

## "C++"

- Started in ⇒ 1979-80
- Renamed in ⇒ 1983  
As  
C with classes / C++ / CPP
- Developed by ⇒ Bjarne Stroustrup
- First appeared in ⇒ 1985
- Developed at ⇒ AT&T Bell Labs
- 4 Main Pillars ⇒ A. Inheritance      C. Data Abstraction  
                      B. Polymorphism      D. Encapsulation

\* OOPs  $\Rightarrow$

→ Object Oriented Programming Structure

→ Concepts  $\Rightarrow$

- A. Class
- B. Object
- C. Encapsulation
- D. Data Abstraction
- E. Polymorphism
- F. Inheritance
- G. Dynamic Binding
- H. Message Passing

## A. Class ⇒

- Blueprint of the program.
- User defined Datatype.
- Contains tools to implement program.
- Class में variables & functions होते हैं, जिन्हें class की property कहा जाता है।
- Class की property default रूप से private होती है, इसलिए इसे class के बाहर Access नहीं किया जाता है।
- Class की property को class से बाहर Access करने के लिए Access Modifiers का use किया जाता है।
- 3 Access Modifiers - (i) private  
(ii) public  
(iii) protected

- class की private property को class से बाहर Access नहीं किया जा सकता है।
- class की public property को पुरे प्रोग्राम में Access किया जा सकता है।
- class की protected property को उस class से बदले वाली child class में access किया जा सकता है।

Ex-

```
class ABC
{
    Member 1; //private
    public:
        member 2; //public
    protected:
        member 3; //protected
}; → class को हमेशा Semicolon से close किया जाता है।
```

Access Modifiers के आज Colon लगाया जाता है।

## B. Object ⇒

- एक class का instance होता है, जिसके द्वारा class की सारी property को class के बाहर Access किया जाता है।
  - Real world entity.
  - Physical entity of class.
  - Class type का variable.
- Ex ⇒      class-name obj-name;  
                  Abc obj1;
- object के द्वारा class की property को class के बाहर Access करने के लिए dot (.) operator use किया जाता है।

Ex=

```
class ABC //ABC class
{
    public:
        int a = 5; //public property
};

int main()
{
    ABC obj; //object of class ABC = obj
    cout << obj.a; //Accessing a of class ABC
    return 0;
}
```

O/P=5

### C. Data Encapsulation ⇒

→ Wrapping up of Data in a single entity.

→ इसे Data Hiding भी कहते हैं।

Ex ⇒

Class ABC

Pack  
wrapped [ {  
    int a=5;  
    void add(int b,int c); } ] by default private property i.e. Data is hidden from all.

## D. Data Abstraction ⇒

→ Hiding of all the processes from the user except what he needs.

Ex⇒

```
class ABC  
{  
    int a;  
    float b;  
};
```

Encapsulation

```
class ABC  
{  
    int a;  
    public:  
        float b;  
};
```

User 1  
Abstraction

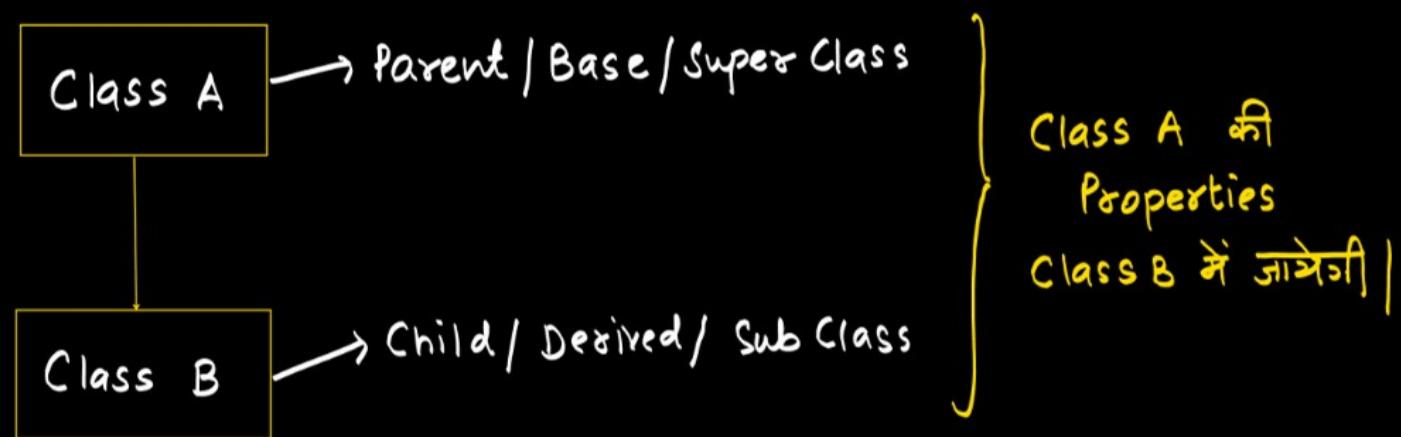
```
Class ABC  
{  
    float b;  
    public:  
        int a;  
};
```

