NATIONAL INSTITUTE OF TECHNOLOGY

Department of

Electronics and Communication Engineering

Subject Name: Computer Programming

Subject Code: CS10I010CS

Unit: I Introduction to C++and Object Oriented Problem

Presented by:

Dr. Dipen Bepari

Unit-I

- Introduction to Objects and Classes
- Encapsulation
- Access Modifiers
- Polymorphism
- Overloading
- Inheritance
- Overriding Methods
- C++Environment

Introduction

- C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C programming language and originally named C with Classes.
- It was renamed as C++ in 1983.
- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- C++ is a **middle-level** language, as it comprises ***a combination of both high-level and low-level language features***.
- C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

*A programming language is said to use static typing when **type checking is performed during compile-time** as opposed to run-time.*

Why to Learn C++

C++ is a MUST for students and working professionals to become a great Software Engineer. Some of the key advantages of learning C++

- ➢ C++ is very close to hardware, so you get a chance to work at a low level which gives you lot of control in terms of memory management, better performance and finally a robust software development.
- ➢ **C++ programming** gives you a clear understanding about Object Oriented Programming. You will understand low level implementation of polymorphism when you will implement virtual tables and virtual table pointers, or dynamic type identification.
- ➢ C++ is one of the every green programming languages and loved by millions of software developers. If you are a great C++ programmer then you will never sit without work.
- ➢ C++ is the ***most widely used programming languages in application*** and system programming. So you can choose your area of interest of software development.
- ➢ C++ really teaches you the difference between compiler, linker and loader, different data types, storage classes, variable types their scopes etc.

https://www.tutorialspoint.com/compile_cpp_online.php

Applications of C++ Programming

As mentioned before, C++ is one of the most widely used programming languages. It has it's presence in almost every area of software development. I'm going to list few of them here:

•**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.

C++ Basic Syntax

C++ program, it can be defined as a ***collection of objects that communicate*** via invoking each other's methods

- **Object** – Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.
- **Class** – A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

C++ Program Structure

Let us look at a simple code that would print the words *Hello World*.

```
#include <iostream> using namespace std;
 // main() is where program execution begins.
int main() {
cout << "Hello World"; // prints Hello World
return 0;
}
```

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.
- The next line '**// main() is where program execution begins.**' is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "Hello World";** causes the message "Hello World" to be displayed on the screen.
- The next line **return 0;** terminates main( )function and causes it to return the value 0 to the calling process.

The semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

A block is a set of logically connected statements that are surrounded by opening and closing braces. ({....})

C++ ***does not recognize the end of the line as a terminator***. For this reason, it does not matter where you put a statement in a line. For example –
X=2+y ;
Z=(X—3) ;
Can be written as- x=2+y; Z= (X—3);

The reserved ***keywords*** in C++. may not be used as constant or variable or any other identifier names.

C++ Keywords

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

C^{++} Data Types

- ✓ While writing program in any language, it is required to use various variables to store various information.
- ✓ Variables reserves memory locations to store values.
- ✓ Based on the data type of a variable, the operating system ***allocates memory*** and decides **what can be stored** in the reserved memory.

Primitive Built-in Types

Following table lists down seven basic C++ data types –

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void

Several of the basic types can be modified using one or more of these type modifiers –

- signed
- unsigned
- short
- long

The table shows the variable type, ***how much memory it takes to store the value in memory***, and what is ***maximum and minimum value*** which can be stored in such type of variables.

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-2,147,483,648 to 2,147,483,647
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 4,294,967,295
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	
wchar_t	2 or 4 bytes	1 wide character

https://www.tutorialspoint.com/compile_cpp_online.php

Reading and Writing Data

- “cin” nad “cout” are two predefined objects used to Read and write data.
- “cin” and “cout” are defined in the ‘iostream.h’ header file.

The cin Object

- ✓ **E.g. ‘cin>>a’ reads the content of variable ‘a’, >> is called extraction operator.**
- ✓ **cin object is used to read string also.**
- ✓ **Input is stored in one or more variables**
- ✓ **cin converts data to the type that matches the variable:**

```
int height;
cout << "How tall is the room? ";
cin >> height;
```

