**C++ Language:**

**Lecture -1 :-date(2-7-25)**

🡪class data + functions

Data member AND MEMBER FUNCTION

Blueprint

🡪object -Real entitiey

Access data and function of a class

Object is type of class

Int a=20;

Ex.class person{};

Nisha=person();

Encapsulation-data member+member function in single unit(hiding data private)

Abstraction -showing only essential detail

Polymorphism -more than one form

Compile time/static binding/method overloading

Runtime/dynamic binding/method overriding

Inheritance-property on eclass can use in another class.

-reusability of code

-parent -child

Clas person{

Int contact\_number,age;

Void getdata(){

}};

Cout<<”hello world”;

Console output;

<<insertion operator

Std==namespace

Cin>>

Console input;

Extraction operator;

----------------------------------------------------------------------------------------------

🡪**lecture** -2:- (d-3-7-25)

Datatype

Primitive

Int

Float

Char

Access modifier-

By default in c++ all class member and function are private.

Public

Private

protected

----------------------------------------------------------------------------------------------

🡪lecture 3(d-4-7-2025)

Function in c++:-

Function cpp

Default parameter=indirectly method overloading

🡪string

/substring

//insert

//Replace

Encapsulation

//getter and setter function

Int salary

setSalary( int sal){

salary=sal;

}

Int getSalary(){

Return salary;

}

Constructor

Are special method

Name same name as class name

automatic called

Overloading

Its does not return any thing.

To initialize object

//two value constructoir int a,b

//call(arithmetic ) the method

//math(23,34);

//string manuplculation(class)

//string parameter one

yogesh

Append

Insert

Length

Substring

---------------------------------------------------------------------------------------------

🡪lecture -4(d-5-7-2025)

🡪Inheritance-

Data access

Reusability

Parent child

Property one one class can use the property of another class.

Class a{parent class,base class

}

Class b: public a{extend by a,child class,derived class

}

🡪types of inheritance

1.single

2.multiple

3.multilevel

4.hierarchical

5.hybrid

1.single inheritance

Class a{

};

Class b:public b{

};

2.multilevel

Class grand parent

{}

Class parent:public grandparent{}

Class child:parent{}

3.multiple

Class a{}

Class b{}

Class c:a,b{}

4.hierarchy

Class a{}

Class b:a{}

Class c:a{}

Hybrid:

Class a

:: scope resolution operator

----------------------------------------------------------------------------------------------

Lecture-5 (d-7-7-2025)

* Test

----------------------------------------------------------------------------------------------

Lecture -6 (d-8-7-2025)

* Polymorphism
* Compoile time
* Static binding,early binding
* Method overloading
* i. same method name and number of argument and type of argument are difeerent.
* Operator overloaidng
* i.all arithmetic operation work with object its call operator overloading
* Run time
* Dymamic binding,late binding
* Method overriding
* //task
* Create multiplication function
* Which takes 2,3,4 argument as int and perform operation accourndingly.
* Also create 2 more same function with mixing of double argument and perorm accordingly.

---------------------------------------------------------------------------------------------

Lecture-7(9-7-2025)

* 🡪Operator overloading
* d3=d1+d2;
* D1 operator+(d2){
* //means first argument as return and second under bracket for work
* }
* Binary operatoar(+,-,\*,/)
* Working with more than one operand
* Unary operator(++,--)
* Working with only one operand
* This keyword or this pointer is used to reference current object.

----------------------------------------------------------------------------------------------

Lecture-8(10-7-25)

🡪method overriding

* Method have same name same parameter and same type
* Used in inheritance
* Must write virtual keyword in base class to overrid or change data for derived class in same method.

🡪Scope resolution operator(::)-

* used to global variable and function .
* used to reperent same name function or method to clarify which method of which class when both class have same name method using ::.

|  |  |  |
| --- | --- | --- |
| * **Situation** | * **Scope resolution needed?** | * **Why** |
| * Defining class functions outside class | * ✅ | * To associate with the class |
| * Accessing global variable when local exists | * ✅ | * To clarify scope |
| * Using namespaces | * ✅ | * To avoid collisions |
| * Accessing static class members | * ✅ | * They belong to class, not object |

----------------------------------------------------------------------------------------------

Lecture-9(11-7-2025)

🡪Abtraction- showing essential details and hiding detailing

* An **abstract class** is a class that **cannot be instantiated** (i.e., you cannot create objects of it) and is used as a **base class** for other classes. It is used to **define a common interface or structure** that derived (child) classes must follow.

**Key Points:**

* **Contains at least one pure virtual function**.
* Used to **enforce a contract** for subclasses.
* Child classes **must override** the pure virtual functions.
* You **cannot create objects** of an abstract class.

**Why Use Abstract Classes?**

* To **enforce a standard interface** for all derived classes.
* To provide **polymorphic behavior**.
* To achieve **partial implementation**, where base class provides some code and expects child classes to implement the rest.
* Data hiding by making them private(c\_no,salary)
* Public(get(),display()

🡪Pure virtual function:-

* A **pure virtual function** is a function in a base class that **has no body** and **must be overridden** by derived classes.
* It is used to create **abstract classes**, which define a **common interface** for all derived classes.

Ex:

Virtual void msg()=0;

**Key Points:**

* A class with at least one pure virtual function is an **abstract class**.
* You **cannot create objects** of an abstract class.
* Derived classes **must override** the pure virtual function(s).