User 2  
Abstraction

## E. Inheritance

→ यदि किसी एक class की properties को दूसरी class में Access किया जाते हों तो इसे Inheritance कहते हैं।

Ex :-

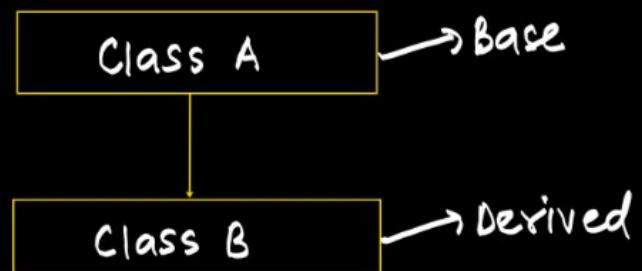


## \* Types of Inheritance →

- (i) Single Level
- (ii) Multi Level
- (iii) Multiple Level
- (iv) Hierarchical
- (v) Hybrid

(i) Single Level  $\Rightarrow$

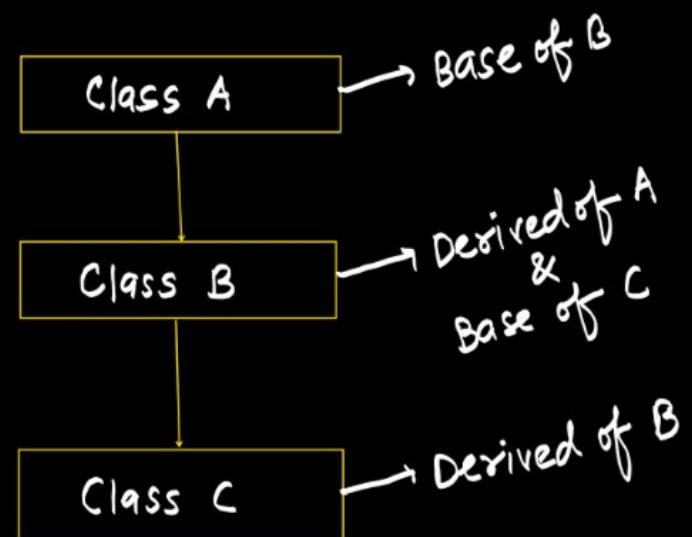
$\rightarrow$  Single Base & Single Derived Class.



(ii) Multi Level  $\Rightarrow$

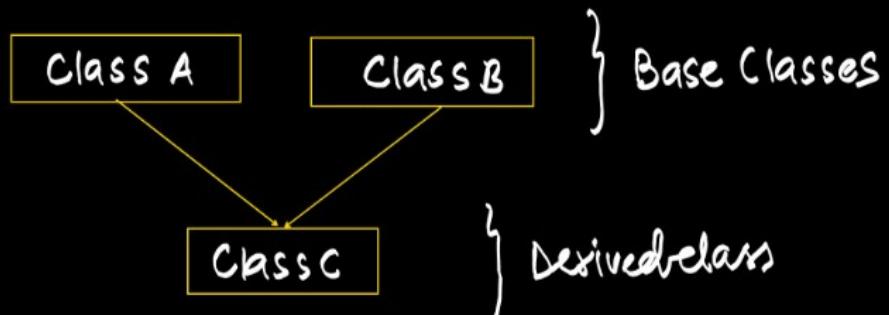
$\rightarrow$  इसमें भी एक ही Base नहीं। एक ही derived class होती है, परन्तु derived class से एक और derived class बना ली जाती है।

$\rightarrow$  पहली derived class दुसरी derived class के लिए Base class का काम करती है।