The cout Object

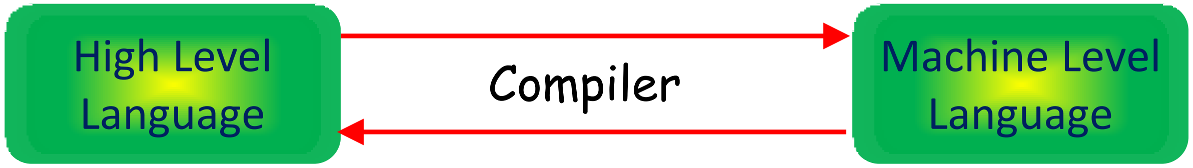
- ✓ **E.g. ‘cout<<a’ displays the content of variable ‘a’, << is called insertion operator.**
- ✓ **If variable is string then *cout* can be used to display the contents of string.**
- ✓ **The cout object Can be used to send more than one item to cout:**
  cout << "Hello " << "there!"; Or cout << "Hello "; cout << "there!";
- ✓ ***char myName[21]; MyName is name of array, 21 is the number of characters that can be stored and cin >> myName; assigns a value***

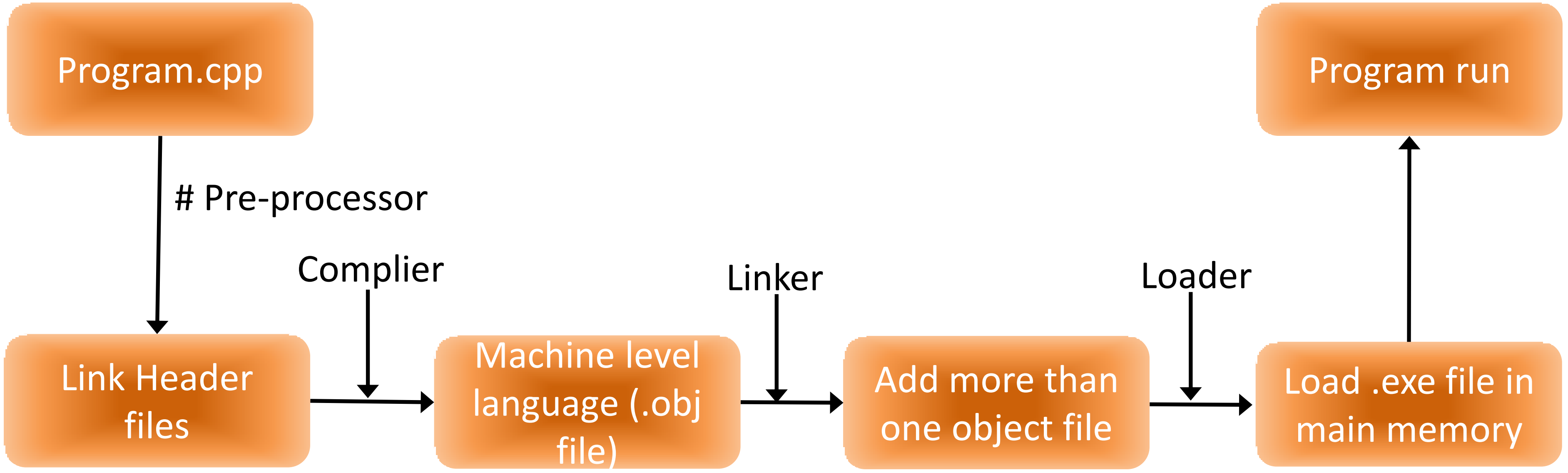
Program Analysis

"std" is a namespace whose members are used in the program.
the members of the "std" namespace are cout, cin, endl, etc.
This namespace is present in the iostream.h header file.
# ==> Pre-processor directory
.h ==> Header file
getch () – to hold the output,
clrscr- clear the screen

