

Object Oriented programming

C++
Java, Pascal } OOL
Ada, Eiffel

object based language
↳ Fortran

pure O.O. language

↓
Python

→ Developer of C++ is Bjarne Stroustrup
in AT&T bell labs.

OOP ⇒

(i) Object :- Object are real time entities composed of data & functions that operate on it.

(ii) Class :- It is a logical construct which is used to define an object. It can be seen as a blueprint of an object.

(Concepts of Paradigms :-)

(iii) Encapsulation :- It is the wrapping of a data to protect from outside uses.

(iv) Information / Data Hiding

(v) Abstraction :- It is the process of hiding the essential working details.

(5.) Inheritance :- It is the property through which one object is acquired the features of existing object.

Paradigm $\Rightarrow$

Paradigm is a concepts to solve the problems.

Types of Paradigm :-

(i) Imperative $\Rightarrow$

- Structural
- Procedural

$\rightarrow$ This paradigm specify the problem and the step to solve the problem.

(ii) Declarative $\Rightarrow$ This paradigm focus on what the result is without concerning the process.

- Logic based
- Rule based

~~(iii)~~ Object oriented paradigm $\Rightarrow$ It is a programming methodology which is based on the concept of the object and consist of data and operation which will operate on the data.

Features of OOP:—

- (i) **Classes:—** It is a logical construct which act as a user define data type for an object.
- (ii) **Object:—** It is a real world entity which has its own data & behaviour.
- (iii) **Information hiding:—** It's hiding of information.

OOP of paradigms

- (i) **Encapsulation:—** It is the wrapping of data and its behaviour into a single unit to protect it from outside interference.
- (ii) **Abstraction:—** It is the process of hiding the details and features of implementation without affecting uses.
- (iii) **Polymorphism:—** It is the mechanism in which an entity is able to perform multiple behaviour depending upon information.
- (iv) **Inheritance:—** It is the process in which one object acquires the properties of existing object.

(v) Message passing :— It is the process in which object communicate with each other.

(vi) Late binding :— It is the process of binding a function to its call which is only known at run time.

(vii) Persistence :—

This features allows an object to outlive its program execution.

(viii) Genericity :—

It is a process in which object components can be reused for multiple information.

General Structure of C++ program $\Rightarrow$

Header files

macro definitions

Comments

user function prototype

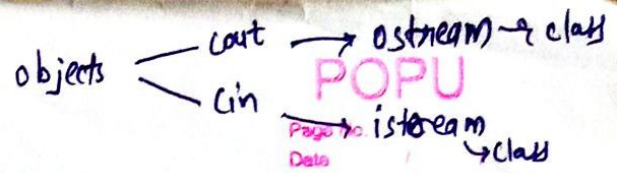
global variables

class declarations paradigms

L user function

(inside / outside)

main()



```

#include <iostream.h>
int main()
{ cin >> a
  cout << "Hello world";
}

```

extraction
System out object
insertion operator

```

int a; float b;
cin >> a >> b;
cout << "A*2=" << a*2;
cout << a*2;
}

```

25 10
Result
A*2=50
Result
50

Q. WAP to input age of a person & determine if he is child or teenager or youngster or old age.

0-10

11-22

22-35

35+

Ans- #include <iostream.h>

#include <conio.h>

void main()

{

int a;

~~cin >> a;~~ cout << "Enter the age";

cin >> a;

if (a <= 10)

{

cout << "He is child";

}

elseif (a > 10 && a <= 22)

{

cout << "He is teenager";

}


```
elseif (a > 22 && a <= 35)
{
```

```
    cout << "He is young";
```

```
}
```

```
else
```

```
{
```

```
    cout << "He is old age";
```

```
}
```

```
}
```

Q. WAP to convert temp in fahrenheit to °C.

Ans- #include <iostream.h>

#include <conio.h>

int main()

{

~~int~~ float f, c;

cin >> f;

c = (f - 32) / (float) 1.8;

cout << "value of fahrenheit in °C is";

cout << c;

}

Q. WAP to add no. until 0 is pressed.

Ans- #include <iostream.h>

#include <conio.h>

int main()

{

int n; s = 0;

cout << "Enter number";

cin >> n;


```

while (n != 0)
{
    s = s + n;
    cout << "sum of no. is ";
    cout << s;
    cin >> n;
}
return 0;
}

```

Q. WAP to find a factorial no.

Solⁿ -

```

#include <iostream.h>
int main()
{
    int n, i, s=1;
    cout << "Enter number ";
    cin >> n;
    for (i=1; i<=n; i++)
    {
        s = s * i;
    }
    cout << "factorial is " << s;
    return 0;
}

```

Class & Object $\Rightarrow$

Access specifiers:— These keywords specify member accessibility of the class.

public
private
protected

- (i) public :- Members are accessible outside the class.
- (ii) private :- Members are accessible within the class.
- (iii) protected :- These members are accessible only when its class is inheritant.