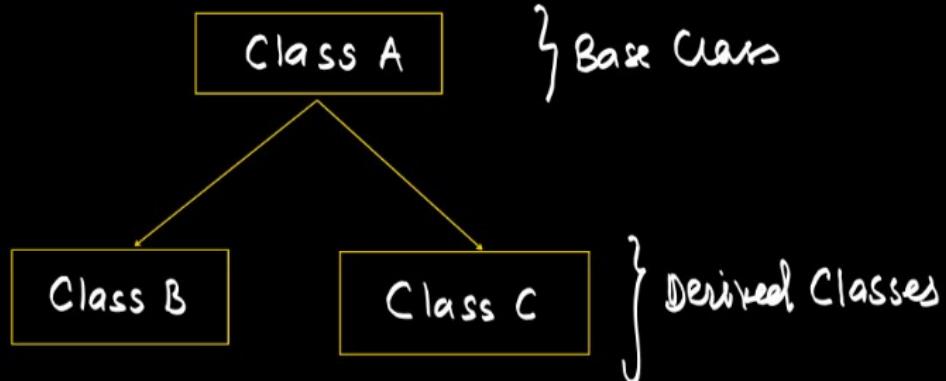
(iii) Multiple level ⇒

→ एक से अधिक Base classes & केवल एक derived classes.



(iv) Hierarchical ⇒

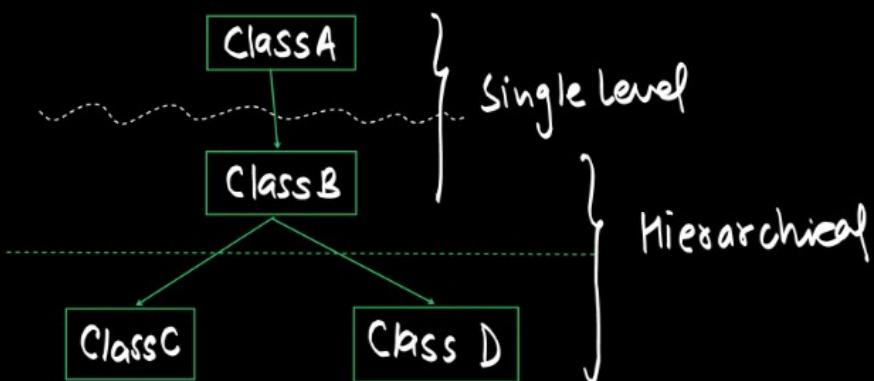
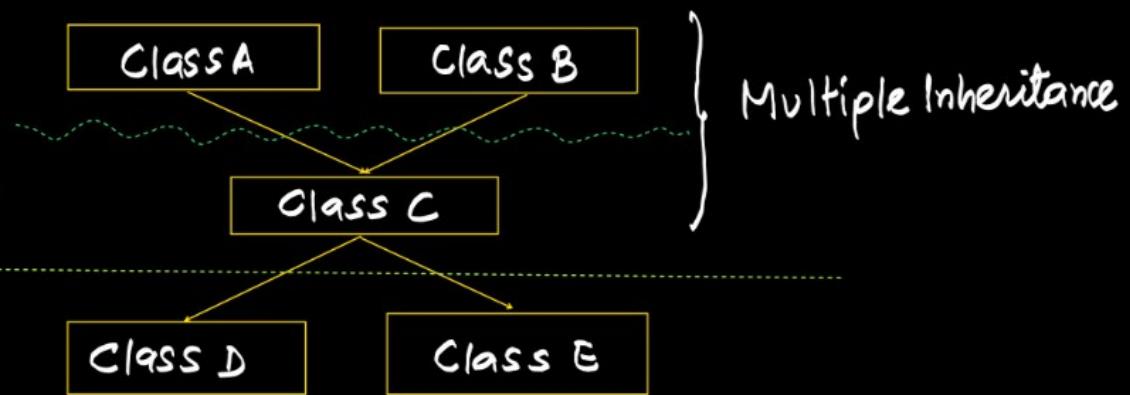
→ एक Base class के एक से अधिक derived classes.



## (V) Hybrid Inheritance $\Rightarrow$

→ 2 अलग-<sup>2</sup> प्रकार की Inheritances का Combination.

Hierarchical  
Inheritance



## (F) Polymorphism ⇒

- Poly = Many
- Morphism = रूप / Forms
- किसी entity की एक से अधिक Forms में use में लेना Polymorphism कहलाता है।
- इस Procedure में किसी function की अलग-2 Arguments के Type पर Arguments की संख्या के आधार पर use में लिया जाता है।

