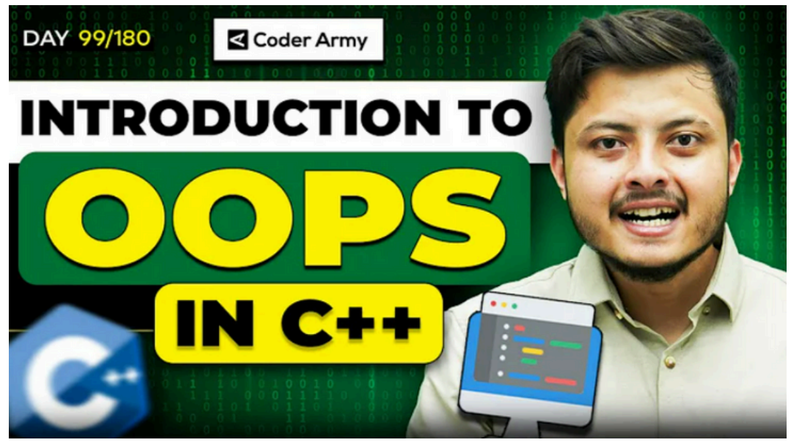
OBJECT ORIENTED PROGRAMMING IN C++

Introduction <What is OOP?>



It is an approach or a programming pattern where programs are structured around object rather than function and Logic

CLASS → It is User Defined data type. Blueprint for creating objects

OBJECT → It is an entity that has a state and behavior Anything that exist in Physical World

Coder Army

keyword — user-defined name

```
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
```

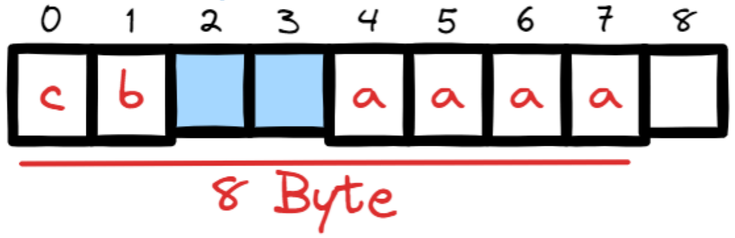
Complete Code Implementation — GitHub

Concept of Padding

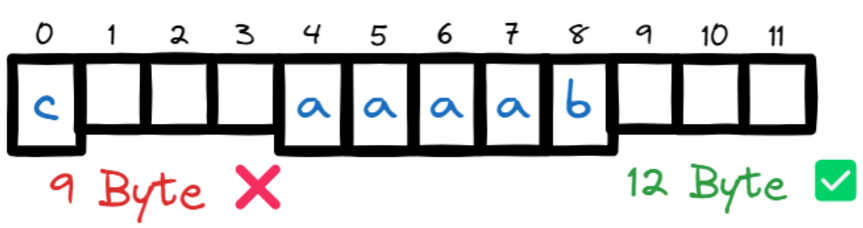
Complete Code Implementation — GitHub

Padding

```
char c;
char b;
int a;
```



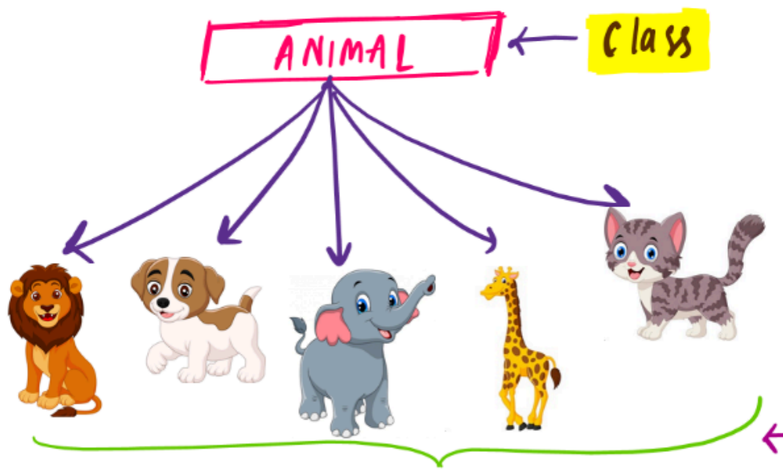

```
char c;
int a;
char b;
```



9 Not divisble by 4 next 12 divisible

The method is compiler dependent and kind of greedy. It aligns till the boundary of maximum memory allocated.

REAL WORLD EXAMPLE



All of them share common properties :
a) 4 legs
b) 1 tail
c) Breeds
d) Color

Static Vs Dynamic Memory Allocation

```
// pointer to store the address returned by the new
int* ptr;
// allocating memory for integer
ptr = new int;

// assigning value using dereference operator
*ptr = 10;

// printing value and address
cout << "Address: " << ptr << endl;
cout << "Value: " << *ptr;
```

```
int *p = new int
```

Heap

stack

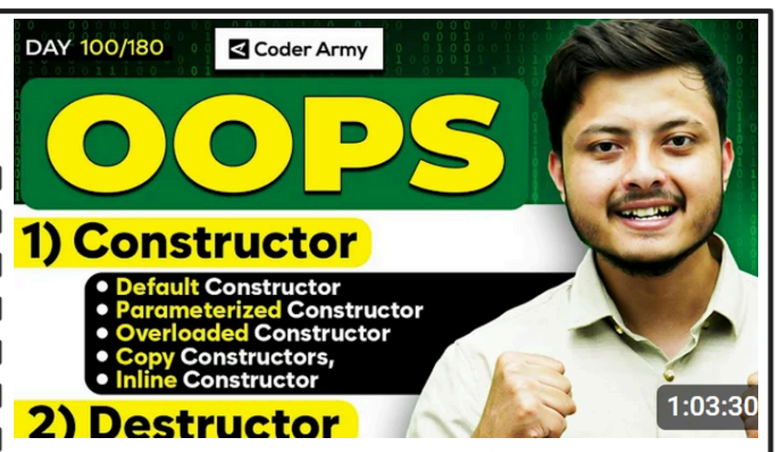
Class And Object Dynamically

!! Complete Code Implementation ->

Complete Code Implementation — GitHub

Constructor And Destructor in C++

Coder Army



- It is a special Function that is invoked automatically at the time of object creation
- Name of the constructor should be same as class name
- It doesn't have any return type.
- It is used to initialize the value
- Constructor is an instance member.
- If we define constructor in private then we can't create object outside the class.
- Constructor can never be a static member

```
class customer{
    string name;
    int account-number;
    int balance;

customer(){
    cout<<"Constructor is Called";
}
customer(string a, int b, int c)
{
    name = a;
    account-number = b;
    balance = c;
}

}

int main(){
    customer A1("Rohit", 123, 10000);
}
```

