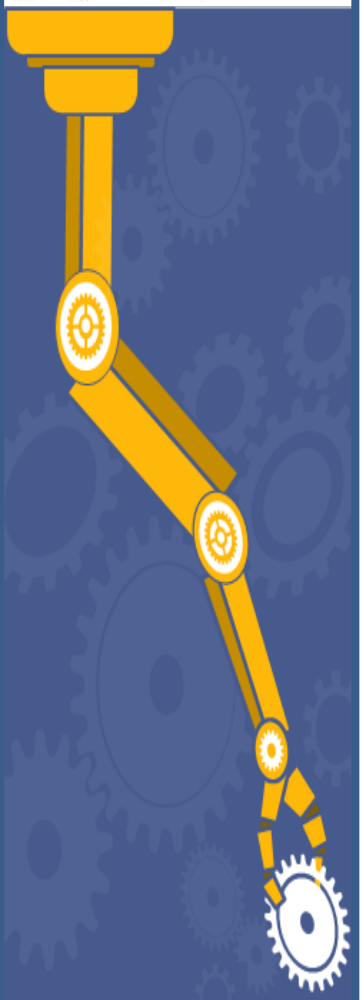
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



BCSE102L- Structured and object-oriented programming

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102P- Structured and object-oriented programming Laboratory

	Indicative Experiments
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

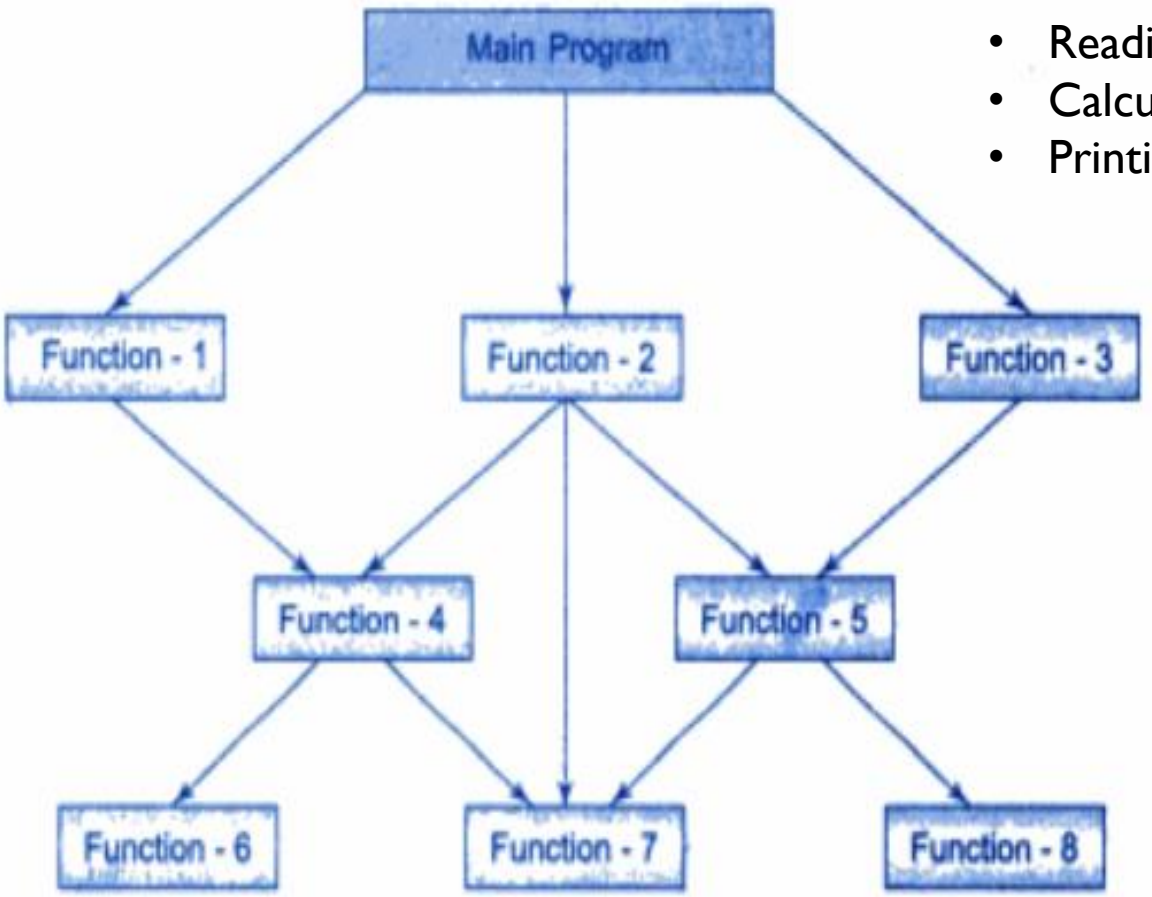
Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-5: Overview of Object Oriented Programming**
  - **Features of OOP- Classes and Objects**
  - **Constructors and Destructors**
  - **Static Data Members and Member Functions**
  - **Inline Functions**
  - **Functions with Default Arguments**
  - **Functions with Objects as Arguments**
  - **Friend Functions and classes**