C++ Software

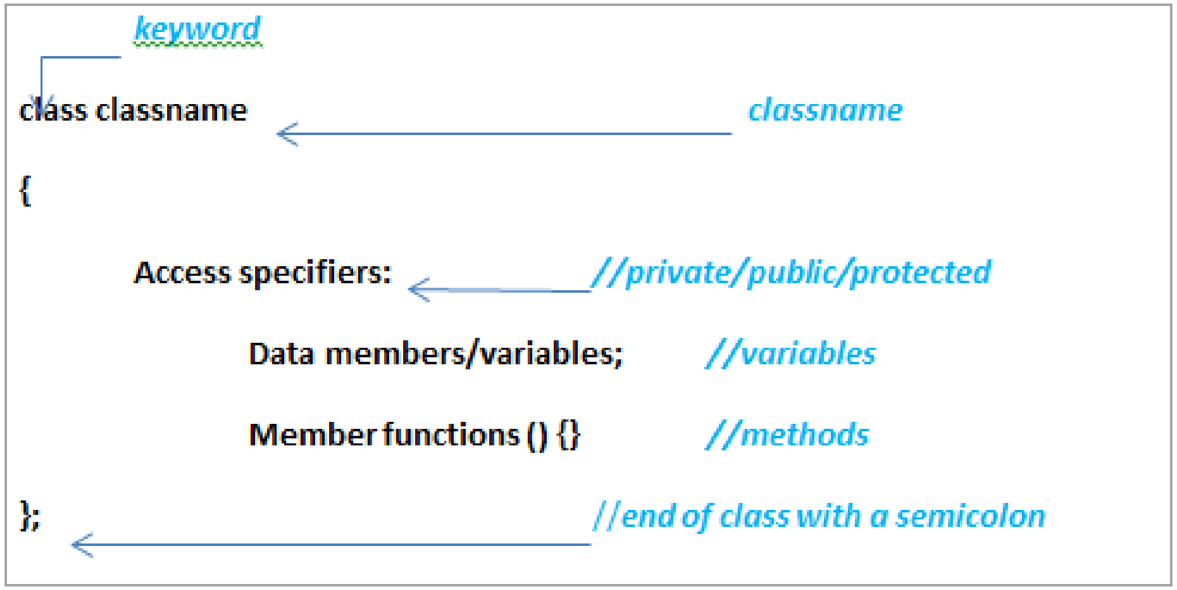
- TurboC++
- DevC++
- Code Block




Program.cpp
# Pre-processor
Link Header files
Complier
Machine level language (.obj file)
Linker
Add more than one object file
Loader
Load .exe file in main memory
Program run

Class and Object

- A class is a ***user-defined data type*** that combines ***data representation and methods for manipulating*** that data into one neat package.
- The data and functions within a class are called ***members of the class***.
- A class in C++ is the building block, that leads to ***Object-Oriented programming***. It is a user-defined data type, which holds its own ***data members and member functions*** that can be accessed and used by creating an instance of that class.
- A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. The syntax of class shown below --



```
        keyword
class classname                         classname
{
      Access specifiers:          //private/public/protected
            Data members/variables;      //variables
            Member functions () {}       //methods
};                                //end of class with a semicolon
```

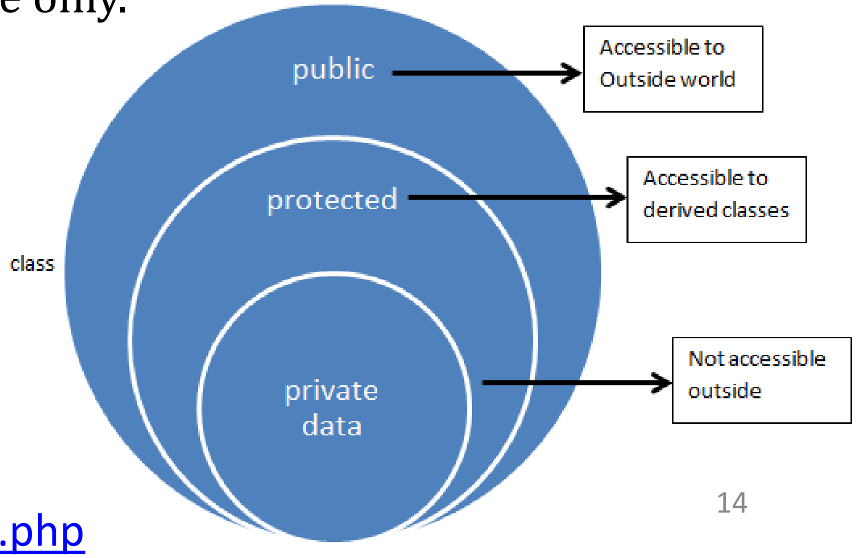
- The keyword **public** determines the access attributes of the members of the class that follows it.
- A ***public*** member can be accessed from outside the class anywhere within the scope of the class object.
- You can also specify the members of a class as **private** or **protected.**

Access Specifiers

Public: Accessible ***from both inside and outside the class*** also i.e by the functions declared in the main() program also.

Private: Only ***members of that class have accessibility***
Can be accessed ***only through member functions of that class*** i.e by the functions declared inside the class only.
Private members and methods are for internal use only.