Default Constructor

Parameterized Constructor

Programmer has to define constructor, so that he can write any code but it is useful to initialize properties of an object

Copy Constructor

- Either programmer has to provide copy constructor or compiler itself provides copy constructor
- Copy Constructor is invoked for newly created obj. which is initialized with the obj. of the same class
- Formal Argument of copy constructor must be a reference variable of same class. Because if we Don't make reference Variable then it will be infinite recursion

Complete Code Implementation GitHub

```
Customer(Customer &B){
    name = B.name;
    account-number = B.account-number;
    balance = B.balance;
}

int main(){
    Customer A2(A1);
}
```

=> When there is no explicit constructor
-> Compiler make 2 constructor Default & Copy constructor

Q.When Compiler makes implicit copy constructor then why need to make explicit copy constructor by programmer?

Ans. When we want to make some changes in copy constructor then we make explicit copy constructor

Constructor And Destructor in C++

- ⇒ Destructor is a special member function of the class whose name is same as the name of the class but preceded with a (~) symbol.
- ⇒ Destructor has no return type.
- ⇒ Destructor has no argument therefore overloading of destruction is not possible.
- ⇒ Destructor is an instance member function.
- ⇒ Destructor is invoked implicitly when object is about to Destroy.
- ⇒ The job of destructor should be to free up the memory resources handled by the object.
- ⇒ In the absence of explicit destructor, compiler defines an implicit destructor in the class which has empty body.

‼ Complete Code Implementation GitHub

Coder Army

Types of Constructors in C++

Constructors can be classified based on in which situations they are being used. There are 4 types of constructors in C++:

1. **Default Constructor**: No parameters. They are used to create an object with default values.
2. **Parameterized Constructor**: Takes parameters. Used to create an object with specific initial values.
3. **Copy Constructor**: Takes a reference to another object of the same class. Used to create a copy of an object.

This Pointer

- -> 'this' is a keyword
- -> this is a local object ptr in every instance member function which contains address of the current object

USE CASES

1. Name conflict b/w instance member variables and local variables

```
eg. class complex
    {
        int a ,b;
    public:
 void set_data(int a, int b)
  {
    this->a = a;
    this->b = b;
  }
};
int main(){
    complex c1;
    c1.set_Data(3,4);
    }
```

Local variables

Caller Object's Variable

2. whenever it is required to represent current object in instance member function

```
Class customer{
    string name;
    int *balance;
public :
customer(String name, int bal){
    this->name = name;
    balance = new int;
    *balance = bal;
    }
~Customer() //Last Fn called
    {
    delete balance;
        }
};
int main(){
    customer A1("Rohit",1000);
    }
```

Complete Code Implementation GitHub

Static Data Member And Member Function

Coder Army

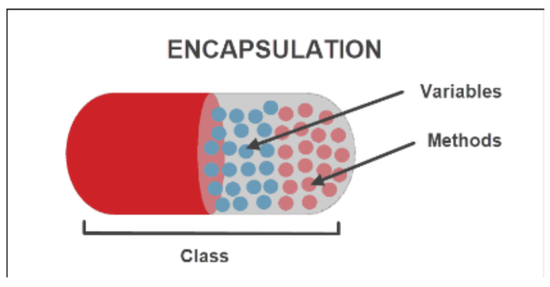
- They are attribute of classes or class member
- It is declare using static keyword
- Only One copy of that member is created for the entire class & is shared by all the object.
- It is initialized before any object of this class is created

```
Class Customer{
    string name;
    int Acc_No, balance;
    static int Total_Customer;
public:   Keyword
    Customer(string a,int b,int c){
        name = a;
        Acc_no = b;
        balance = c;
        Total_Customer++;
    }
int Customer :: Total_Customer = 0;
};

int main(){
    Customer A1("Rohit",1,1000);
    Customer A2("Mohit",2,1010);
}
```

Complete Code Implementation GitHub

ENCAPSULATION



Wrapping up of data and info in a single unit, while controlling access to them

Example

```
class customer{
    public:
string name;
int balance;
int age;

customer(string a, int b,int c){
    name = a;
    balance = b;
    age = c;
}
void deposit(int amount){
    if(amount>0)
    balance += amount;
    else
    count<<"Invalid amount";
    }
};

int main(){
    customer A1("Rohit", 1000,20);
    A1.deposit(100); ✅
    A1.deposit(-400); ❌
}
```

Complete Code Implementation GitHub

Features of Encapsulation

Below are the features of encapsulation:

1. We can not access any function from the class directly. We need an object to access that function that is using the member variables of that class.
2. The function which we are making inside the class must use only member variables, only then it is called *encapsulation*.
3. If we don't make a function inside the class which is using the member variable of the class then we don't call it encapsulation.
4. Encapsulation improves readability, maintainability, and security by grouping data and methods together.
5. It helps to control the modification of our data members.

ABSTRACTION

Displaying only essential information and hiding the details

Customer And Bank Real World Example in Code

Advantages of Data Abstraction

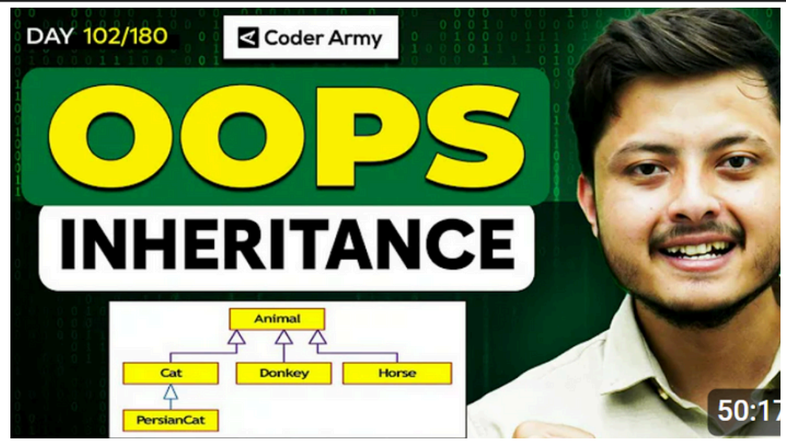
- Helps the user to avoid writing the low-level code
- Avoids code duplication and increases reusability.
- Can change the internal implementation of the class independently without affecting the user.
- Helps to increase the security of an application or program as only important details are provided to the user.
- It reduces the complexity as well as the redundancy of the code, therefore increasing the readability.