Ex ⇒

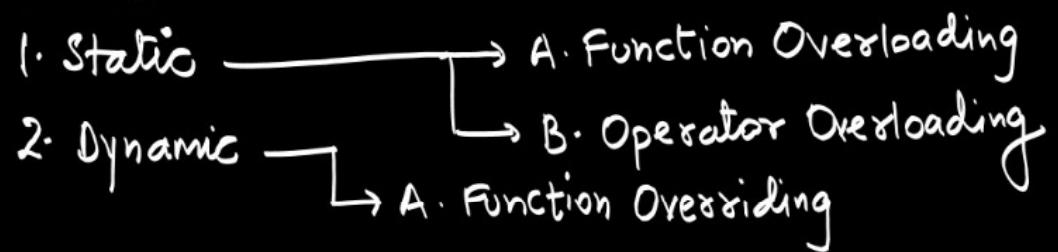
```
void add (int a, int b);  
void add (int a, float b);  
void add (int a, int b, int c);
```

Type of Arguments different  
Number of Arguments different

\* 2 conditions ⇒

1. Either number of arguments must be different.
2. Or Type of Arguments must be different.

\* Types of Polymorphism ⇒



## (G) Dynamic Binding ⇒

→ Program की सभी properties को Library files के साथ Bind करता।

→ 2 त्रिकार

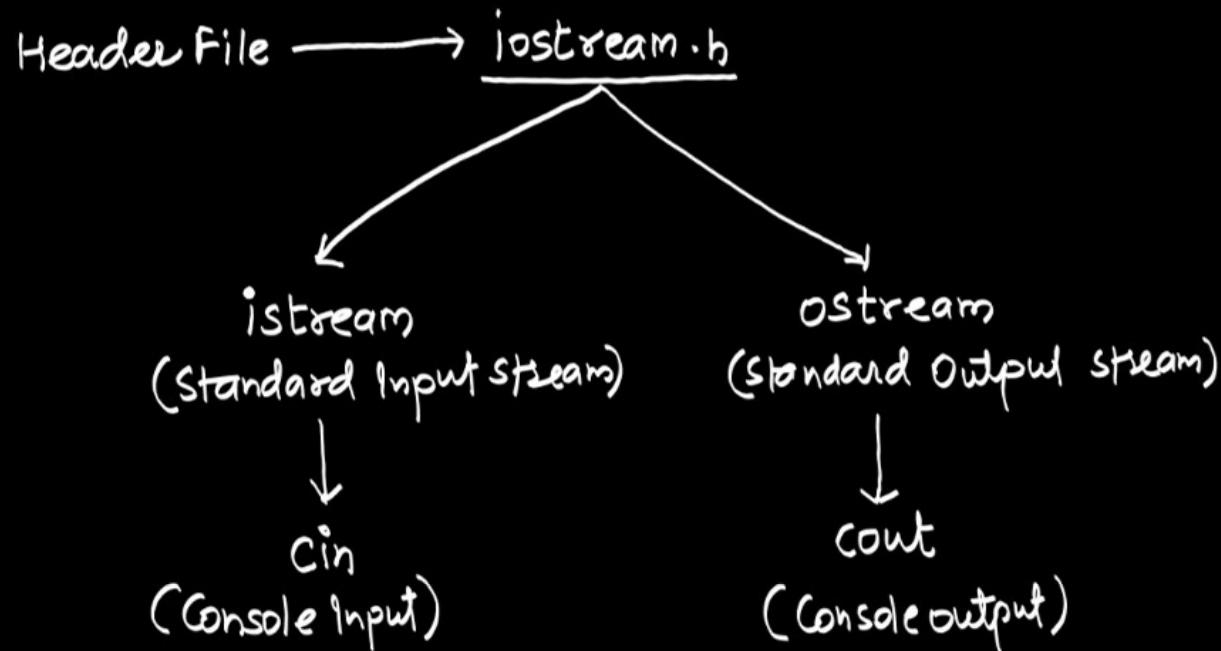
A. Static / Early / Compile Time Binding (To check Errors)

B. Dynamic / Late / Run Time Binding (To Display Output)

## (H) Message Passing ⇒

→ किसी भी class के objects unlimited होते हैं, इन objects के बीच data communication/transmission message passing कहलाता है।

## \* Input - Output Statement ⇒



## \* `Cin (console input) ⇒`

- Standard input stream का object
- Variables की values देता है।
- इसका use करके multiple variables की भी values के सकते हैं।
- इसके साथ `>>` (Extraction Operator) का use होता है।

Ex ⇒

`Cin >> a;` ✓

`Cin > a, b, c, d;` ✗

`Cin >> a >> b >> c >> d;` ✓

## \* cout (Console Output) →

→ standard output stream का object.

