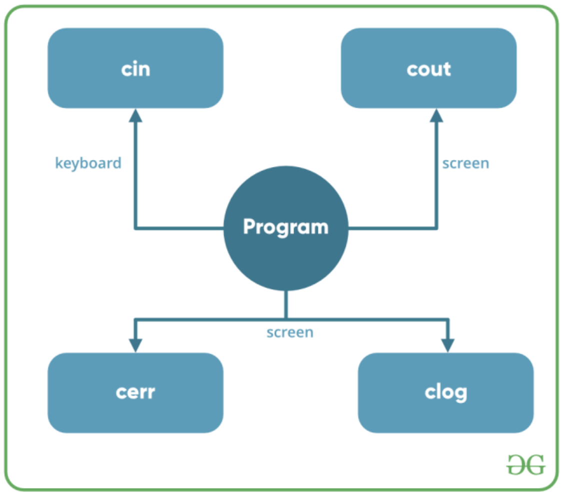
C++ is one of the most widely used programming languages. It has it's presence in almost every area of software development.

- **Application Software Development** - C++ programming has been used in developing almost all the major Operating Systems like Windows, Mac OSX and Linux. Apart from the operating systems, the core part of many browsers like Mozilla Firefox and Chrome have been written using C++. C++ also has been used in developing the most popular database system called MySQL.
- **Programming Languages Development** - C++ has been used extensively in developing new programming languages like C#, Java, JavaScript, Perl, UNIX’s C Shell, PHP and Python, and Verilog etc.
- **Computation Programming** - C++ is the best friends of scientists because of fast speed and computational efficiencies.
- **Games Development** - C++ is extremely fast which allows programmers to do procedural programming for CPU intensive functions and provides greater control over hardware, because of which it has been widely used in development of gaming engines.
- **Embedded System** - C++ is being heavily used in developing Medical and Engineering Applications like softwares for MRI machines, high-end CAD/CAM systems etc.

Input and Output in C++

C++ comes with libraries which provides us with many ways for performing input and output. In C++ input and output is performed in the form of a sequence of bytes or more commonly known as streams.

- **Input Stream:** If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.



Header files available in C++ for Input/Output operations are:

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions to objects like cin, cout, cerr etc.
2. **iomanip**: iomanip stands for input output manipulators. The methods declared in this files are used for manipulating streams. This file contains definitions of setw, setprecision etc.
3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

The two keywords **cout in C++** and **cin in C++** are used very often for printing outputs and taking inputs respectively. These two are the most basic methods of taking input and printing output in C++. To use cin and cout in C++ one must include the header file *iostream* in the program.

We discuss the objects defined in the header file *iostream* like the cin and cout.

I. **Standard output stream (cout)**: Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

```
#include <iostream.h>
int main()
{
    char sample[] = "GeeksforGeeks";

    cout << sample << " - A computer science portal for geeks";

    return 0;
}
```

Output:

```
GeeksforGeeks - A computer science portal for geeks
```

In the above program the insertion operator(<<) inserts the value of the string variable **sample** followed by the string “A computer science portal for geeks” in the standard output stream **cout** which is then displayed on screen.

II. **Standard input stream (cin)**: Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard.
The extraction operator(>>) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keboard.

```
#include <iostream.h>
int main()
{
    int age;

    cout << "Enter your age:";
    cin >> age;
    cout << "\nYour age is: " << age;

    return 0;
}
```

Input :

```
18
```

Output:

```
Enter your age:

Your age is: 18
```

The above program asks the user to input the age. The object cin is connected to the input device. The age entered by the user is extracted from cin using the extraction operator(>>) and the extracted data is then stored in the variable **age** present on the right side of the extraction operator.