‼ Complete Code Implementation

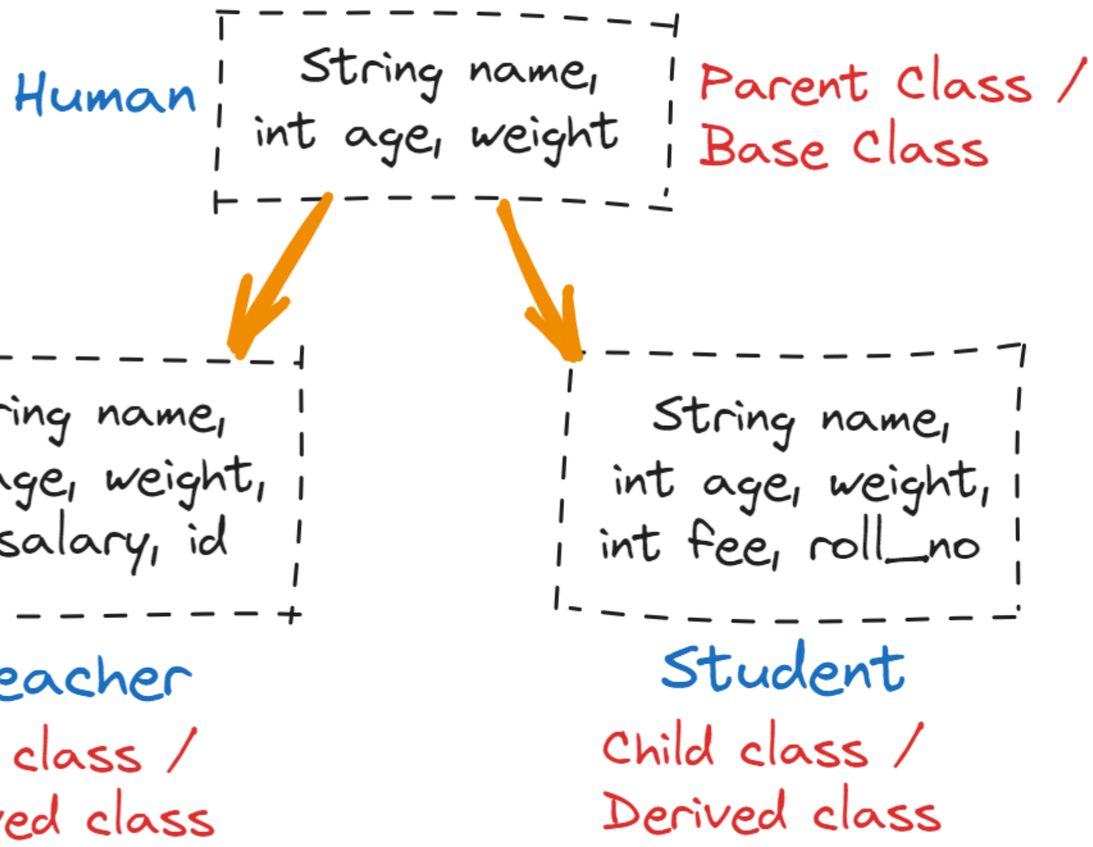
Complete Code Implementation GitHub

INHERITANCE & ACCESS MODIFIER

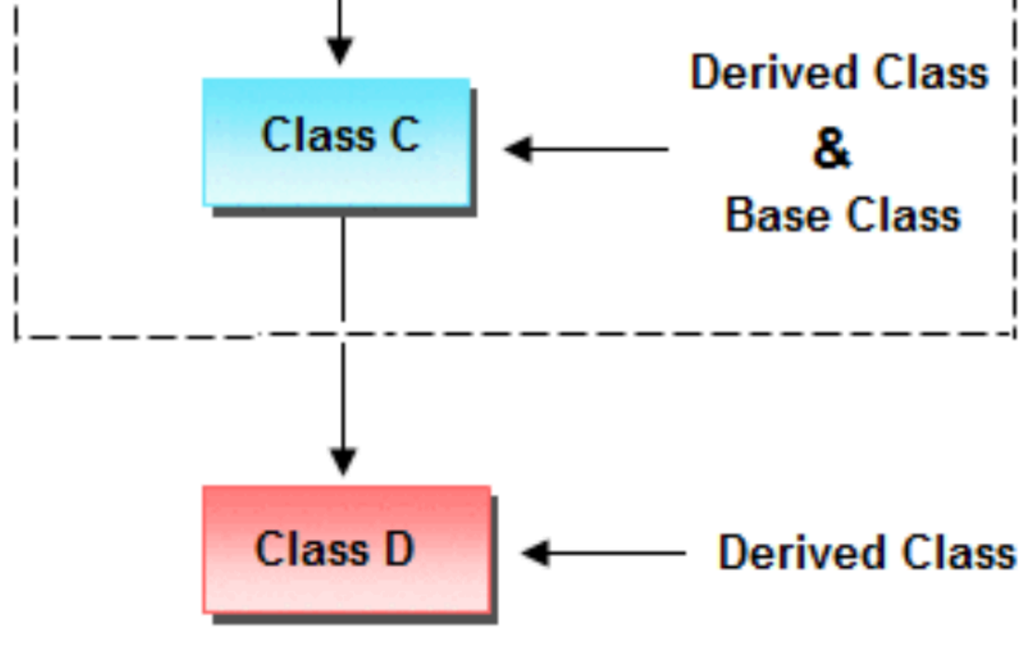
Coder Army



=> The capability of a class to derive property & characteristic from another class



	External Code	Within Class	Derived Class
Public	Yes	Yes	Yes
Private	No	Yes	Yes
Protected	No	Yes	No