→ इसके साथ << (Insertion Operator) का use होता है।

→ इसके द्वारा किसी statement को output में display करने के लिए उसे double inverted comma (" ") के बीच लिखा जाता है और यह किसी variable की value output में display करने के लिए उसे बिना inverted comma के लिखा जाता है।

Ex ⇒      int a=5;  
cout<<" a";    ⇒ O/P = a  
cout<<a;       ⇒ O/P = 5

## \* Additional Operators in C++ $\Rightarrow$

### A. Scope Resolution Operator (`::`) $\Rightarrow$

→ किसी Variable के global version को access करता।

→ This can't be overloaded.

```
int a=10;
void main()
{
    int a=20;
    cout<<a;
    {
        int a=30;
        cout<<a;
        cout<<::a;
    }
    getch();
}
```

O/p = 20 30 10

## \* Dynamic Memory Allocation $\Rightarrow$

- A. new = Allocates Memory
- B. delete = Deallocates Memory

int \*p;

int n = 10;

p = new int;  $\rightarrow$  2 Byte का Block p को Allocate कर देगा।

p = new int(n);  $\rightarrow$  n element के लिए p को integer size का Block allocate करेगा।

delete p;

## ★ Constructor ⇒

- ऐसा Special Member function जिसका नाम class के नाम के समान होता है।
- मध्य class का object बनते ही auto initialize हो जाता है।
- No return type.
- इसका नाम class के नाम समान होने के कारण मध्य class की private property भी access कर सकता है।
- We can't inherit constructor.
- Constructor का use करते समय Minimum = 1 Argument का होगा जरूरी है।
- We can overload constructor.
- 3 Types ⇒
  - A. Parameterised Constructor
  - B. Default Argument Constructor
  - C. Copy Constructor

## A. Parameterised Constructors $\Rightarrow$

$\rightarrow$  ऐसा Constructor जो कुछ न कुछ Arguments पेकर ही declare होता है।

Ex  $\Rightarrow$

<pre>class PC {     int a; public:     PC(int b)     {         a=b;         cout &lt;&lt; a;     } }</pre> <p>Parameterised Constructor</p>	<pre>int main() {     PC obj(5); // Constructor initialization done <u>b=5</u>     return 0; }</pre> <p>O/P = 5</p>
---	---

## (B) Default Argument Constructors

→ ऐसा constructor जो declare होने समय variables की values लेकर ही declare होता है।

जैसे-      PC (int a); → Parameterised  
                  PC (int a=5); → Default Argument

Pgm :-

```
class DAC
{
    int a, b;
public:
    DAC(int c=5, int d=10)
    {
        a=c; b=d;
        cout<<a<<b;
    }
}
```

```
int main()
{
    DAC obj;
    return 0;
}
O/P = 5 10
```

## C. Copy Constructor

→ ऐसा constructor जो खुद की class के object को लेकर declare होता है।

Ex ⇒

```
class CC
{
    int a, b;
public:
    CC(int x)
    {
        a = x;
        cout << a;
    }
    CC(int y, int z)
    {
        a = y;
        b = z;
        cout << a << b;
    }
};
```

```
int main()
{
    CC obj(5);
    CC obj2(10, obj);
    return 0;
}
o/p = 5 105
```

## \* Constructor Overloading ↴

→ जब किसी class में multiple constructors same name से use होते जा रहे हों, जिनमें प्रा हो Number of Arguments different हो या Number of arguments same होने पर Type of arguments different हो, तो इसे Constructor Overloading कहते हैं।

Ex⇒ Class CO

```
int a, b, c;
public:
CO(int x)
{
    a=x;
    cout<<a*a;
}
```

CO (int y, int z)

```
{
    b=y;
    c=z;
    cout<<b*c;
}
```

int main()

```
{
    CO obj1(5);
    CO obj2(10, 2);
    return 0;
}
```

O/P = 25 20

## \* Destructor →

→ Delete Constructor

→ इसका नाम वी Class के नाम के समान होता है, लेकिन इसमें Tild (~) Symbol का use किया जाता है।

→ पर्द कोई भी Argument नहीं लेता है, इसलिए overload नहीं किया जा सकता है।

→ we can't inherit it.