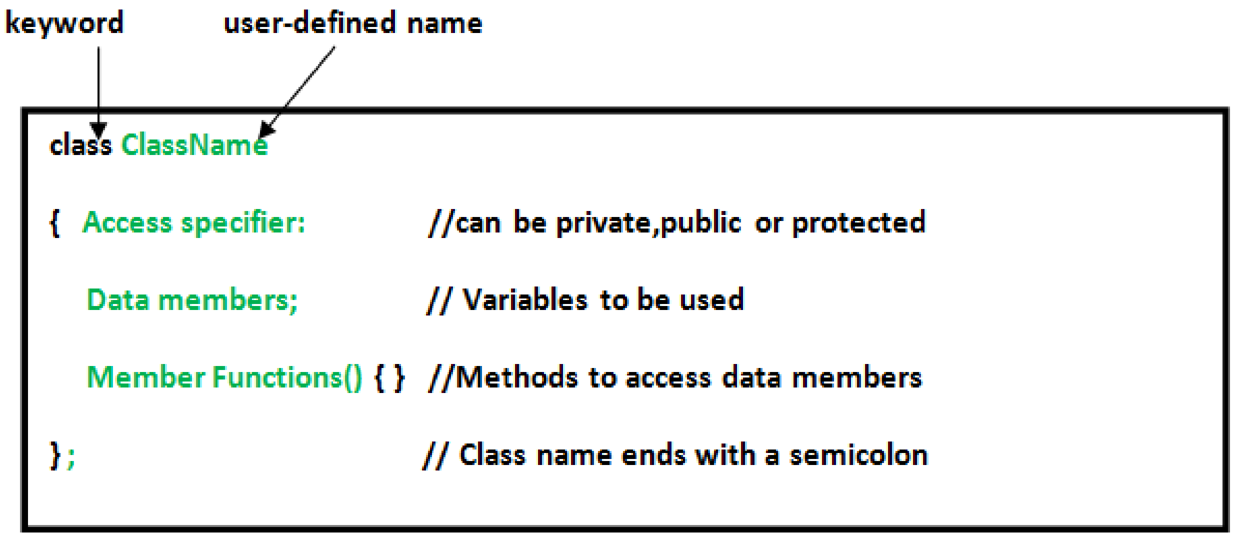
Protected***: Stage between private and public access.***
They can be accessed by the ***member function or friend functions of the class***.
They are similar to private members in the sense that they ***cannot be accessed by the non- member functions of the class***.



https://www.tutorialspoint.com/compile_cpp_online.php

Various Terms of Class

- ***Class Data Member***- ***The variable mentioned inside the class is called class data member.***
- ***Class Member Functions***-- A member function of a class is a ***function which is defined inside the class***.
- ***Class Access Modifiers***-- A class member can be defined as ***public, private or protected***. By default members would be assumed as private.
- ***Constructor & Destructor***-- A class constructor is a special function in a class that is called ***when a new object of the class is created.*** A destructor is also a special function which is called ***when created object is deleted.***



```
class ClassName
{  Access specifier:            //can be private,public or protected
   Data members;                // Variables to be used
   Member Functions() { }       //Methods to access data members
};                              // Class name ends with a semicolon
```

- ***Instance of class--*** – Data member and member function can be used only when we create "Instance of class". Instance of class indicates ***object of class.***
- ***Copy Constructor***—It is a constructor which ***creates an object*** by initializing it with an object of the same class, which has been created previously.
- ***Friend Functions***-- A **friend** function is ***permitted full access to private and protected members of a class.***
- ***This Pointer***-- Every object has a special pointer **'this'** which points to the object itself.
- ***Static Members of a Class***--- Both ***data members and function members*** of a class can be declared as static.