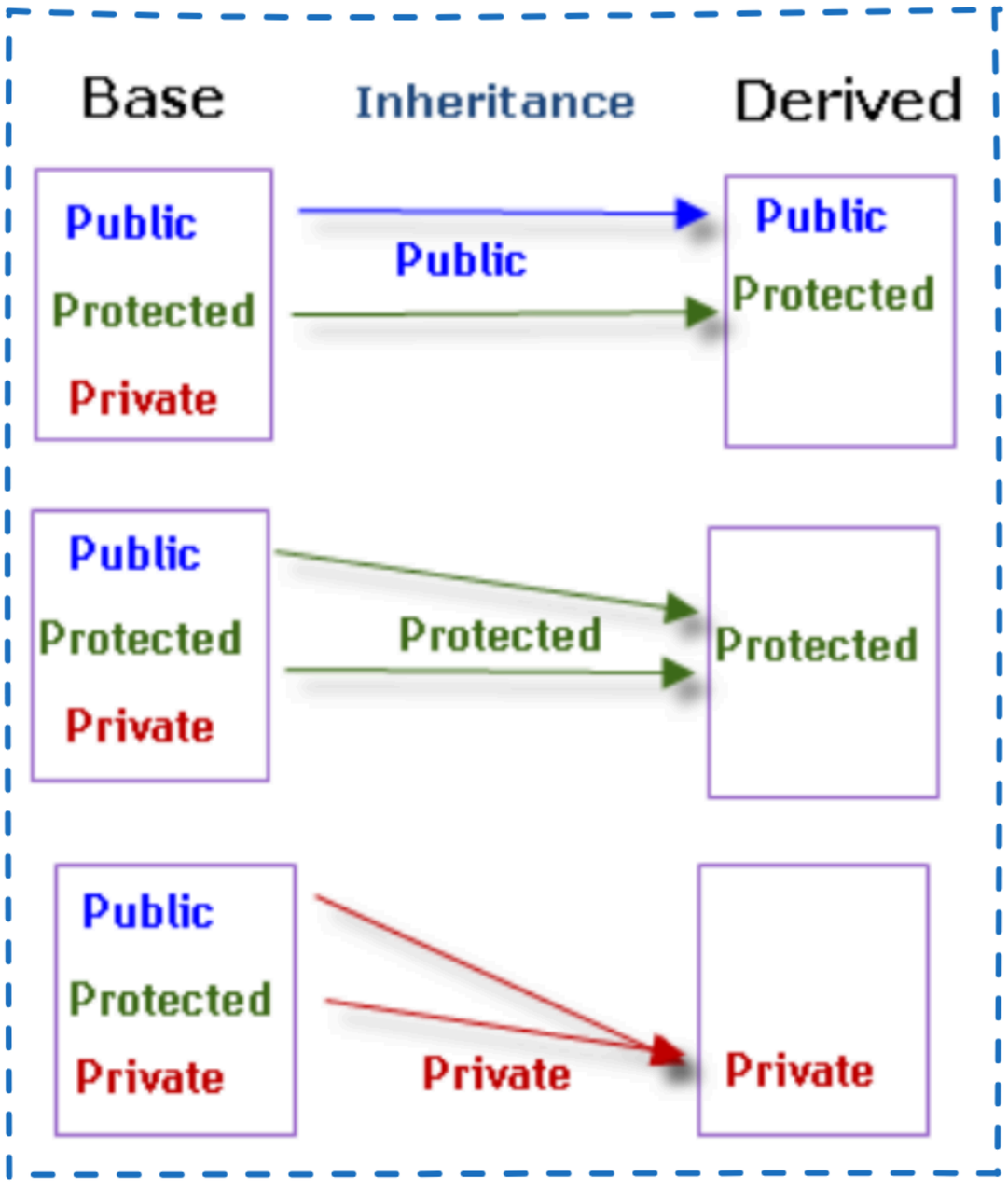

```
// Base class that is to be inherited
class Parent {
public:
    // base class members
    int id_p;
    void printID_p(){
        cout << "Base ID: " << id_p << endl;
    }};
// Sub class or derived publicly
inheriting from Base
// Class(Parent)
class Child : public Parent {
public:
    // derived class members
    int id_c;
    void printID_c(){
        cout << "Child ID: " << id_c << endl;
    }};
int main(){
    Child obj1;
    obj1.id_p = 7;
    obj1.printID_p();
    obj1.id_c = 91;
    obj1.printID_c();

    return 0;
}
```

Complete Code Implementation GitHub

OUTPUT

```
Base ID: 7
Child ID: 91
```



‼ Complete Code Implementation

Complete Code Implementation GitHub

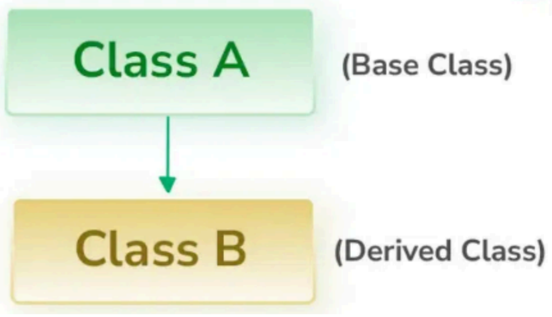
TYPES OF INHERITANCE IN C++

Coder Army

Single Inheritance

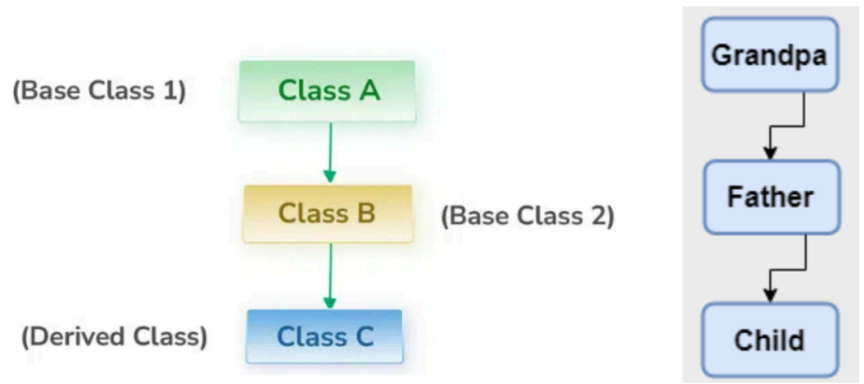
Complete Code Implementation GitHub

In single inheritance, a class is allowed to inherit from only one class. i.e. one base class is inherited by one derived class only.