Ex ⇒ class DES

```
{ int a;  
public:  
DES(int x=5)  
{ a=x;  
cout<<a;  
}
```

~DES();

```
};  
int main()  
{  
DES obj;  
return 0;  
}
```

O/P = 5

## ★ Inheritance ⇒

→ किसी एक class की properties का use कर दुसरी class बनाना या किसी एक class की properties का प्रयोग दुसरी class में लेना Inheritance कहलाता है।

## → 5 Types ⇒

- A. Single Level
- B. Multi Level
- C. Multiple
- D. Hierarchical
- E. Hybrid

## A. Single Level $\Rightarrow$

→ One Base & One Derived

Ex  $\Rightarrow$  Class A

```
{  
public:  
int x=5;  
};
```

Class B : public A

```
{  
public:  
cout << x;  
};
```

{  
public:  
void display()  
{  
cout << x;  
};  
};

```
int main()  
{  
B obj1;  
cout << obj1.x; ✓  
cout << obj1.display(); ✓  
return 0;
```

O/P = 5

## B. Multilevel $\Rightarrow$

→ एक Base तथा एक derived व इस derived के एक और derived class multilevel inheritance कहलाता है।

Ex =

```
class A
{
public:
    int x=5; ✓
};
```

```
Class B : public A
{
public:
    void display()
    {
        cout<<x;
    }
};
```

```
Class C : public B
```

```
{
};

int main()
{
    C obj;
    obj.display();
    return 0;
}
```

O/p = x: 5

## ★ Inheritance ⇒

→ किसी एक class की properties का use कर दुसरी class बनाना या किसी एक class की properties का प्रयोग दुसरी class में लेना Inheritance कहलाता है।

## → 5 Types ⇒

- A. Single Level
- B. MultiLevel
- C. Multiple
- D. Hierarchical
- E. Hybrid

## A. Single Level $\Rightarrow$

→ One Base & One Derived

Ex  $\Rightarrow$  Class A

```
{  
public:  
    int x=5;  
};
```

Class B : public A

```
{  
public:  
    cout << x;  
};
```

```
{  
public:  
    void display()  
    {  
        cout << x;  
    }  
};
```

```
int main()  
{  
    B obj1;  
    cout << obj1.x; ✓  
    cout << obj1.display(); ✓  
    return 0;  
}
```

O/P = 5

## B. Multilevel $\Rightarrow$

→ एक Base तथा एक derived वर स्थ derived के एक और derived class multilevel inheritance कहलाता है।

Ex =

```
class A
{
public:
    int x=5; ✓
};

class B : public A
{
public:
    void display()
    {
        cout<<x;
    }
};
```

```
class C : public B
{
};

int main()
{
    C obj;
    obj.display();
    return 0;
}
```

O/p = x: 5

### C. Multiple Inheritance $\Rightarrow$

→ इस Inheritance में 2 या 2 से अधिक Base classes का use करके एक ही derived class बनाई जाती है।

<u>Ex <math>\Rightarrow</math></u>	class A { public: int a=5; };  Class B { public: int b=10; };	Class C : public A, public B { public: void display() { int c=a+b; cout << c; } };	int main() { C obj; obj.display(); return 0; }  <u>O/P = 15</u>
------------------------------------	---	--	--

## D. Hierarchical Inheritance $\Rightarrow$

→ एक Base Class से Multiple derived classes बनाते हैं।

Ex  $\Rightarrow$  Class A

```
{ public:  
    int x=5, y=10;  
};
```

Class B : public A

```
{ public:  
    void add()  
    {  
        cout << x+y;  
    }  
};
```

Class C : public A

```
{ public:  
    void multiply()  
    {  
        cout << x*y;  
    }  
};
```

int main()

```
{  
    B obj1;  
    C obj2;  
    obj1.add();  
    obj2.multiply();  
    return 0;  
}
```



$$\frac{0/p}{5} = \underline{15}$$

$$50$$

## E. Hybrid Inheritance $\Rightarrow$

→ 2 अलग-<sup>2</sup> प्रकार की inheritance को एक राश use में ले जा।

Ex  $\Rightarrow$

```
class A
{
public:
    int x=5, y=10;
};
```

```
class B : public A
{
public:
    void add()
    {
        cout << x+y;
    }
};
```

```
class C : public A
```

```
{
public:
    void mul()
    {
        cout << 2*x*y;
    }
};
```

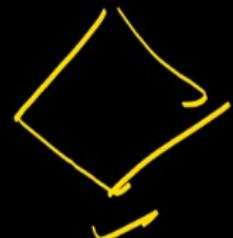
```
class D : public B, public C
```

```
{
```

```
}
```

```
int main()
{
    D obj;
    obj.add();
    obj.mul();
    return 0;
}
```

O/P = 15  
50



## ★ Function overloading ⇒

→ उब किसी Program में किसी एक function को बार-<sup>2</sup> अलग-<sup>2</sup> Arguments के साथ प्रा भवग-<sup>2</sup> Data Types वाले Arguments के साथ use के लिया जाते हों इसे Function Overloading कहते हैं।