→ Default access specifier is private.

eg- #include <iostream.h>

class tag

{

public:

void print()

{

cout << "Hello";

}

}

int main()

{

tag t1;

t1.print();

}

Q. WAP to perform arithmetic calculations.

Solⁿ #include <iostream.h>

class calc

{

int a, b;

public:

void inp()

{

cin >> a >> b;

}


```
void add()
{
    cout << a+b;
}
void sub()
{
    cout << a-b;
}
void multi()
{
    cout << a*b;
}
void div()
{
    cout << a/b;
}
void mod()
{
    cout << a%b;
}
}
int main()
{
    calc cl;
    cl.inp();
    cl.add();
    cl.sub();
    cl.multi();
    cl.div();
    cl.mod();
}
```


Q. WAP to input two no. and find which one is greater

Solⁿ

```
#include <iostream.h>
```

```
class big
```

```
{
```

```
    int a, b;
```

```
    void input()
```

```
{
```

```
        cin >> a >> b;
```

```
    int greater()
```

```
{
```

```
        if (a > b)
```

```
        { return a; }
```

```
    else
```

```
        { return b; }
```

```
    }
```

```
};
```

```
int main()
```

```
{
```

```
    big b1;
```

```
    b1.input();
```

```
    cout << b1.greater();
```

```
    return 0;
```

```
}
```

~~Declar~~ Defining function Outside the class ⇒
SRO [Scope regulation Operator] :-

operation which is used to It is an

- (i) Identify a global variable hidden under local variable of same name.

(ii) It is used to identify scope of the function corresponding to a class.

Syntax: —

returntype classname :: functionname (arguments)

eg-

```
#include <iostream.h>

class big
{
    int a, b;
    void input ();
    int greater ();
};

void big :: input ()
{
    cin >> a >> b;
}

int big :: greater ()
{
    if (a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}

int main ()
{
    big b1;
    b1.input ();
    cout << b1.greater ();
    return 0;
}
```

Q.

WAP to add and subtract two no.

Ans

Ans -

```
#include <iostream.h>

class operator
{
    int a, b;
public:
    void input();
    int add();
    int sub();
};

void operator::input()
{
    cin >> a >> b;
}

int operator::add()
{
    int c;
    c = a + b;
    return c;
}

int operator::sub()
{
    int c;
    c = a - b;
    return c;
}

int main()
{
    operator b1;
    b1.input();
    cout << b1.add();
    cout << b1.sub();
    return 0;
}
```

Most

```
void input(int, int);

void operator::input(int x, int y)
{
    a = x;
    b = y;
}

int main()
{
    int x, y;
    cin >> x >> y;
    c1.input(x, y);
    0%
    int main()
    {
        c1.input(5, 6);
    }
```

Q. WAP to calculate x^y

Ans -

```
#include <iostream.h>

class power
{
    int a, b;
public:
```



```

void inpt (int, int);
int pow (int, int);
};
void inpt power:: inpt (int x, int y)
{
    a = x;
    b = y; }
int power:: pow (int x, int y)
{
    if (y != 0)
    {
        return (x * pow(x, y-1));
    }
    else
    {
        return 1;
    }
}
int main ()
{
    power p1;
    int x, y;
    cout << "Enter no = ";
    cin >> x;
    cout << "Enter power = ";
    cin >> y;
    p1 = input (x, y);
    cout << p1 pow (x, y);
}

```

Structures in C++ ⇒

Structures in C++ are user defined datatype whose variable is able to store value of heterogeneous type. These structures are able to organised group of information together in a meaningful way.

In C++, these structures can be defined or associated with functions as well and can have specification as public or private by default it is public.