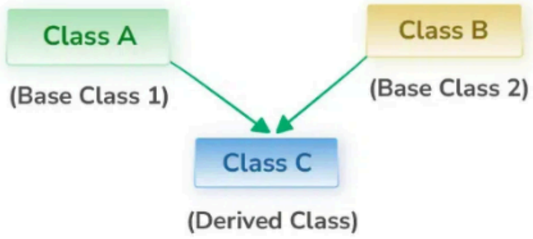
Multilevel Inheritance

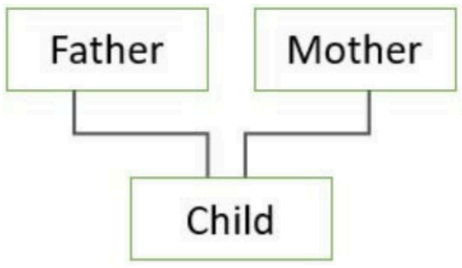
In this type of inheritance, a derived class is created from another derived class and that derived class can be derived from a base class or any other derived class. There can be any number of levels.



Multiple Inheritance

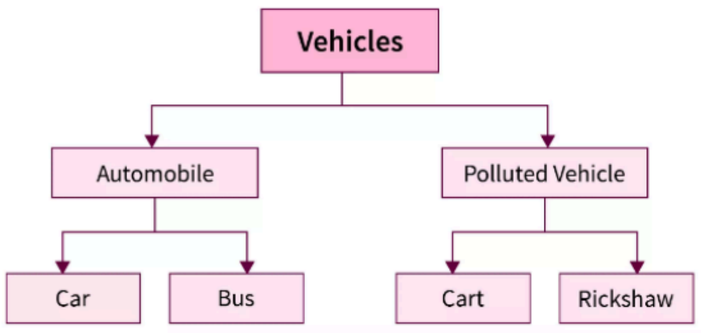
Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one **subclass** is inherited from more than one **base class**.



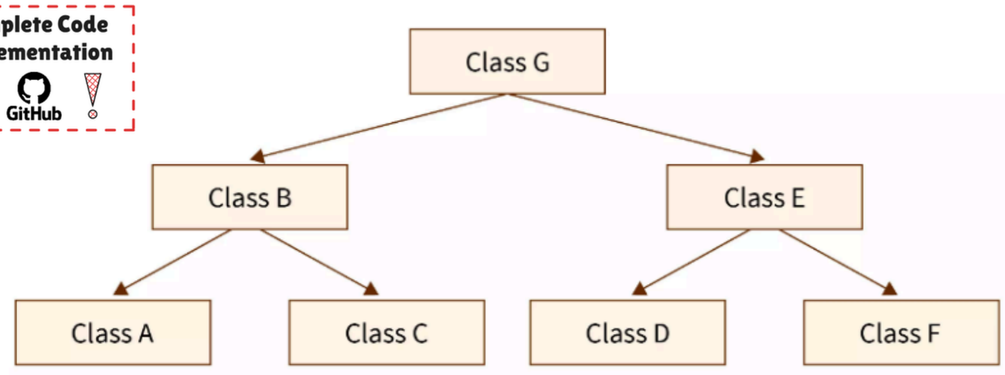


Hierarchical Inheritance

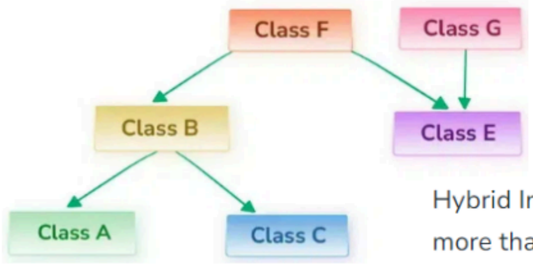
In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



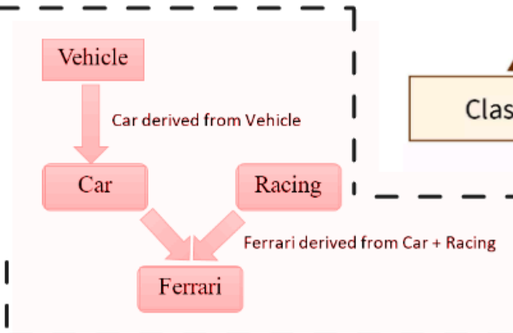
Complete Code Implementation GitHub



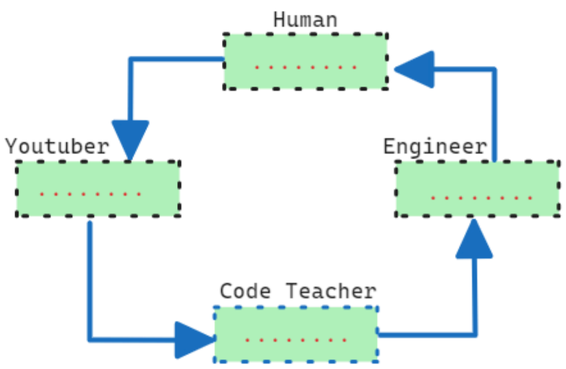
Hybrid Inheritance

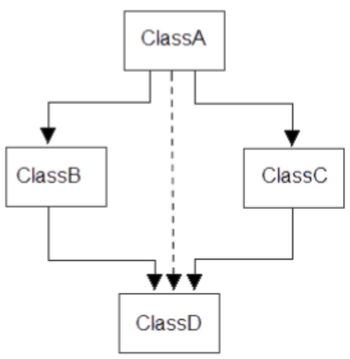


Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance will create hybrid inheritance in C++



Multipath Inheritance





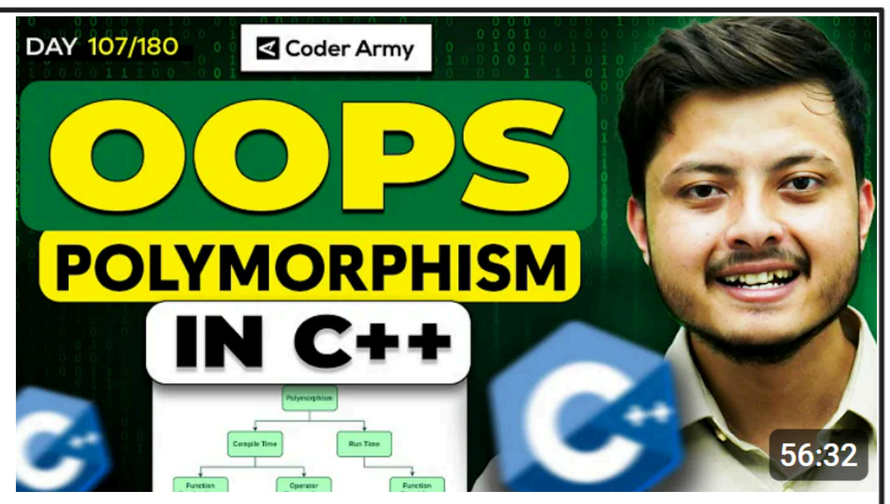
All Types of Inheritance					
Single	Multilevel	Multiple	Hierarchical	Hybrid	Multipath