Define C++ Objects

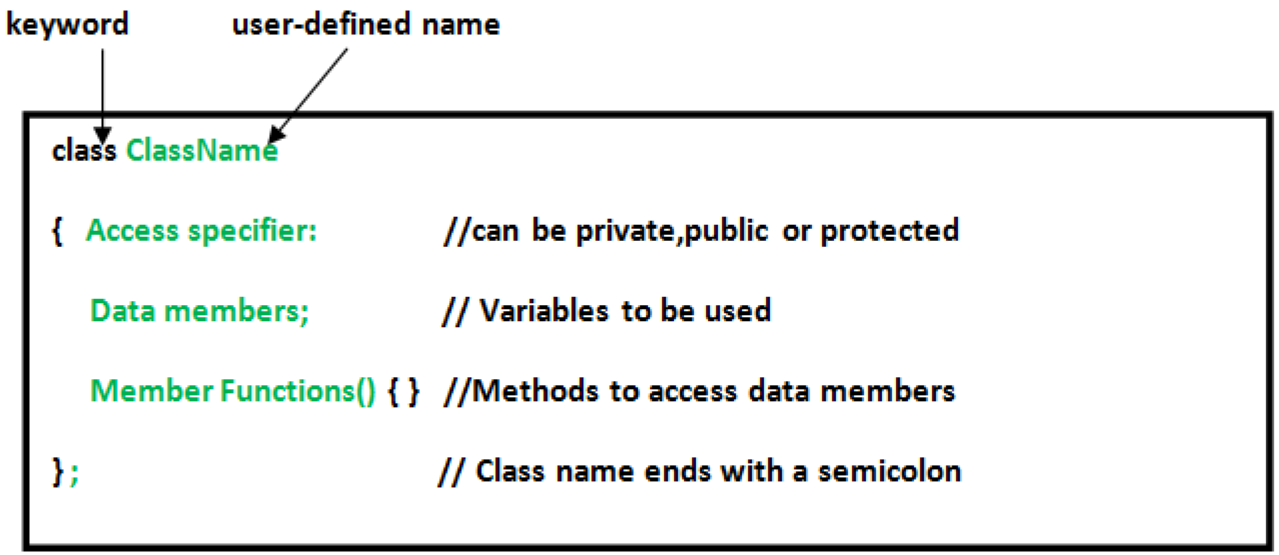
A C++ class is like a blueprint for an object, so basically an object is created from a class.

An ***Object is an instance of a Class***. When a class is defined, no memory is allocated but when an object is created memory is allocated.

To use the data and access functions defined in the class, you need to create objects.

Accessing data members and member functions: The data members and member functions of class can be accessed using the dot('.') operator with the object.

For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .



```
class ClassName
{  Access specifier:           //can be private,public or protected
   Data members;               // Variables to be used
   Member Functions() { }      //Methods to access data members
};                             // Class name ends with a semicolon
```

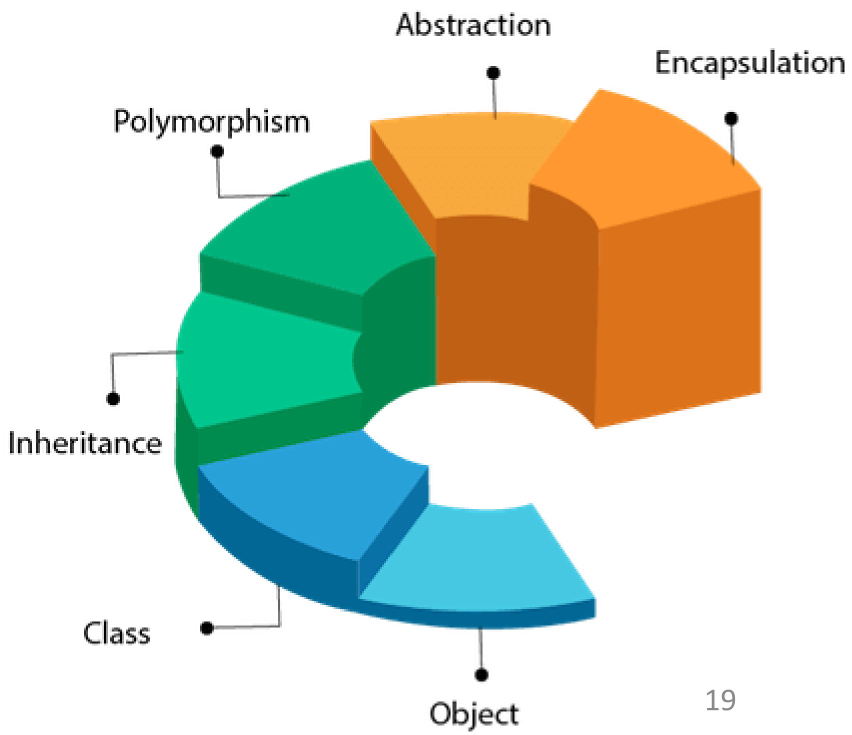
```
// C++ program to demonstrate
// accessing of data members

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
        public:                 // Access specifier
        string geekname;        // Data Members
        void printname()        // Member Functions()
        {
        cout << "Geekname is: " << geekname;
        }
};
int main() {
        Geeks obj1;             // Declare an object of class geeks
        obj1.geekname = "Abhi";       // accessing data member
        obj1.printname();       // accessing member function
        return 0;
}
```

https://www.tutorialspoint.com/compile_cpp_online.php

Object Oriented Programming

- The fundamental idea behind object oriented languages is to ***combine into a single unit, both data and functions***. Such a unit is called as object.
- OOP languages provide the programmer, the ability to create class hierarchies. Programmer ***can create modular and reusable code***, modify the existing modules.
- Object-oriented programming aims to ***implement real-world entities*** like inheritance, hiding, polymorphism, etc in 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.
- The fundamental concepts of object-oriented programming are
  - ✓ Object
  - ✓ Class
  - ✓ Inheritance
  - ✓ Polymorphism
  - ✓ Abstraction
  - ✓ Encapsulation



Class of OOP

- 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 a ***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***.
- ***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.