Syntax:

```
struct tag  
{
```

```
    type member;
```

```
    type member;
```

```
};
```

Q. WAP to input and display detail of a single student.

Ans - #include <iostream.h>

```
struct student  
{
```

```
    char name[10];
```

```
    int roll;
```

```
    void inpt();  
}
```

```
    cin >> name >> roll;
```

```
    void display();  
}
```

```
    cout << name << roll;
```

```
};
```

```
int main()
```

```
{
```

```
    s.inpt();
```

```
    s.display();
```

```
    return 0;
```

```
}
```

```
// }
```

```
int main()
```

```
{
```

```
    student *p = s;
```

```
    p->inpt();
```

```
    p->display();
```


Q. WAP to input name and marks of 5 subject for a student & calculate his ~~persen~~ %.

Ans- #include <iostream.h>

struct student

{ int per, total = 0;

char name[10];

~~int char~~ ~~sub[5];~~ int ~~sub~~ [5];

void inpt ()

{ int i;

for (i = 0; i < 5; i++)

{ cin >> name;

cin >> sub[i]; }

}

}

void display ()

{

for (i = 0; i < 5; i++)

{

cout << "total no. ";

total = total + sub[i];

}

per = (float) total / 5;

~~printf~~ cout << per; }

};

int main ()

{

s.inpt ();

s.display ();

return 0;

}

Unit-2

Reference variables :- These variables are those variables which provide an alternate name for an existing value variable. They are initialised when declared and refer to that initialised variable.

Syntax :-

eg - `datatype &variablename;`
`int &a;`
`float &b;`

eg - `int a, b;`
`a = 5;`
`int &p = a;`
`cout << p;`
`p = 7;`
`cout << p;` ~~`p = b;`~~

Q. WAP to swap two numbers in a function.

Ans - `#include <iostream.h>`

`void swap(int, int);`

`int main()`

`{`

`int x, y;`

`cin >> x >> y;`

`cout << x << y;`

`swap(x, y);`

`cout << x << y;`

`return 0;`

`}`

$\left\{ \begin{array}{l} 12, 15 \\ 12, 15 \end{array} \right\}$

// Call by value only
A copy value will pass in the function.


```
void swap (int a, int b) {
    int t;
    t = a;
    a = b;
    b = t;
}
```

→ In the above program, x & y are passed by value therefore their copies (not original) are sent to the swap function and changes, swapping is done on the copies. Therefore when x & y are printed the 2-time in the main function their original values are printed. But swapping is done only on the copied value so both cout gain the same result. To solve this problem if x & y are passed as references then any changes made to the references will be reflected in the original value as well.

DMA ⇒ (Dynamic memory allocation)

It is the mechanism in which memory is allotted to the object at run-time. This allows the allotted memory to reduce or increase as per requirement. DMA in C++ is implemented using NEW operator and DELETE operator that is NEW allocate space whereas DELETE free-up memory.

eg-

```
int main ()
{
    int *a;
    a = new int;
    *a = 5;
    cout << a;    delete (a);
}
```


Q. WAP to create a variable size array.

Solⁿ

```
#include <iostream.h>

int main()
{
    int *p;
    int n, i;
    cout << "enter size";
    cin >> n;
    p = new int;
    for (i=0; i<n; i++)
    {
        cin >> p[i];
    }
    delete (p);
}
```

Q. WAP to find largest value in an vector (v.s) array.

Solⁿ

```
#include <iostream.h>

int main()
{
    int *p;
    int n, i; int max=0;
    cout << "enter size";
    cin >> n;
    p = new int;
    for (i=0; i<n; i++)
    {
        cin >> p[i];
    }
}
```



```

for (i = 0; i < n; i++)
{
    for (j = 0; j < i; j++)
    {
        for (k = 0; k < i; k++)
        {
            if (max < p[i])
            {
                max = p[i];
            }
        }
    }
    cout << max;
    delete (p);
}

```

Inline function → These are the functions which are expanded to their actual code at the place of their call.

→ Many times time taken to execute a function is less than the time taken to switch b/w functions call. This extra time is known as overhead and can reduce efficiency of the program.

→ To eliminate the overhead a regular function can be declared inline, this causes that inline function to expand at the place where it is called thereby removing the switching time (Jumping b/w one function to other). Thus reducing overhead.

eg -

```

inline int sqn (int a)
{
    return a*a;
}
inline int cube (int x)
{
    return x*x*x;
}
int main()
{
    cout << sqn(3); // cout << a*a;
    cout << cube(4); // cout << x*x*x;
}

```

Functions that are defined inside the class are inline by default.