**Why Use Pure Virtual Functions?**

* To define a **standard interface** (set of rules) for child classes.
* To support **runtime polymorphism**.
* To make sure every derived class **must implement** its own version of the function.
* **Difference Between Pure Virtual Function and Simple Virtual Function in C++**

|  |  |  |
| --- | --- | --- |
| **Feature** | **🧩 Simple Virtual Function** | **🎯 Pure Virtual Function** |
| **Definition** | A function with a body in base class | A function with no body in base class (= 0) |
| **Syntax** | virtual void show() {} | virtual void show() = 0; |
| **Must Override in Derived?** | ❌ Not necessary (optional override) | ✅ Must be overridden in derived class |
| **Creates Abstract Class?** | ❌ No | ✅ Yes |
| **Object Creation** | ✅ You can create objects of the class | ❌ You **cannot** create objects of abstract class |
| **Purpose** | Allows **optional** overriding and polymorphism | Enforces **mandatory** overriding (interface-style) |

🡪1.if class contain one or more pure virtual function then that class is abstract class.

🡪2. Can not create object of abstract class

🡪3.must have override pure virtual function in derived class.

🡪4.virtual function it have body{}

🡪5.Pure virtual function not have body but =0;

🡪difference between abstract and concreat class:

* 1.Object- you can not create object of abstract class
* While you can create object of concreat class.
* 2.methods- Abstact class must contain one pure virtual function
* Abstarct class have normal function also.
* While concreat class not have any pure virtual function.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Abstract Class** | **Concrete Class** |
| Can create objects? | ❌ No | ✅ Yes |
| Pure virtual function? | ✅ At least one | ❌ None |
| Purpose | Provides a structure/interface | Provides full implementation |
| Usage | Used as base class only | Can be used directly |

🡪Difference between Virtual and normal function method : -

* Virtual function must be override while normal may or may not be overridden
* A **virtual function** is a member function in a base class that you **expect to override** in derived (child) classes. It supports **runtime polymorphism** (i.e., the correct function is called **based on the object type**, not pointer type).

|  |  |  |
| --- | --- | --- |
| * **Feature** | * **🟢 Normal Function** | * **🔵 Virtual Function** |
| * **Definition** | * Regular member function | * Declared with virtual keyword |
| * **Overriding in Derived Class** | * Possible, but resolved at **compile time** | * Overriding is resolved at **runtime** |
| * **Polymorphism Support** | * ❌ No **runtime polymorphism** | * ✅ Supports **runtime polymorphism** |
| * **Function Call Based On** | * Type of **pointer/reference** | * Type of **actual object (instance)** |
| * **Keyword Used** | * None | * virtual |
| * **Use Case** | * General behavior | * To allow different behavior in derived classes |

🡪Lecture-10(12-7-25):-

🡪inline function

Simple function there is no complexity

No loop

No recursive

🡪friend function,class

Access the private data of classes using simple function

* It is not a part or member of class but also must be declare in class ;
* Friend keyword
* Ex;
* Friend void display1(A);
* Also pass class and its object as argument.

🡪friend vs normal function

-private data access by friend while normal function not access

Member not a friend whilke normal also

* Calling
* Parameter must class object
* Must be declare first in class friend function
* While normal no need any declaration in class;

//task

Lecture -11(14-7-2025)

* Friend function /friend class
* Main goal is private or protected data access.

🡪most importanant

🡪static data -concept

* Static data initialize only ones.
* Its not belong to object its belong to class
* Its access by return type class name::any method or data initialize
* Its also when you required data only once time initialize.
* For memory management purpose its also use

🡪static method:-

* Static method can not use non static data.
* Static method also use static data.
* Todays task.
* Employee s data are multiple but company name is same

Lecture-12(d-15-7-2015)

* Malloc and calloc(dynamic memory allocation run time memory allocation
* Malloc take only one parameter it will be size ,by defaut garbage values initialize
* Calloc takes two parameters ,by defaut 0 initialize
* Both function return \* void pointer.
* Generally used in array

🡪destructor:-

A **destructor** is a **special member function** in C++ that is **automatically called** when an object **goes out of scope** or is **explicitly deleted**. It is used to **free resources** that the object may have acquired during its lifetime.

**🔹 Syntax of Destructor:**

~ClassName();

* Same name as the class, **preceded by a tilde ~**
* No return type
* No parameters
* Only one destructor per class (not overloaded)

~Class name(){}

🡪free function

* Its memory deallocation function in c used with stdlib library
* It also used when with malloc and calloc only.

Free();

🡪file handling:

Read,write,append;

Open read/write/close;

* Fstream header file also include
* Ifstream-reading only
* Ofstream-write only
* Fstream-read/write both
* Cout
* cin

lecture-13(16-7-25)

* file handling:
* task
* menu driven
* 1.to read-which file?
* Press 1 for Read by line/
* Press 2 read by word
* 2.to write-which file
* 3 to append ios::app

Lecture -14(17-7-25)

* File
* Position -current ,ask/put
* File.tellg() function by default
* File.seekg(9) take position
* Fstream
* Escape sequence=\n,\t,\”.
* Ios::out=write
* Ios::in=read
* Ios::app=append
* Database oraganize way data storing
* Dbms
* Rdbms
* Procedure oriented
* object orientetsd
* database language sql structure query language
* pl/sql
* table row,col where data can store
* row-data
* column=field