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 knowing 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.
- Data encapsulation led to the important OOP concept of **data hiding**.
- **Data encapsulation** is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the ***interfaces and hiding the implementation details from the user.***

C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called **classes**. A class can contain **private, protected** and **public** members.

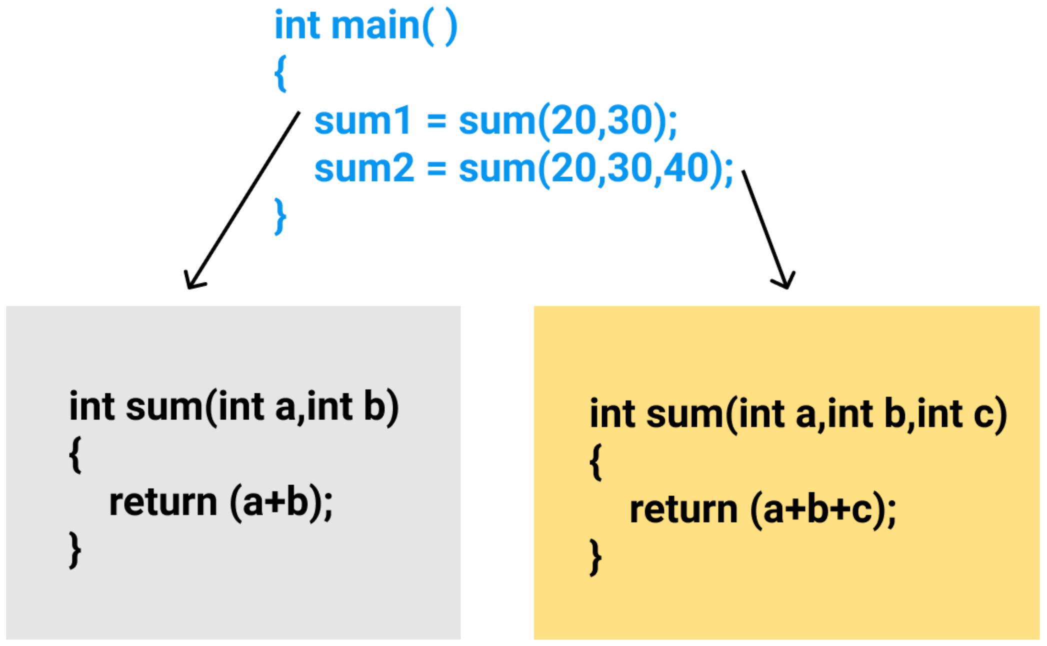
Why Encapsulation?

- In C++, encapsulation helps us ***keep related data and functions together***, which makes our code cleaner and easy to read.
- It helps to control the modification of our data members.
- Encapsulation also provides a way for data hiding. Data hiding is a way of ***restricting the access of our data members*** by hiding the implementation details.

Polymorphism

- Poly means many, and morph means forms. So Polymorphism means many forms. polymorphism defined as the ability of an operator of function 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.
- The ***ability to use an operator or function in different ways*** or giving different meaning to the operators or functions is called polymorphism.
- An operation may exhibit different behaviours in different instances. The behaviour ***depends upon the types of data used in the operation***.
- ***Operator overloading and function overloading are the two kinds of polymorphism are supported by C++*** .
- ***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.