III. **Un-buffered standard error stream (cerr)**: The C++ cerr is the standard error stream which is used to output the errors. This is also an instance of the iostream class. As cerr in C++ is un-buffered so it is used when one needs to display the error message immediately. It does not have any buffer to store the error message and display later.

```
#include <iostream.h>
 int main()
{
    cerr << "An error occured";
```

```
   return 0;
}
```
Output:
```
An error occurred
```

IV. **Buffered standard error stream (clog)**: This is also an instance of iostream class and used to display errors but unlike cerr the error is first inserted into a buffer and is stored in the buffer until it is not fully filled. The error message will be displayed on the screen too.

```
#include <iostream.h>
   int main()
{
   clog << "An error occured";

   return 0;
}
```
Output:
```
An error occured
```

Unit-1

Unformatted and Formatted Console I/O

Console I/O opeartion take input from standard input devices like keyboard,compute and give output to standard output device like monitor.
In C++ Programming, the console I/O operations are performed using the header file **iostream.h.** This header file provides two objects **cin** and **cout** to perform input and output operations respectively.
There are mainly two types of consol I/O operations.

- **Unformatted console I/O**
- **Formatted console I/O**

➢ Unformatted Console Input/Output(I/O) operations:

Unformatted console input/output functions are used for performing input/output operations at console and the resulting data is left unformatted and untransformed i.e. it is left in its raw and original form.

In C++, we can read the input entered by a user at console using an object cin of istream class and through this object we can access the functions of istream class, such as - get(char *), get(void) and getline().

In C++, we can write the output at console using an object cout of ostream class and through this object we can access the functions of ostream class, such as - put(), write().
Some of the most important **unformatted console input/output functions** are –

Unformatted Console Input Functions:

There are following unformatted console input function-

Functions	Description
get()	**Reads a *single* character from the user at the console, *and returns it.***
get(char* array)	**Reads a *single* character entered by the user at the console and assigns it to the char array,but needs an Enter key to be pressed at the end..**
getline(char* array, int length)	**Reads a line of characters, entered by the user at the console which ends with a newline character or until the length of .**

Unformatted Console Input Functions

- **Using cin object to read a string from console**

  We can read a string entered by the user using the in-built cin object of *istream* class.

```
// Using cin function to read a string in C++
#include<iostream.h>
int main()
{
char season[20];
cout<<"Enter your favorite season : ";
cin>> season;
cout<<"Your favorite season is : " << season;

return 0;
}
```

Output

```
Enter your favorite season : Spring
Your favorite season is :  Spring
```

- **Using get(void) function to read a char.**

  We can read a character value entered by the user at the console, which gets stored in a **char** variable. The reading ends when user presses **Enter**(*a newline character*)
  Let's see the syntax of **get(void)** function -

```
 get(void);
```

The **get()** function pulls the character entered at the console using *istream* object **cin** and stores into a ***char*** *variable*.

```
// Using get(void) function to read a character in C++
#include<iostream.h>
int main()
{
char c;
cout<<"Please enter a character : ";
c = cin.get();
cout<< "The value in char is : " << c <<"\n";

char d;
cout<<"Please enter a character : ";
d = cin.get();
cout << "The value in char is : "<< d;

char e;
cout<<"Please enter a character : ";
e = cin.get();
cout<< "The value in char is : " << e;
}
```

Output

```
Please enter a character : h
The value of char is : h
Please enter a character : The value in char is :
Please enter a character : p
The value in char is : p
```

- **Using getline(char* array, int length) function to read a string**

  We can read a string entered by the user at the console, which gets stored in a *char array*, using an in-built function getline().

  Let's see the syntax of getline() function –

```
getline (char* arr, int length);
```

The **getline()** function pulls the string entered at the console using *istream* object **cin** and stores into a *c-style* ***char*** *array*, with the maximum *length*.

```
// Using getline() function to read a string in C++
#include<iostream.h>
```

```
int main()
{
char country[20];
cout<<"Enter a country you want to visit : ";
cin.getline(country,20);
cout<<"The country you want to visit is : " << country;

return 0;
}
```

Output

```
Enter a country you want to visit : New Zealand
The country you want to visit is :  New Zealand
```

Unformatted Console Output Functions:

There are following unformatted console output function-

Functions	Description
put(char ch)	**Reads a *single* character from the user at the console, *and returns it.***
write(char *arr, int num)	**Writes a number of characters in a *char array* to the console.**

Unformatted Console Output Functions

- **Using put(char ch) function to display a char.**
  Using the *put(char ch)*, we can display a single character value(*stored in a char variable*) at the console.