!! Complete Code Implementation

Complete Code Implementation GitHub

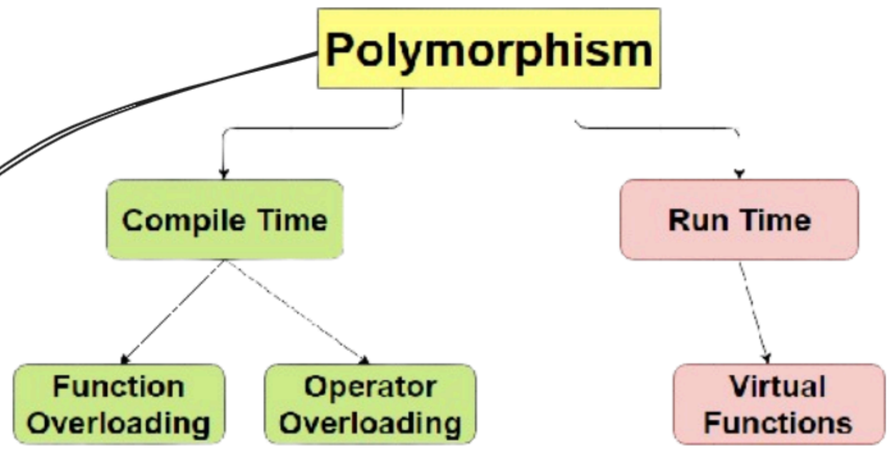
Polymorphism and Virtual Function

Coder Army



POLY (Many) MORPHISM (Forms)

Geek Word



Function overloading

When there are multiple functions with the same name but different parameters, then the functions are said to be overloaded, hence this is known as Function Overloading. Functions can be overloaded by changing the number of arguments or/and changing the type of arguments.

eg

```
class Area{
public:
    // Function Overloading
    int calcArea(int r){
        // Circle
        return 3.14 * r * r;
    }

    int calcArea(int l, int b){
        // Rectangle
        return l * b;
    }
};
int main(){
Area A1,A2;
cout<<A1.calcArea(4)<<endl;
cout<<A2.calcArea(3,4)<<endl;
}
```

OPERATOR OVERLOADING

eg

```
Class complex{
    int real,img;
    public:
    complex(int real,int img){
    this->real = real;
    this->img = img;}
void display(){
    cout<<real<<" +i "<<img;
    }
Complex Operator +(Complex &c){
    complex ans;
    ans.real = real + c.real;
    ans.img = img + c.img;
    return ans;
    }
};
int main(){
    complex c1(3,2);
    complex c2(4,6);
    complex c3 = c1+c2;
    c3.display();
}
```

Complete Code Implementation GitHub

Eg. int a = 5, int b = 10;
cout<<a+b; //15

string str1 = "Rohit";
string str2 = "Negi";
cout<<str1+str2;

=> Here Operation is same but working differently according to the data types.

Virtual Function

```
Class Animal{
    public:
    void speak(){
        cout<<"HuHu";
    }
};
class Dog:public Animal{
    public:
    void speak(){
        cout<<"Bark";
    }
};
int main(){
    Animal *p;
    p = new Dog();
    p->speak(); ⇒ It will Point Huhu;
}
```

Complete Code Implementation GitHub

A virtual function is a member function that is declared within a base class and is re-defined by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual fn for that object & execute the derived class's version of the method.

Rules for Virtual Functions

The rules for the virtual functions in C++ are as follows:

1. Virtual functions cannot be static.
2. A virtual function can be a friend function of another class.
3. Virtual functions should be accessed using a pointer or reference of base class type to achieve runtime polymorphism.
4. The prototype of virtual functions should be the same in the base as well as the derived class.
5. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
6. A class may have a virtual destructor but it cannot have a virtual constructor.

Note: *If we have created a virtual function in the base class and it is being overridden in the derived class then we don't need a virtual keyword in the derived class, functions are automatically considered virtual functions in the derived class.*

For More Details --> Check Code Part

‼ Complete Code Implementation

Complete Code Implementation GitHub

C++ | Exception Handling

Coder Army

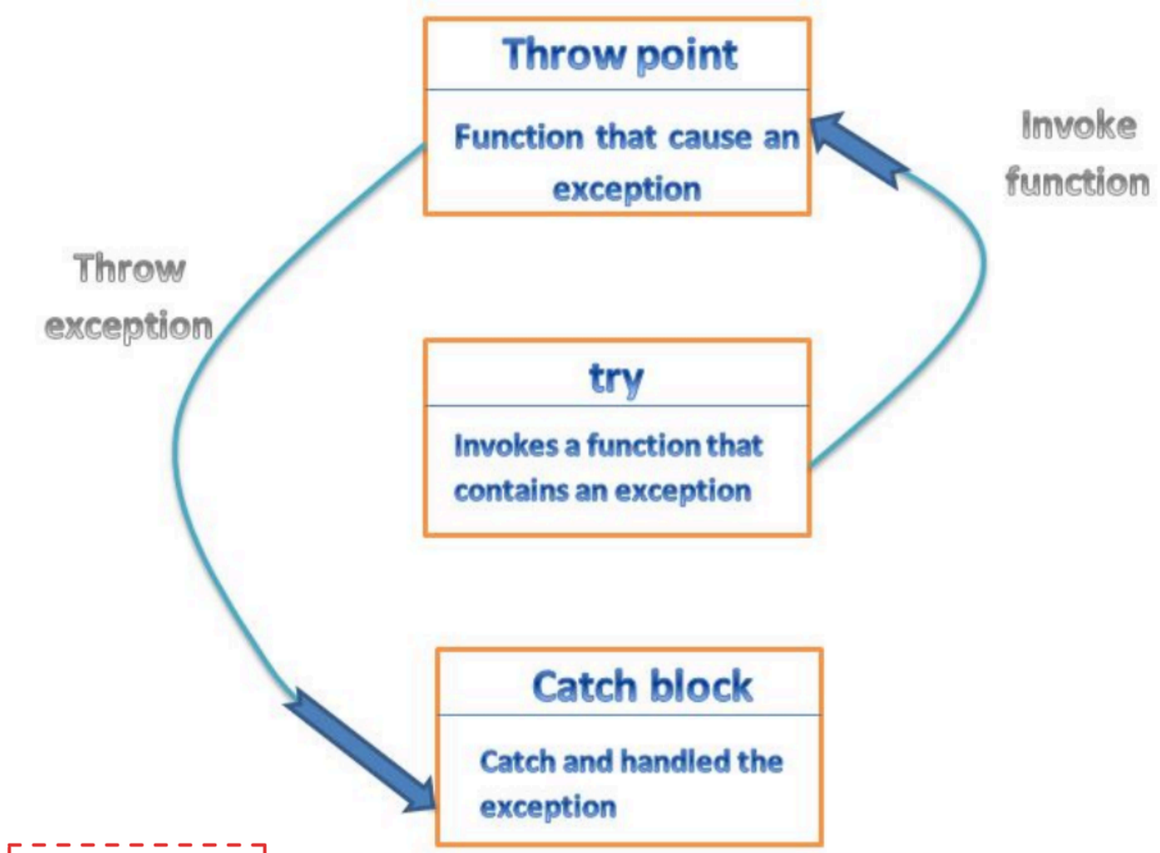
An Exception is an unexpected problem that arises during execution of a program And our program terminate suddenly with some error/issues