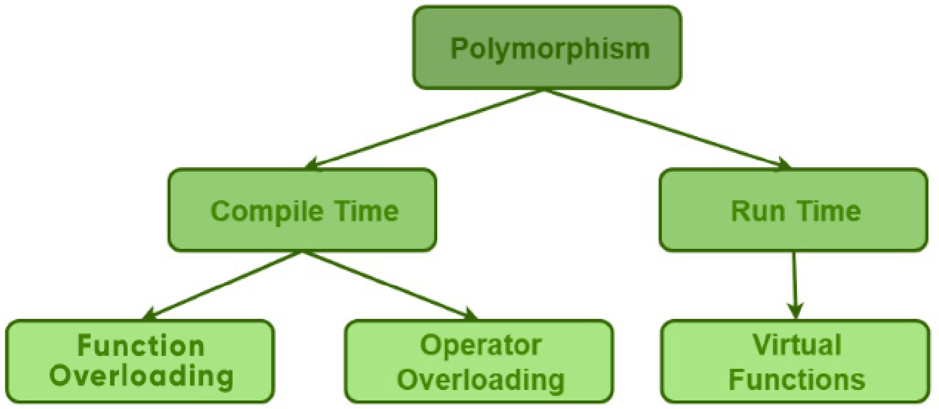
```
int main( )
{
    sum1 = sum(20,30);
    sum2 = sum(20,30,40);
}
```

```
int sum(int a,int b)
{
    return (a+b);
}
```

```
int sum(int a,int b,int c)
{
    return (a+b+c);
}
```

Overloading

- C++ allows to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.
- When there are multiple functions with same name but different parameters then these functions are said to be ***overloaded***.

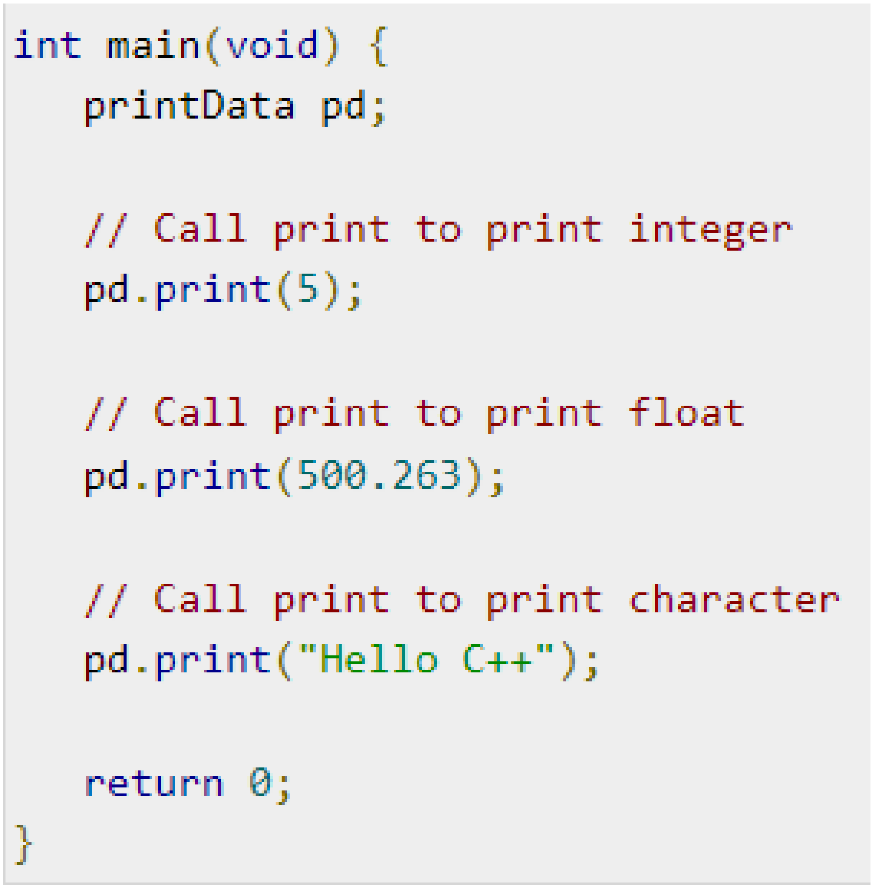


- When you call an overloaded **function** or **operator**, the compiler determines the ***most appropriate definition*** to use, by comparing the ***argument types*** you have used to call the function or operator with the parameter types specified in the definitions.
- The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

Function Overloading

- Function overloading is using a single function name to perform different types of tasks.
- Consider an object/class has several methods with the same name that take different parameters: Calculate(int a, float b), Calculate(int a, int b), Calculate(float a, float b)
- Here ***one of the three Calculate functions is executed*** depending upon the arguments passed to it. It's known as function overloading.