  Let's see the syntax of put(char ch) function -

```
put(char ch);
```

The **put(char ch)** function displays a character value at the console using *ostream* object, **cout**.

```
// Using put(char ch) function to display a character in C++
#include<iostream.h>
int main()
{
char ch = 'H';
cout<<"Displaying a char value : ";
cout.put(ch);

cout<<"\nDisplaying another char value : ";
cout.put('X');

}
```

Output

```
Displaying a char value : H
Displaying another char value : X
```

- **Using write(char *arr, int num) to read a string**

  Using the unformatted output function write(), we can display the contents of a *char array* at the console.

  Let's see the syntax of write() function -

```
write(char *arr, int num);
```

The **write()** function pulls the string entered at the console using *istream* object **cin** and stores into a *c-style* ***char*** *array*, with the maximum *length*.

```
// Using write(char *arr, int num) function to read a string in C++
#include<iostream.h>
int main()
{
char arr1[] = "Hello World";
cout<<"Displaying the first 5 characters of char array : ";
cout.write(arr1,5);

cout<<"\nDisplaying the first 10 characters of char array : ";
cout.write(arr1,10);
}
```

Output

```
Displaying the first 5 characters of char array : Hello
Displaying the first 10 characters of char array : Hello Worl
```

➢ Formatted Console I/O operations:

Formatted console input/output functions are used for performing input/output operations at console and the resulting data is formatted and transformed.

In C++, we can *read the input* entered by a user at console using an object **cin** of *istream* class and we can *write the output* at console using an object **cout** of *ostream* class. Through the **cin** and **cout** objects, we can access the formatted I/O functions.

Some of the most important formatted console input/output functions are -

Functions	Description
width(int width)	**Using this function, we can specify the width of a value to be displayed in the output at the console.**
fill(char ch)	**Using this function, we can fill the unused white spaces in a value(*to be printed at the console*), with a character of our choice.**
setf(arg1, arg2)	**Using this function, we can set the *flags*, which allow us to display a value in a particular format.**
unsetf(char ch)	**Using this function, we could clear the flag specified fixed by the function setf().**

precision(int num_of_digts)	**Using this function, we can specify the number of digits(*num_of_digits*) to the right of decimal, to be printed in the output.**

Formatted Console I/O Function

- **Using the output width() function**

  Using this function, we can specify the width of a value to be displayed in the output at the console. Let's see the syntax of **width()** function -

```
width(int width);
```

Where, *width* is the value to be displayed in the output at the console.

Example of *width(int width)* function -

```
//The width() function defines width of the next value to be displayed
//in the output at the console.
#include<iostream.h>
int main()
{
char ch = 'a';

//Adjusting the width of the next value to displayed to 5 columns.
cout.width(5);

cout<<ch <<"\n";


int i = 1;

//width of the next value to be displayed in the output will not be
adjusted to 5 columns.
cout<<i;
}
```

Output

```
    a
1
```

- **Using the fill(), output function**

Using this function, we can fill the unused white spaces in a value(*to be printed at the console*), with a character of our choice
Let's see the syntax of **fill()** function -

```
fill(char ch);
```

Where, *ch* is a char value which fills the unused white spaces in a value.

Example of ***fill(char ch)*** function -

```
//The fill() function fills the unused white spaces in a value(to be
//printed at the console), with a character of our choice.

#include<iostream.h>
int main()
{
char ch = 'a';

//Calling the fill function to fill the white spaces in a value with a
character our of choice.
cout.fill('(');

cout.width(10);
cout<<ch <<"\n";


int i = 1;

//Once you call the fill() function, you don't have to call it again
to fill the white space in a value with the same character.
cout.width(5);
cout<<i;
}
```

Output

```
(((((((((a
((((1
```

- **Using the setf(), output function**
  Using this function, we can set the flags, which allow us to display a value in a particular format.