Revisit to C

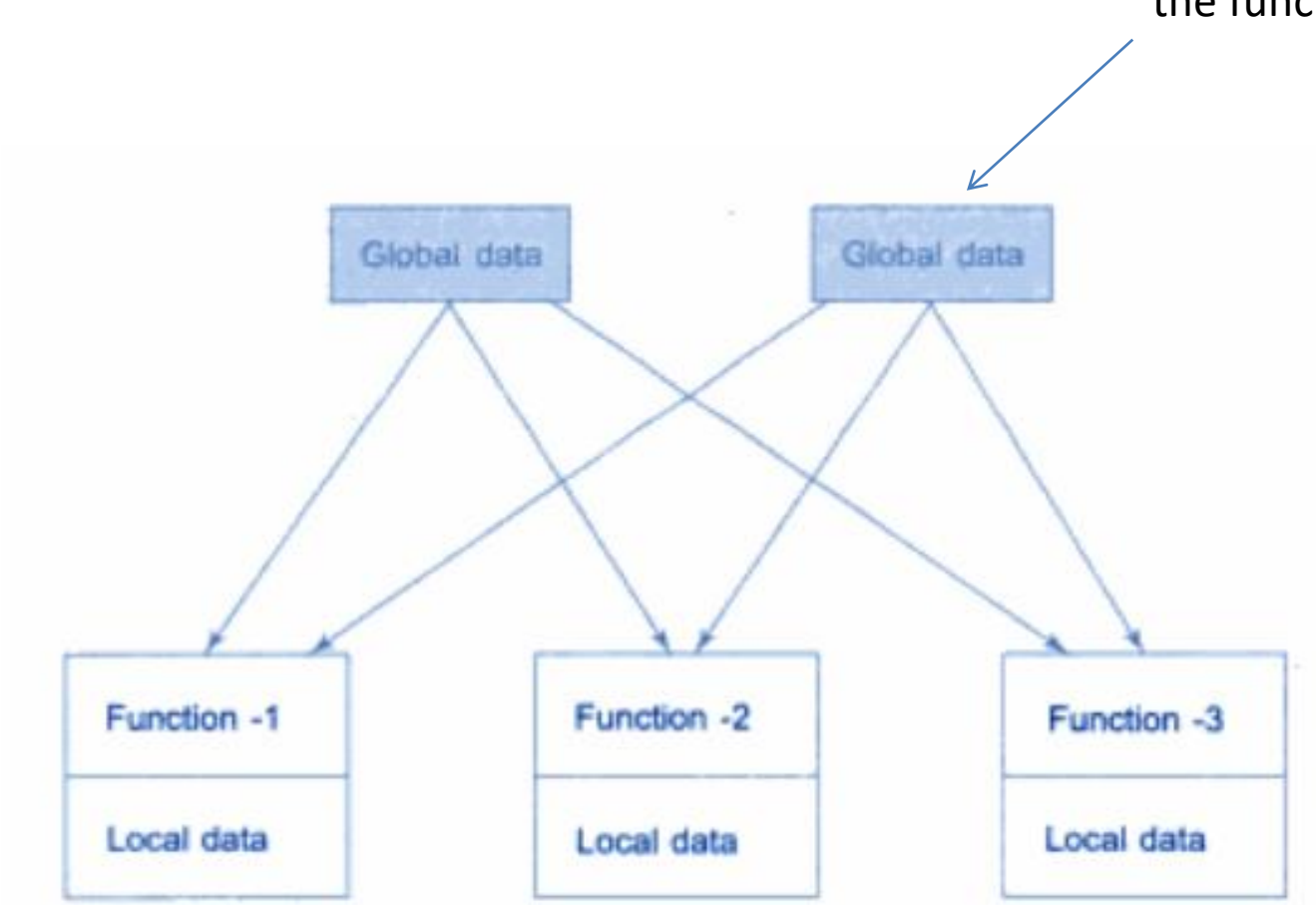
C- Commonly known as Procedure Oriented Approach