- You can have multiple definitions for the same function name in the same scope.
- The definition of the function *must differ from each other by the types* and/or the *number of arguments* in the argument list.
- In the example, same function ***print()*** is being used to print different data types



```
int main(void) {
   printData pd;

   // Call print to print integer
   pd.print(5);

   // Call print to print float
   pd.print(500.263);

   // Call print to print character
   pd.print("Hello C++");

   return 0;
}
```

Live Example
https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm

Operator Overloading

- The process of making an operator to ***exhibit different behaviours in different instances*** is known as operator overloading.
- Operator overloading is a type of polymorphism in which a single operator is overloaded to give user defined meaning to it.
- Operator overloading provides a flexibility option for creating new definitions of C++ operators.
- There are some C++ operators which we can't overload.
  - ✓ Class member access operator (. (dot), .* (dot-asterisk))
  - ✓ Scope resolution operator (::)
  - ✓ Conditional Operator (?:)
  - ✓ Size Operator (sizeof)
- An overloaded operator is used to perform an operation on the user-defined data type.

For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands.
So a single operator '+' when placed between integer operands , adds them and when placed between string operands, concatenates them.

Inheritance

- Inheritance is the process by which an object of one class ***acquires the properties of objects of another class***.
- The class which inherits the properties from another class and the relation is called ***parent – child relation***.
- The parent class which gives its properties to another class is known as ***Base class***.
- The child class which inherits properties from parent class is known as ***derived class***.
- It allows the extension and ***reuse of existing code*** without having to rewrite the code from scratch.
- The new derived class inherits the members of the base class and also adds its own.

Inheritance Contd…

Consider a class B inherits the class A, the class B is called the derived class or sub class, child class.

- ✓ The class A is called the base or super class, parent class
- ✓ The class B has two parts.
  - **Derived part**: which is inherited from the base class A
  - ***Incremental part***: which is the new code included in the class B.

For example, Maruti, sports cars and Benz are all types of cars

- ➢ In the object oriented language, sports cars, Maruthi and Benz are subclasses of the class CAR.
- ➢ The class CAR is a "super class" (parent class or base class) of Maruthi, Benz, and sports cars.
- ➢ Every subclass will inherit data (state) and functions (properties) from the super class.
- ➢ The various types of cars such as Maruthi and Benz will share certain properties such as ***break, escalator, steering*** etc.

Inheritance Contd…

- The attributes once declared in the super-class which are inherited by its subclasses, need not repeated. They can be accessed form any subclass unless they are private.
- Only the methods of a class can access its private attributes.
- The attributes which are declared as protected are accessible to subclasses.

Access Control and Inheritance

A derived class can access all the ***non-private*** members of its base class.
Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

Access	Public	Protected	Private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Type of Inheritance

- ➢ When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance.
- ➢ We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used.
- ➢ While using different type of inheritance, following rules are applied –

<u>Public Inheritance</u> –

- ✓ When deriving a class from a ***public*** base class, ***public*** members of the base class become ***public*** members of the derived class and ***protected*** members of the base class become ***protected*** members of the derived class.
- ✓ A base class's ***private*** members are never accessible directly from a derived class, but can be accessed through calls to the ***public*** and ***protected*** members of the base class.

<u>Protected Inheritance –</u>

- ✓ When deriving from a ***protected*** base class, ***public*** and ***protected*** members of the base class become ***protected*** members of the derived class.

<u>Private Inheritance –</u>

- ✓ When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

Single Inheritance:

It refers to deriving a class from a single base class .

Multiple Inheritance:

In this, the class is derived from more than one base class is known as multiple inheritance.

Example of Inheritance

https://www.tutorialspoint.com/compile_cpp_online.php

Function Overriding

- ✓ Function Overriding is **a feature that allows us to use a function in the child class that is already present in its parent class**.
- ✓ Function overriding means creating a newer version of the parent class function in the child class.
- ✓ Inheritance allows a derived class to inherit the ***properties*** of its parent class. Function overriding provides you with a ***way to override an existing functionality*** of a class inside a particular derived class.
- ✓ Function overriding in C++ is a concept by which you can define a function of the ***same name and the same function signature*** (parameters and their data types) in both the base class and derived class with a different function definition.
- ✓ It ***redefines a function of the base class inside the derived class***, which overrides the base class function.
- ✓ Function overriding is an implementation of the ***run-time polymorphism***. So, it overrides the function at the run-time of the program.

Advantages-

1. Function overriding helps to improve the ***readability of the code***.
2. If both the child class and base class have the same function, it ***will not affect the independence*** of the child class function
3. Overriding a ***function saves the memory***, increases the consistency of code, and ***enhances code reusability.***
4. A function with the same name can be used to ***perform different operations*** and hence makes the code clean.

Abstraction

- _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.
- ***Abstraction using Classes:*** We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.
- ***Abstraction in Header files***: One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

End of Unit-I