Advantages of inline function $\Rightarrow$

inline function is that it reduces overhead thus increasing efficiency. Advantage of

$\rightarrow$ And disadvantages is that multiple inline functions can unnecessarily increase the size of binary execution.

FUNCTION OVERLOADING:-

```

class Sum
{
public:
    int add (int a, int b);
    int add (int a, int b, int c);
}

```



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

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

```
int main()
{
    sum s1;
    cout << s1.add(5,6);
    cout << s1.add(2,3,4);
}
```

It is a concept of polymorphism in C++ and is a mechanism in which multiple functions with same name co-exist in a single program. It is possible only when such functions have different parameters or signatures.

Q. WAP to swap two no. & 2 characters

Solⁿ

```
class ABC
{
public:
    void swap (int a, int b) ;
    void swap (char a, char b) ;
};

void ABC:: swap (int a, int b)
{
    int t;
    t = a;
```



```

    a=b;
    b=t;
    cout << a << b;
}
void ABC :: swap (char a, char b)
{
    char k;
    k=a;
    a=b;
    b=k;
    cout << a << b;
}

```

```

int main ()
{
    ABC D1;
    D1.swap (5,6);
    D1.swap (9,16);
}

```

Function with default arguments:-

```

class sum
{
public:
    float add (int a=1, float b=0.0);
};

float sum::add (int a, float b)
{
    return a+b;
}

int main()
{

```



```
sum s1;
cout << s1.add(5); // 5.0
cout << s1.add(1); // 1.0
cout << s1.add(3, 2.5); // 5.5
}
```

V.V Imp

Constructor:—

These are special functions which are called automatically when an object is created. These functions do not have a return type & are generally used to initialize member variables of the class. They are of the following type:—

- (i) Dummy constructor
- (ii) Default "
- (iii) Parameterized "
- (iv) Copy (Reference) constructor

Constructors have the same name as that of class.

eg - class check

```
public:
```

```
check()
```

```
{ cout << "in a constructor"; }
```

```
void show()
```

```
{ cout << "in a function"; }
```

```
int main()
```

```
{ check c1;
```

```
c1.show();
```

```
}
```


Q. WAP to implement default constructor.
Solⁿ:-

```
class add
{
    int a, b;
public:
    add() // automatically call
    {
        a = 5;
        b = 2;
    }
    void sum()
    {
        cout << a + b;
    }
};

int main()
{
    add a1;
    a1.sum();
}
```

Q. WAP to initialise 5 no. using constructor and calculate their average.

```
Soln:-
class average
{
    int a, b, c, d, e;
public:
    average()
    {
        cin >> a >> b >> c >> d >> e;
    }
    void sum()
    {
        cout << (a + b + c + d + e) / 5;
    }
};
```



```

int main()
{
    average a1;
    a1.sum();
}

```

Q. WAP to find largest b/w 3 no.

Solⁿ:-

```

class find
{
    int a[3] a[3];
public:
    find() // default constructor
    { for(i=0; i<3; i++)
      { cin >> a[i]; cin >> a[i]; }
    }
    void show()
    {
        int i, max=0;
        for(i=0; i<3; i++)
        {
            if (max < a[i])
            {
                max = a[i];
            }
        }
        cout << max;
    }
};

int main()
{
    find a1;
    a1.show();
}

```


POPUP
Page No.:
Date: / /

Parameterised Constructor:- This type of constructor contains parameter as arguments and similar to regular functions it can also be overloaded.

Q. WAP to swap 2 no.
Soln:- class ex

```
{  
    int a, b;  
public: ex (int x, int y)  
    {
```

```
        a=x; b=y;
```

```
    }
```

```
    void swap ()
```

```
    {  
        int t;
```

```
        t = a;
```

```
        a = b;
```

```
        b = t;
```

```
        cout << a << b;
```

```
    }
```

```
};
```

```
int main ()
```

```
{
```

```
    ex int x, y;
```

```
    cin >> x >> y;
```

```
    ex e1 (x, y);
```

```
    e1.swap();
```

```
    ex e2 (5, 7);
```

```
    e2.swap;
```

```
}
```


Q. WAP which uses parameterised constructor to calculate factorial of a no.

Soln:-

class fac
{

int n;

• fac (int x)

n = x;

int factorial (int y)

n = y;

if (n == 0)

{

return n * factorial (n-1)