TRY — It represent a block of code that may through a exception placed inside the try block

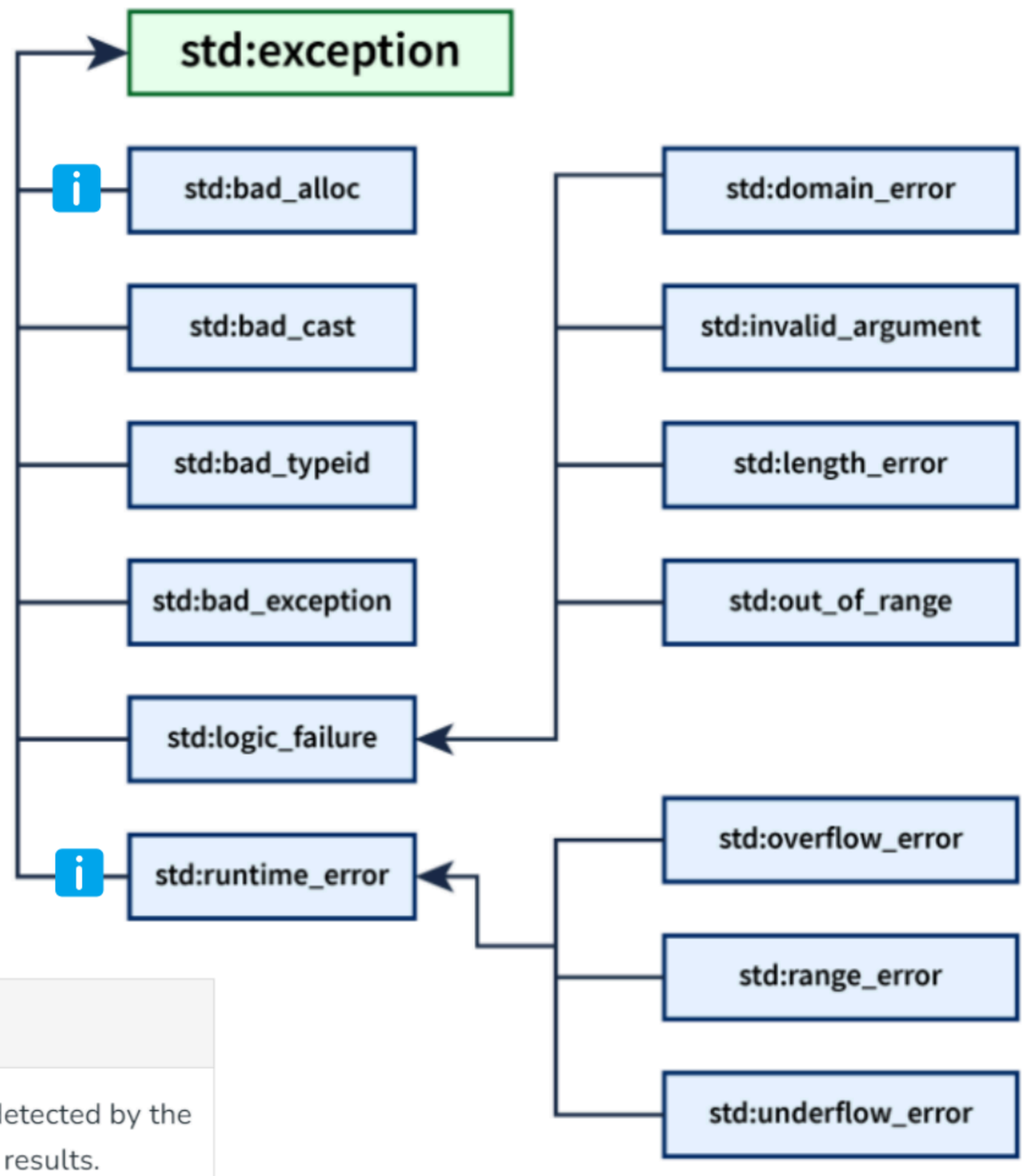
Catch — It represent a block of code that is executed when a particular exception is thrown from the try block

Through — An Exception in C++ can be thrown using the throw keyword.

Complete Code Implementation GitHub



Complete Code Implementation GitHub



Compile-Time Errors	Runtime-Errors
These are the syntax errors which are detected by the compiler.	These are the errors which are not detected by the compiler and produce wrong results.
They prevent the code from running as it detects some syntax errors.	They prevent the code from complete execution.
It includes syntax errors such as missing of semicolon(;), misspelling of keywords and identifiers etc.	It includes errors such as dividing a number by zero, finding square root of a negative number etc.