Let's see the syntax of **setf()** function -

```
setf(arg1, arg2);
```

Where, *arg1* and *arg2* are formatting flags defined in the class ***ios***.

Example of ***setf()*** function –

```
//The setf() function is used to set the flags, which allow us to
//display a value in a particular format.

#include<iostream.h>
int main()
{
int a = 10;

//Calling setf() function to set the flags to display a number, which
is right-justified.
cout.setf(ios::right,ios::adjustfield);

//Calling the width() function to set the number of columns used to
display a value at the console.
cout.width(10);

//Displaying the value of a after calling setf() and width() function
cout<< a << "\n";


int b = 20;

//Calling setf() function to set the flags to display a number, which
is left-justified.
cout.setf(ios::left,ios::adjustfield);

//Calling the width() function to set the number of columns used to
display a value at the console.
cout.width(10);

//Displaying the value of b after calling setf() and width() function
cout<< b;

return 0;
}
```

Output

```
         10
20
```

- **Using the precision(), output function**
  Using this function, we can specify the number of digits to the right of decimal, to be printed in the output at the console.

  Let's see the syntax of precision() function -

```
precision(int num_of_digits);
```

Where, num_of_digits is number of digits to the right of decimal, to be printed in the output at the console.

Note : Before we call precision function, we should call the setf() function to set the flag to display a fixed number of digits after the decimal in a floating-point number by using the flag values ios::fixed and ios::floatfield, otherwise, calling precision() function directly would only fix the total number of digits to display in the output.

Example of precision() function -

```
//The precision() function defines the number of digits to be diplayed after the
decimal, after rounding off a floating number, at the console.
#include<iostream.h>
int main()
{
float d = 12.3458998f;
cout<< "The floating value in variable d : " << d <<"\n";

//Calling setf() function to set the flags to display a fixed number of digits after
decimal
cout.setf(ios::fixed, ios::floatfield);

//Defining the number of digits to be displayed after the decimal in a floating
number.
cout.precision(3);

cout<< "The floating value in variable d, after calling the precision() function : "
<< d;
}
```

Output

```
The floating value in variable d : 12.3459
The floating value in variable d, after calling the precision()
function : 12.346
```

Formatting using Manipulators:

The second way you can alter the format parameters of a stream is through the use of special functions called manipulators that can be included in an I/O expression.
The standard manipulators are shown below:

- **endl:** The endl manipulator of stream manipulators in C++ is used to Output a newline character.
- **hex:** The hex manipulator of stream manipulators in C++ is used to Turns on hex flag
- **setfill(int ch)**: The setfill manipulator of stream manipulators in C++ is used to Set the fill character to ch
- **setprecision(int p):** The setprecision manipulator of stream manipulators in C++ is used to Set the number of digits of precision
- **setw(int w):** The setw manipulator of stream manipulators in C++ is used to Set the field width to w
- **setiosflags(fmtflags f):** The setiosflags manipulator of stream manipulators in C++ is used to Turns on the flag specified in f
- **setprecision(int p):** The setprecision manipulator of stream manipulators in C++ is used to Set the number of digits of precision
- **setw(int w):** The setw manipulator of stream manipulators in C++ is used to Set the field width to w
- **showbase**: The showbase manipulator of stream manipulators in C++ is used to Turns on showbase flag
- **showpoint**: The showpoint manipulator of stream manipulators in C++ is used to Turns on show point flag
- **showpos**: The showpos manipulator of stream manipulators in C++ is used to Turns on showpos flag

To access manipulators that take parameters (such as setw( )), you must include “iomanip” header file in your program.

Example

```
#include <iostream.h>
#include <iomanip.h>
int  main()
{
    // performs ssame as setf( )
    cout << setiosflags(ios::showpos);
    cout << 123<<"\n";

    // hexadecimal base (i.e., radix 16)
    cout << hex << 100 << endl;
    // Set the field width
    cout << setfill('*') << setw(10) << 2343.0;
return 0;
}
```

Output:

```
+123

64

*****+2343
```