- Reading
- Calculating
- Printing

Revisit to C

Relationship among data and functions



Object Oriented Programming

- It is an approach for program organization and development that **attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features** with several new concepts.
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- OOP allows decomposition of problem into a number of entities called **objects** and then builds data and functions around these objects.

Theme of OOP

- View the problem world as a set of objects and communication between the objects.
- Entire world around as can be thought as a set of objects.
- Objects have characteristics and they can perform some functions
- Similar objects may be grouped together.
- For example: **Human Body parts in OOP : Hand**

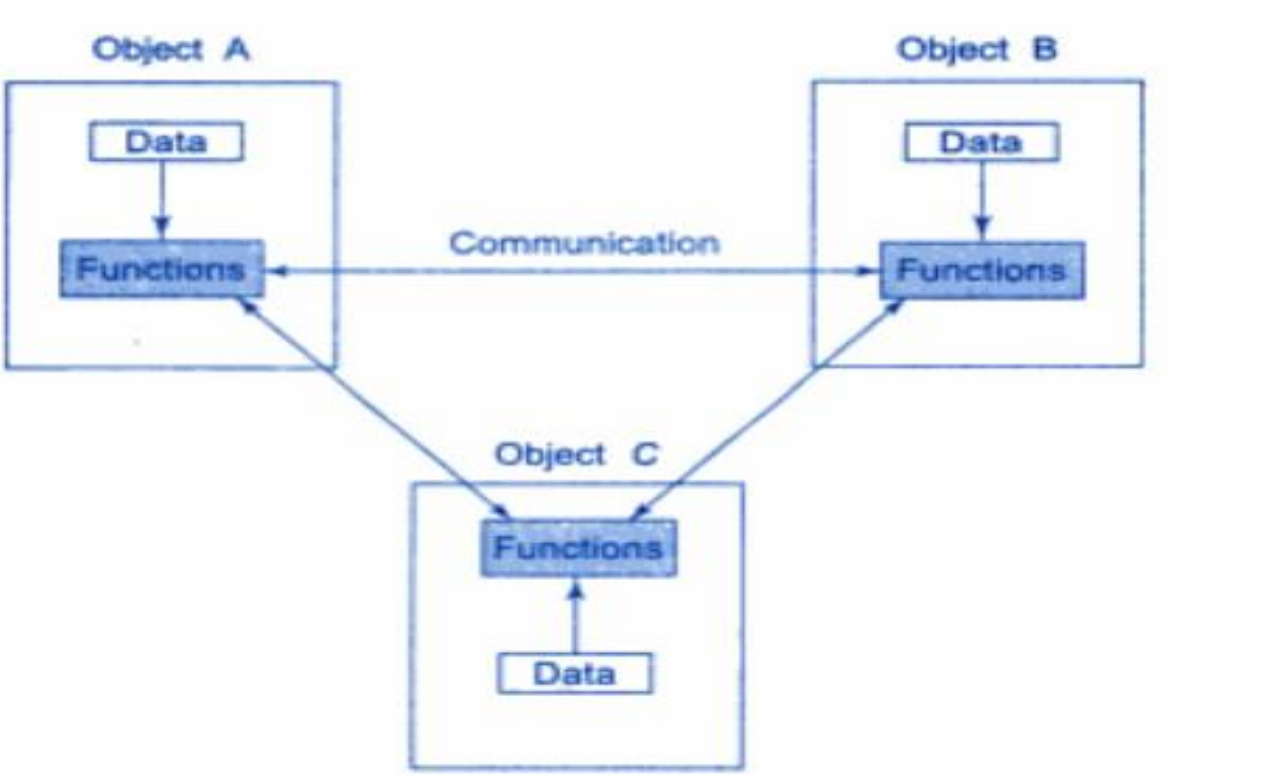
Characteristics/ Properties:

- Number of Bones
- Number of Fingers
- Covered in Skin

Methods/Things it can do:

- Grasping any object
- Count to five with fingers
- Point index finger

Organization of data and functions in OOP



- Different things to different people.
- Data of an object can be accessed only by the functions associated with that object.
- Functions of one object can access the functions of other objects.

Revisit to C Structure

- Consider the following declaration:

```
struct Student
{
        char name[30];
        int roll_number;
        float  total_marks;
} struct student a;  // Declaration in C
```

- "a" is a variable of type student and has three member variables as defined by the template.
- Member variables can be accessed using dot operator like

  a.roll_number

  a.total_marks

- It allows arrays, pointers or structures as members.

Limitations in C Structure

- Standard C does not allow struct data type **to be work like built in types**.

```
struct complex
{
    float  x;
    float  y;
} struct complex c1,c2,c3;  // Declaration in C
```

- c1,c2,c3 can be easily assigned values using dot operator.
- **Cannot able to add two complex numbers or subtract from one another**
- **For Example c3= c1+c2 // Not possible in C**
- Does not Permit " Data Hiding"
- Structure members directly accessed by the structure variables by any function anywhere in their scope. Hence it is called as Public members.

Extension to C Structure

- C++ supports all the features of structures + OOP Concepts.
- C++ - expands capabilities to suit OOP philosophy.
- Tries to bring the user-defined types to built in data types as close as possible.
- Inheritance- One type can inherit characteristics from other types
- In C++, a structure can have both variables and functions as members.
- In C++, the structure names are stand-alone and can be used like any other type names.

  student A; // C++ declaration but error in c.
- C++ incorporates all these extensions in another user-defined type known as **CLASS**

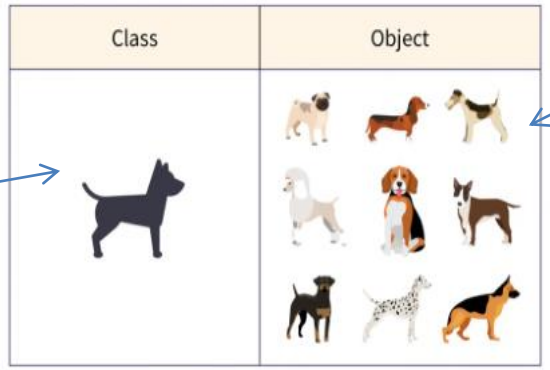
Basic Concepts of OOP

- Concepts extensively used in Object-Oriented Programming.
  - Objects
  - Classes
  - Data Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Dynamic Binding
  - Message Passing

Class

- A class in C++ is a **user-defined type or data structure** declared with a keyword class that has data and functions as its members.
- A class can be used by declaring an object.
- Classes act as a user-defined data type to create objects with similar properties.
- A Class is merely a blueprint of data.
- When a class is created no memory is allocated but when an instance is created by declaring an object, memory is then allocated to store data and perform required functions on them.

Common



Real
Physical

Defining Class in C++

- A Class is defined by a keyword class followed by a class name (user's choice) and a block of curly brackets with semicolons after the block.T
- The block starts with access specifiers followed by data members and member functions.
- Access Specifiers - defines how the members of the class can be accessed.
- **Public:** members can be accessed outside the class.
- **Private:** members cannot be accessed outside the class.
- **Protected:** members cannot be accessed(viewed) from outside the class, but can be accessed in inherited classes(subclasses).