}

else

return 1;

}

}; int main()

{ fac f1(5);

f1.factorial(5);

}

class fac

{

int n;

public: fac (int x)

{

n = x;

void factorial ()

{

int i, s = 1;

for (i = 1; i <= n; i++)

{

s = s * i;

~~cout << s;~~

}

cout << s;

}

};

int main()

{

int x;

cin >> x;

fac f1(x);

f1.factorial();

return 0;

}

do not
use Recursion

Q. WAP to input² height ~~para~~ in feet and inches and add them.

```

Ans - #include <iostream.h>
class height
{
    int a, b;
public:
    height (int x, int y)
    {
        a = x; b = y;
    }
    height (int p, int q)
    {
a = p; b = q; height = add (height h)
    {
        height h;
        h.a = h.a a + h.a a;
        h.b = h.b b + h.b b;
        return h;
    }
    void show ()
    {
        cout << a << " " << b;
    }
};

```

```

int main ()
{
    height h1, h2, h3;
    h1.height (5, 2)
    h2.height (4, 10)
    h3 = h1.add (h2);
    cout << h3;
}

```

OR

```

class add
{
    int f1, f2, f3, in1, in2, in3;
public:
    add (int a, int b, int c, int d)
    {
        f1 = a; f2 = b; in1 = c; in2 = d;
    }
}

```



```

void sum ()
{
    f3 = f1 + f2;
    in3 = in1 + in2;
    if (in3 >= 12)
    {
        f3 = f3 + 1;
        in3 = in3 - 12;
    }
    cout << "feet : " << f3;
    cout << "In inch : " << in3;
}

int main ()
{
    add a1 (5, 4, 2, 10);
    add a1.sum ();
}

```

Copy Constructor $\Rightarrow$ It is a special member function which is used to initialize a new object from an existing object. It has the following roles:-

- (i) It copies an existing object to a new one.
- (ii) Object is copied when it is passed as an argument.
- (iii) It is called when a function returns an object.


```

eg- class check
    {
        int a, b;
    public:
        check (int x, int y)
        {
            a = x;
            b = y;
        }
        check (const check &c)
        {
            a = c.a;
            b = c.b;
        }
        void show ()
        {
            cout << a << b;
        }
    };

int main ()
{
    check c1 (5, 10);
    check c2 = c1; or check c2 (c1);
    c1.show ();
    c2.show ();
}
    
```

Concept of FRIEND :—

In C++, a class or a function can be declared as a friend of an existing class. This allows that

friend function on class to access even the private member of the existing class.
• Even though, a friend function on class can access private data but is not member of direct class.

Q. WAP to implement friend function.

Ans- class check
{
 int a, b;
public:
 check (int x, int y)
 {
 a = x; b = y;
 }
 friend void add (check c1);
};
void add (check c1)
{
 cout << c1.a + c1.b;
}
int main ()
{
 check c (5, 6);
 add (c);
}

Q. WAP to implement friend class.

Ans- #include <iostream.h>
class check
{
 int a, b;


```

public:
    check ( int x , int y )
    {
        a=x;
        b=y;
    }
friend add class add;
};

class add
{
public:
    void show ( check c1 )
    {
        cout << c1.a + c1.b;
    }
};

int main ()
{
    check c(5,5); ✓
    add a1;
    a1.show(); ✓
}
    
```

This pointer $\Rightarrow$ It is a pointer variable which has the address of the object in question. i.e. each member function of the current object has access to this pointer. Using this pointer one can access the data associated with the object & also can be used to refer the object with which the member function is associated. In short, it points to the current object.


```
class Hello
{
    int a;
    int b;
public:
    Hello (int a, int b)
    {
        this . a = a;      or  this → a = a ;
        this . b = b;      this → b = b ;
    }
};

int main ()
{
    Hello h1 (5, 10);
}
```

Destructor ⇒ Destructor is a member function which when called destroys the object & releases the memory. There can only be one destructor in a class.

★ Q. WAP to demonstrate working of a destructor.