Ex ⇒

```
class A
{
public:
    int add(int 5a, int 6b)
    {
        cout << a+b;
    }
    int add(int c, int d, int e)
    {
        cout << c+d+e;
    }
}.
```

```
int main()
{
    A obj;
    obj.add(5,6);
    obj.add(3,4,5);
    return 0;
}
```

O/p = 11 12

## \* Function Overriding →

- मानि कि सी एक function को दी अलग-2 classes में प्रयोग में लिया जावे, तो इसे function overriding कहा जाता है।
- इस Method में मानि Base class का function derived class में प्रयोग में लिया जावे, तो पहले derived class वाली values को display करता है।

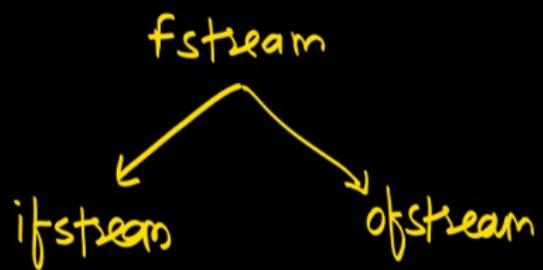
Ex ⇒ class BASE  
{  
public:  
void display()  
{  
cout << "Hi";  
}  
};

class Derived : public BASE  
{  
public:  
void display()  
{  
cout << "Hello";  
}  
};

int main()  
{  
Derived obj;  
obj.display();  
return 0;  
}  
O/p = Hello

## ★ File handling in C++ ⇒

- Collection of Data की जब एक नाम देकर storage पर save किया जाये, तो इसे file कहते हैं।
- प्रत्येक file को user द्वारा एक Name दिया जाता है तथा इसके Type के मनुसार इसका Extension System Generated होता है।
- File handling के लिए C++ में fstream, ifstream (Input FileStream) & ofstream (Output FileStream) तीन classes का use किया जाता है।



\* Stream  $\Rightarrow$  Sequence of Bytes

Sequence of Bits = String

\* Opening a file  $\Rightarrow$

1. using open() function
2. using constructor

## \* File opening Modes →

→ scope Resolution Operator का use.

- A. ios::in → open file to Read
- B. ios::out → open file to write
- C. ios::app → Append mode  
File के Data के End में हमारा Data Add करता।  
Same as ios::out
- D. ios::trunc → Truncate Mode  
Delete all data of file
- E. ios::binary → Open file in binary Mode

\* एक file को एक से अधिक Modes में open किया जा सकता है, इसके लिए Bitwise OR (|) operator use करें।  
जैसे - ios::out | ios::in

## \* Functions of file Handling =>

1. fopen() => file को open करना।
2. fputc() => file में एक character लिखना।
3. fgetc() => file से एक character read करना।
4. fclose() => file को close करना।
5. fseek() => file में दी गई जगह पर file pointer को set करना।
6. fputw() => file में एक Integer write करना।
7. fgetw() => file से एक Integer Read करना।
8. ftell() => file pointer की current position को return करना।
9. rewind() => file pointer को file की beginning में set करना।

\* fopen() function की use में लैने वाले निचे modes का use किया जाए है -

1. r = Read mode

2. w = Write mode

3. w+ = Both read & write modes

4. a = Append mode

5. r+ = Both read & write modes

6. a+ :            " "

7. rb = Binary file in read mode

8. wb = Binary file in write mode

9. ab = Binary file in Append mode

10. rb+ / wb+ / ab+ = Binary file in Both read & write mode.

## \* Friend function =>

- इस function का use करके किसी class की private & protected property को class के बाहर Access किया जा सकता है।
- "friend" keyword is used.
- Syntax =  
    friend return-type function-name (arguments);
- इसे class की Boundary में declare किया जाता है।

## \* Characteristics of friend function \*

1. friend function को class के object द्वारा call नहीं किया जा सकता है।
2. friend function उस class के scope में नहीं होता है, जिसमें उसे declare किया जाता है।
3. इस सामान्य function की तरह ही call किया जाता है।
4. यह Member Names को direct call नहीं कर सकता है और इसे Member Name के साथ object name और dot operator का use करना पड़ता है।
5. इसे class में private या public किसी भी section में declare किया जा सकता है।

## \* Disadvantages of friend function ⇒

1. यदि किसी class में अधिक friend function हो, तो Encapsulation की value की कम करता है।