Class- Syntax and Example

SYNTAX:

```
class ClassName
{
    Access specifier:
    Data members;
    Member Functions()
    {
        // member function defintion
    }
};
```

Properties

Also called as methods

EXAMPLE:

```
class Dog{                          // class ClassName
    public:                         //Access specifiers
        string breed, color;        //Data members

        void displayColor(){        //Member functions
            cout<<color<<" ";
        }

        void displayBreed(){
            cout<<breed<<" ";
        }
};                                  // end with semicolon
```

Object

- Objects are basic run-time entities in an object oriented system.
- Note: When a class is defined only the blueprint of data structure is defined as no memory is allocated.
- To use data and its member function we can declare objects. We can declare objects by mentioning the class followed by the user-defined object name.
- Simply it is called as instance of class

- Syntax:

  ClassName ObjectName;

- Example:

  fruit mango; - will create an object mango belonging to the class fruit.

Classes and Objects in Railway Reservation System

Class	Objects
Train	InterCity Express
Coaches/Compartment	S1
Station	Katpadi
Passenger	Any Person Travelling in Train
Employees of Railways	TTR
Ticket	Ticket possessed by a passenger

Data Abstraction

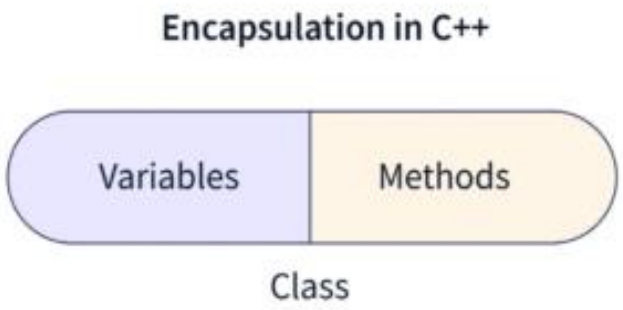
- Providing only essential information to the outside world and hiding their background details.
- In other words, avoiding unnecessary and irrelevant information and only showing the specific details of what users want to see
- By abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.
- In C++, classes uses the concept of data abstraction, they are called as " Abstract Data Types"(ADT)- we use abstract classes and interfaces to achieve this property.

Data Abstraction in Railway Reservation System

Class	Information Hidden
Train	Engine Manufacturing date
Coaches/Compartment	Cleanliness, Color
Station	Length, Shops
Passenger	Number of children
Employees of Railways	Employee id, Salary

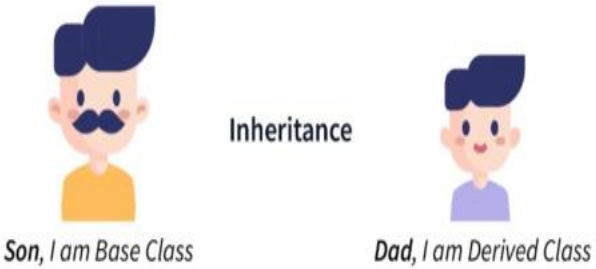
Encapsulation

- Encapsulation means encapsulating (binding or wrapping)code and data together into a single unit.
- Data is not accessible to the outside world, and only those functions which are wrapped inside the class can access it.
- It provides interface between the object's data and the program.