!! Complete Code Implementation

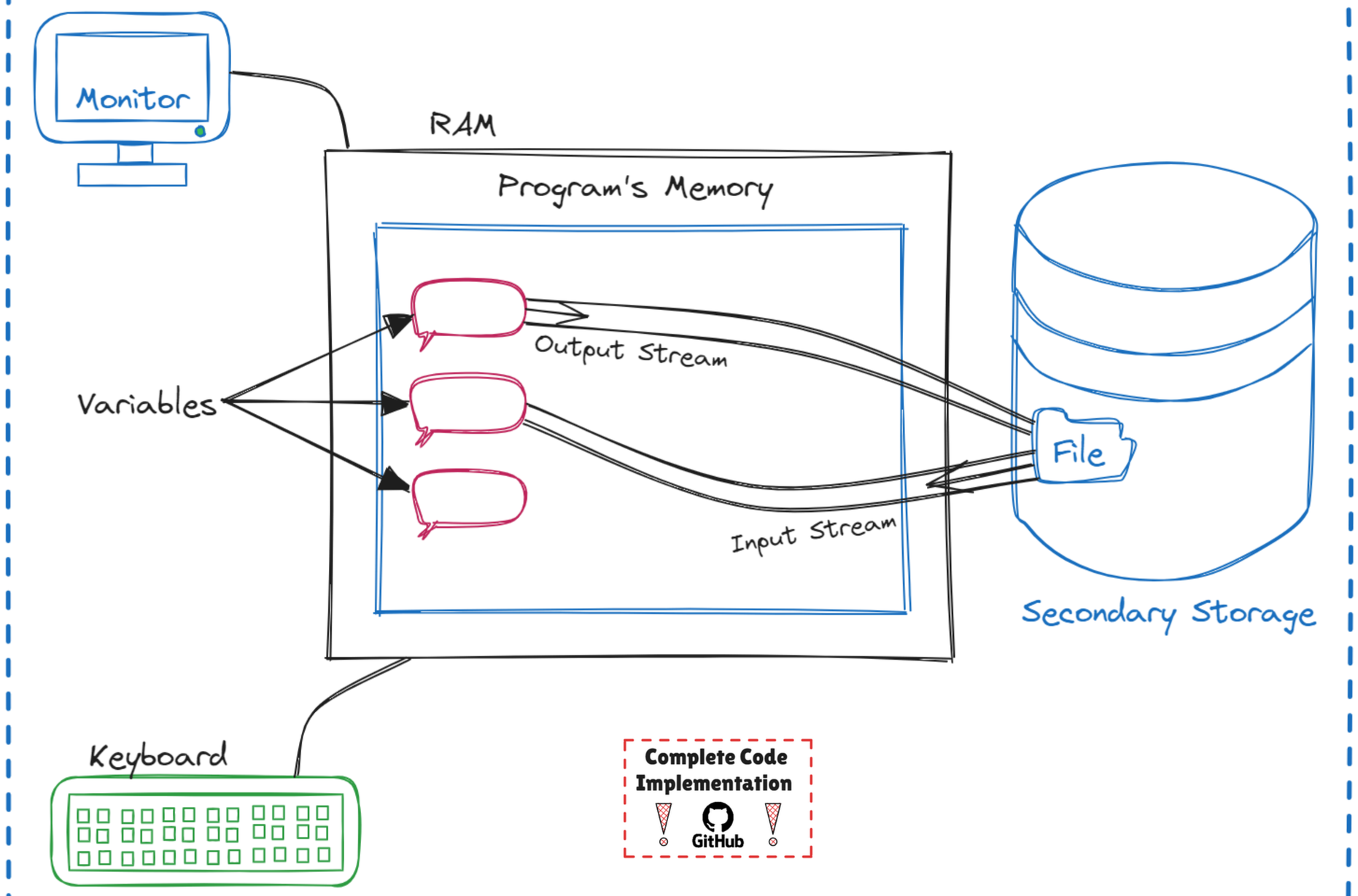
Complete Code Implementation GitHub

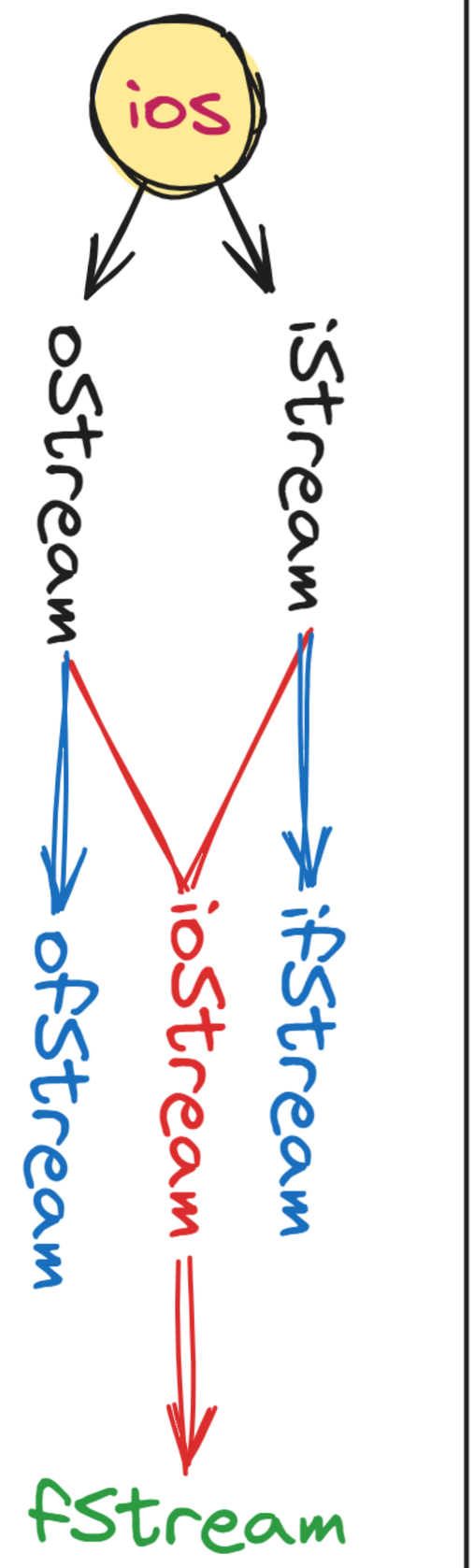
C++ File Handling

Coder Army

=> If you want to do manual tasks of files automatically then file Handling comes into place.

=> Our codes are executed in RAM & if you want to store the result for later use. then you have to store it in secondary Memory





File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk).

How to achieve the File Handling

For achieving file handling we need to follow the following steps:-

STEP 1-Naming a file

STEP 2-Opening a file

STEP 3-Writing data into the file

STEP 4-Reading data from the file

STEP 5-Closing a file.

!! Complete Code Implementation

Complete Code Implementation
GitHub

ifstream	ios::in
ofstream	ios::out
fstream	ios::in | ios::out