Ex ⇒

```
#include <iostream.h>
using namespace std;
class ATM
{
private:
    int balance; 1000
public:
    ATM (int b) //constructor
    {
        1000 balance = b;
    }
    friend void chkbalance (ATM a); 1000
};
```

```
void chkbalance (ATM a) 1000
{
    cout << "Balance = " << a.balance;
}
int main()
{
    ATM user (1000);
    chkbalance (user);
    return 0;
}
O/P = Balance = 1000
```

## ★ Exception Handling ⇒

\* Try ⇒ प्रथम एक Block होता है, जिसमें ऐसे Code की लिखवा जाता है जिससे exception generate होने की संभावना होती है।  
⇒ प्रथम प्रोग्राम में कोई logical code ही नहीं होते लिए कि उसके execution से exception generate हो सकती है, तो उसे try Block में लिखवा जाता है।

Syntax ⇒

```
try
{
    Code;
}
```

- \* throw ⇒ यह keyword try block में use में लिया जाता है।
  - ⇒ इसके द्वारा try block से exception को throw किया जाता है।
  - ⇒ इस Exception को catch block catch करता है।
  - ⇒ इसके द्वारा किसी Variable को भी catch block को throw किया जा सकता है, जिसका use catch द्वारा Exception handling में किया जाता है।

\* catch ⇒ इसी Block में exception handle होता है।

⇒ इस Block के द्वारा exception माने पर कोई message display कर सकते हैं तो  
दुसरा code execute करवाया जा सकता है।

⇒ पहले Block "try block" के होशा बाद में लिखा जाता है।

⇒ multiple exceptions की handle करने के लिए multiple catch blocks का  
use किया जा सकता है।

## \* Process of exception handling ⇒

1. Detection of Exception
2. Throw keyword द्वारा exception throw
3. Catch द्वारा exception को catch
4. catch block में exception handle .

Ex

```
#include<iostream.h>
using namespace std;
int main()
{
    int num=10;
    int result;
    try
    {
        result = num/0;
        throw 0;
    }
```

```
    catch ( int e)
    {
        cout << "divided by " << e;
    }
    return 0;
}
```

O/p = divided by 0

## \* File Handling in C ⇒

- File is a container which stores data.
- File handling एक प्रोसेस है, जो Data की file में store करने के लिए program का use करती है।
- 6 operations ⇒
  - A. Create
  - B. Open
  - C. Read
  - D. Write
  - E. Delete
  - F. Close

## ★ Functions of File Handling ⇒

1. fopen() ⇒ file को open करना।
2. fprintf() ⇒ file में Data को write करना।
3. fscanf() ⇒ file से Data की Read करना।
4. fputc() ⇒ file में एक character को write करना।
5. fgetc() ⇒ file से एक character की read करना।
6. fclose() ⇒ file की close करना।
7. fseek() ⇒ दी गई location पर file pointer को set करना।
8. rewind() ⇒ file pointer को file की शुरूआत में set करता है।

9. `fputw()` ⇒ file में integer को write करता।
10. `fgetw()` ⇒ file से integer को read करता।
11. `ftell()` ⇒ file pointer की current position को display करता है।

\* file को Open करना ⇒

FILE \*fopen( const char\* filename, const char\* mode);

→ fopen() function 2 parameters A. filename & B. mode होता है।

\* File Modes ⇒

r ⇒ Read Mode

w ⇒ Write Mode

a ⇒ Append Mode

r+ ⇒ Read & Write Both Modes

w+ ⇒ Read & write Both Modes

a+ ⇒ Read & write Both Modes

rb ⇒ Binary file in read mode

wb ⇒ Binary file in write mode

ab ⇒ Binary file in Append Mode

rb+ ⇒ Binary file in read & write Modes

wb+ ⇒ " "

ab+ ⇒ " "

## \* Structure & Union =

Structure	Union
<ol style="list-style-type: none"><li>1. struct keyword to declare structure.</li><li>2. Structure की जब declare किया जाता है, तो Compiler प्रत्येक variable की मेमोरी Allocate करता है, इसलिए structure की Total Data Members की size के बराबर होती है।</li><li>3. इसमें प्रत्येक data member को अलग-<sup>2</sup> size Allocate की जाती है, इसलिए जब sizeof() का use करके size पता की जाती है, तो सबसे बड़े Data Element के बराबर Storage सभी Elements को Available करवाया जाता है।</li></ol>	<ol style="list-style-type: none"><li>1. union keyword to declare union.</li><li>2. Union की जब declare किया जाता है, तो Compiler सबसे बड़े Data Member की size के बराबर Memory Allocate करता है।</li><li>3. इसमें सबसे बड़े Data Element की size के बराबर ही Allocation होता है।</li></ol>