Inheritance

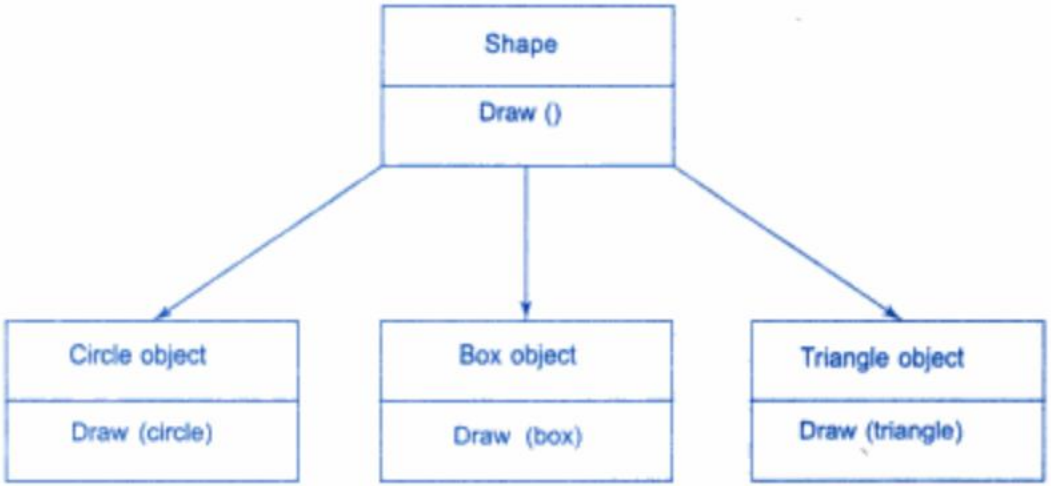
- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- It helps to reduce the code.
- Reusability- we can add additional features to an existing class without modifying it.



- **Base Class :** It is the class whose properties are inherited by another class. It is also called as super class or Parent Class.
- **Derived Class :** It is a class that inherits properties from base class. It is also called as sub class or Child Class

Polymorphism

- Ability to take more than one form, An operation may exhibit different behavior in different instances.
- Behavior depends on the type of the data being used in the operation.



Example:

int a=5, b=7;
c=a+b;
c=12

Example:

a="Hai";
b=" Welcome";
c=a+b;
c= HaiWelcome

Dynamic Binding

- Binding refers to linking of a procedure call to the code to be executed in response to the call.
- Associated with polymorphism and Inheritance.

- For example: **The procedure " draw" in the previous slide, every object will have this method/procedure.**
- Its algorithm is unique with respect to each object and it will be redefined in each class that defines the object.
- At runtime, the code matching the object under current reference will be called.

Message Passing

- An Object-Oriented Program consists of set of objects that communicate with each other.
- The process involves
  - Create class that define objects and their behavior.
  - Create Objects from class definition and
  - Establish communication among objects

– A message for an object is a request for execution of a procedure – will invoke a function(procedure)in the receiving object that generates the desired result.

Message Passing- Example

```
Class shape
{
    int area(int x,int y)
    {
        cout<<"Message passing Value";
        cout<<x<<y;
    }
};

Void main()
{
    shape s1;
    s1.area(5,10);
}
```

Features of OOP

- Emphasis is on data rather than procedure.
- Programs are divided into what is known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by the external functions.
- Objects may communicate with each other through functions.
- New data and functions may easily be added whenever necessary.
- Bottom-up approach is followed in program design.

Difference Between C and C++

C	C++
C is a structural or procedural oriented programming language. Does not support classes and objects.	C++ is an object-oriented programming language, so it is safer and well-structured programming language than C. Supports classes and Objects
C follows the top-down approach	C++ follows the bottom-up approach
Data can be easily manipulated by the outsiders	C++ - Secure Language- Supports Encapsulation and Data Hiding
Does not support the function overloading	C++ supports the function overloading.
Does not support the function overriding.	C++ supports the function overriding.
32 Keywords	52 Keywords

Difference Between C and C++

C	C++
Exception Handling- C does not provide direct support to the exception handling	C++ provides direct support to exception handling by using a try-catch block
scanf and printf functions are used for input and output operations	cin and cout are used for input and output operations
C supports calloc() and malloc() functions for the memory allocation, and free() function for the memory de-allocation	C++ supports a new operator for the memory allocation and delete operator for the memory de-allocation.
C language does not support the inheritance	C++ supports the inheritance.
C program uses **<stdio.h>** header file	C++ program uses **<iostream.h>** header file.

Example C++ Program

```
# include<iostream.h>
Using namespace std;
int main()
{
        float n1, n2;
        int sum, average;
        cin>>n1;
        cin>>n2;
        sum=n1+n2;
        average=sum/2;
        cout<<"Sum= "<<sum<< "\n";
        cout<<"Average= "<<average<< "\n";
        return 0;
}
```

Contains standard input and output functions

Defines the scope for the identifiers.
Std- namespace where standard class libraries are defined.