Ans -

```
int obj = 0;
int alive = 0;
class Test
{
    Test ()
    {
        obj++;
        alive++;
    }
    ~ Test ()
```



```

    { alive --; }
};

```

```

int main()
{

```

```

    Test t1;
    Test t2;
}

```

```

    Test t3;
}

```

```

    Test t4;
}

```

```

}

```

```

}

```

O/p
obj = 4
alive = 2

Q. WAP to transfer money from account 1 (object) to account 2.

Ans - #include <iostream.h>

```

class acct
{

```

```

    int account;
    float balance;

```

```

public:
    acct ()
    {

```

```

        cout << "Enter account no." << endl;
        cin >> account;
        cout << "Enter balance " << endl;
        cin >> balance;
    }

```

```

    void transfer ( acct &a, float amount)
    {

```

```

        balance = balance - amount;
    }
}

```



```

        a.balance = a.balance + amount;
    }

    void show()
    {
        cout << "updated " ;
        cout << "account 1" ;
        cout << account << balance ;
        cout << "account 2" ;
        cout << account << balance ;
    }
};

int main()
{
    accmt a1, a2;
    int moneytransfer;
    cout << "Enter money transfer from account 1 to account 2:" ;
    cin >> moneytransfer;
    a2.transfer(a1, moneytransfer);
    a2.show();
}

```

Dynamic Constructor :-

Q. WAP to create a variable sized array.

Ans - #include <iostream.h>

class vector

{

int size;

int *ar;

public:

~~Unit - 2~~ ~~Inheritance~~

```
vector (int s) // constructor
{
```

```
    size = s;
```

```
    arr = new int[size];
}
```

```
void show ()
{
```

```
    int i;
```

```
    for (i=0; i<size; i++)
    {
```

```
        cin >> arr[i];
```

```
        cout << arr[i];
```

```
    }
```

```
    }
```

```
int
```

```
main ()
```

```
    int s;
```

```
    cout << "Enter size";
```

```
    cin >> s;
```

```
    vector v1(s);
```

```
    v1.show();
```

```
    return 0;
```

```
}
```

```
vector ()
```

```
{
```

```
    delete [] arr;
```

```
}
```

When size of an object is determined dynamically when using new operator in the constructor such constructor is known as dynamic constructor.

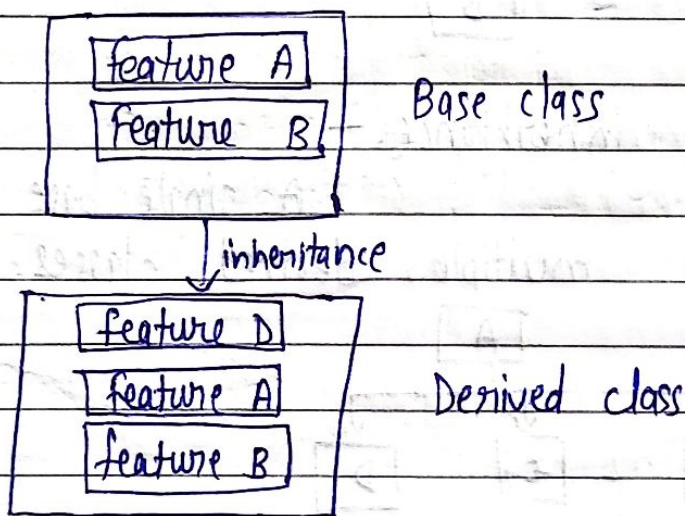
Unit-3

Inheritance

Inheritance ⇒

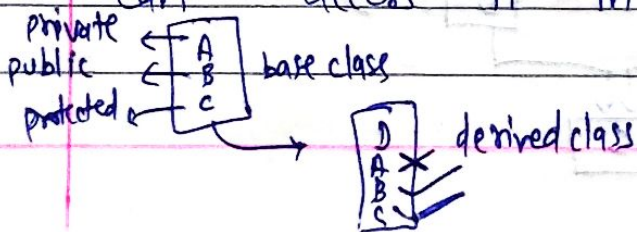
The property of inheritance allows one object to acquire features of some other object. It is basically a technique in which new classes are generated from existing classes. These new classes are generally specialised classes while the older ones are general classes.

The new classes are known as derived class. And the existing classes are known as base classes. These derived classes inherit features of existing class and all have their own exclusive features.



→ Protected access specifier :-

This specifier has the access limit of a private variable but a class which is inheriting the base class protected member can access it in that ^{same} ~~very~~ class only.