Without using class

Namespaces in C++

- Consider a situation, when we have two persons with the same name, "x", in the same class.
- **What we do to differentiate them??**
- Same situation in C++ applications too--- writing some code that has a **function called xyz()** and there is another library available which is also having same **function xyz().**

- A **namespace** is designed to overcome this difficulty and is used as additional information to **differentiate similar functions, classes, variables etc. with the same name available in different libraries**. Using namespace, you can define the context in which names are defined.(*Simply it defines the Scope*)

Structure of C++ Program


```
# include<iostream.h>
Using namespace std;

Class person
{
  char name[20];
  int age;
  public:
            void getdata();
            void display();
};
Void person:: getdata()
{
            cin>>name;
            cin>>age;
}
Void person:: display()
{
            cout<<name;
            cout<<age;
}
int main()
{
            person p;
            p.getdata();
            p.display();
            return 0;
}
```

Note: Based on the concept of Client-Server Model

Commonly used Header files

<assert.h>	Contains macros and information for adding diagnostics that aid program debugging	<cassert>
<ctype.h>	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.	<cctype>
<float.h>	Contains the floating-point size limits of the system.	<cfloat>
<limits.h>	Contains the integral size limits of the system.	<climits>
<math.h>	Contains function prototypes for math library functions.	<cmath>
<stdio.h>	Contains function prototypes for the standard input/output library functions and information used by them.	<cstdio>
<stdlib.h>	Contains function prototypes for conversion of numbers to text, text to numbers, memory allocation, random numbers, and various other utility functions.	<cstdlib>
<string.h>	Contains function prototypes for C-style string processing functions.	<cstring>
<time.h>	Contains function prototypes and types for manipulating the time and date.	
<iostream.h>	Contains function prototypes for the standard input and standard output functions.	<iostream>
<iomanip.h>	Contains function prototypes for the stream manipulators that enable formatting of streams of data.	<iomanip>
<fstream.h>	Contains function prototypes for functions that perform input from files on disk and output to files on disk.	<fstream>

Commonly used Header files

Header file	Contents and purpose
<utility>	Contains classes and functions that are used by many standard library header files.
<vector>, <list>, <deque> <queue>, <set>, <map>, <stack>, <bitset>	The header files contain classes that implement the standard library containers. Containers store data during a program's execution. We discuss these header files in Chapter 14.
<functional>	Contains classes and functions used by algorithms of the standard library.
<memory>	Contains classes and functions used by the standard library to allocate memory to the standard library containers.
<iterator>	Contains classes for manipulating data in the standard library containers.
<algorithm>	Contains functions for manipulating data in the standard library containers.
<exception>, <stdexcept>	These header files contain classes that are used for exception handling.
<string>	Contains the definition of class string from the standard library. Discussed in Chapter 15
<sstream>	Contains function prototypes for functions that perform input from strings in memory and output to strings in memory.
<locale>	Contains classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.)
<limits>	Contains a class for defining the numerical data type limits on each computer platform.
<typeinfo>	Contains classes for run-time type identification (determining data types at execution time).

C++ Keywords

asm	**double**	new	**switch**
auto	**else**	operator	template
break	**enum**	private	this
case	**extern**	protected	throw
catch	**float**	public	try
char	**for**	**register**	**typedef**
class	friend	**return**	**union**
const	**goto**	**short**	**unsigned**
continue	**if**	**signed**	virtual
default	inline	**sizeof**	**void**
delete	**int**	**static**	**volatile**
do	**long**	**struct**	**while**
Added by ANSI C++			
bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar_t
explicit	namespace	typeid	

Recall

- Basic Data types
- Derived Data Types
- Size and range of C++ data types
- User Defined Data Types
- Type Compatibility
- Declaration and Initialization of variables
- Operators
- Expressions
- Control structures
- C++ functions
- C++ Arrays
- C++ pointers

https://www.javatpoint.com/cpp-tutorial
https://www.geeksforgeeks.org/introduction-to-c-programming-language/?ref=lbp