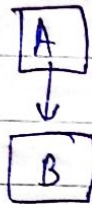
main()
 2 derived d1;
 d1.C X d1.A X
 d1.B ✓

Types of inheritance $\Rightarrow$

- (i) Single inheritance
- (ii) Hierarchical inheritance
- (iii) Multiple inheritance
- (iv) Multilevel inheritance
- (v) Hybrid inheritance
- (vi) Multi path inheritance

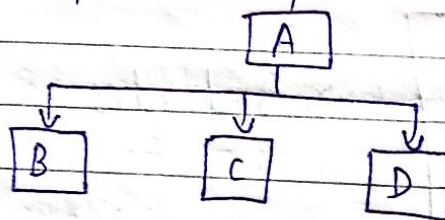
(i) Single inheritance:—

In this type of inheritance a single base class is directly inherited by a single derived class.



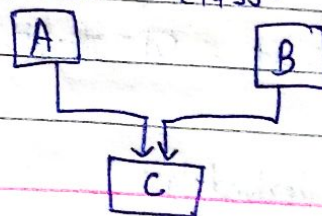
(ii) Hierarchical inheritance:—

A single base class is inherited by multiple derived classes.



(iii) Multiple inheritance:—

This inheritance involves many base classes to be inherited by a single derived class.

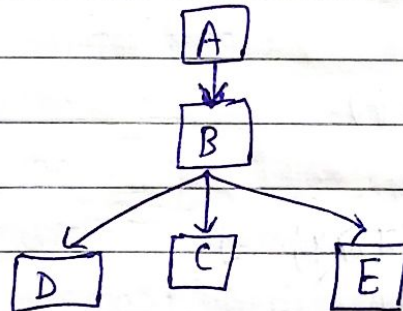


(iv) Multilevel inheritance :- This type of inheritance occurs when a base class is inherited by a derived class and this derived class behaves as base class to some other derived class.

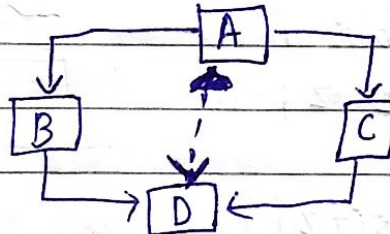


(v) Hybrid inheritance :-

This inheritance occurs when there are more than one form of inheritance



(vi) Multipath inheritance :-



When there are multiple ways for a derived class to access base class such inheritance is multipath.

ii) Single inheritance: -

Syntax: -

	class	derived :	visibility	base class
	{			
	}			
	;			
<u>Base class</u>		<u>visibility</u>		
public	public	private		
protected	protected	private		
private	not	not		

Q. WAP to implement single inheritance

Ans: -

class A

{ int x;

public:

int y;

void setxy()

{

cin >> x >> y;

}

int getx()

{

return x;

}

};

class B: public A

{

int z;

public;

void setz()

{

cin >> z;

}

void show()

{

setxy();

cout << getx() << y << z;

};


```

int main ()
{
    A = a1;
    B = b1;
a1.x; b1.x; a1.setX(); b1.setX();
a1.z; b1.z; a1.setZ(); b1.setZ();
    b1.show();
}

```

Q. WAP to create a class student specifying name & roll no. create function to input & print them. Create a class marks which inputs percentage of the student & prints student name, roll & percentage.

Ans-

```

class student
{
protected:
    char name[20];
    int roll;
public:
    void inpt ()
    {
        cout << "Enter name" << endl;
        cin >> name;
        cout << "Enter roll" << endl;
        cin >> roll;
    }
};

class marks : public student
{
    int percentage;
}

```



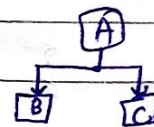
```

public:
    void per()
    {
        cout << "Enter percentage" << endl;
        cin >> percentage;
    }

    void show()
    {
        cout << name << endl;
        cout << roll << endl;
        cout << percentage << endl;
    }
};

int main()
{
    marks m;
    m.inpt();
    m.per();
    m.show();
}
    
```

(ii) Hierarchical inheritance:—



eg- class A

```

{
    protected:
        int a;
    public:
        void setA()
        {
            a=5;
        }
}
    
```



```

int getA()
{
    return a;
}

};

class B : public A
{
    protected:
        int b;
    public:
        void setB()
        {
            b = 10;
            setA();
        }
        int getB()
        {
            return b + getA();
        }
};

class C : public A
{
    protected:
        int c;
    public:
        void setC()
        {
            c = 15;
            setA();
        }
        int getc()
        {
            return c + getA();
        }
};

int main()
{
    B b1;
    C c1;
    b1.setB();
}

```

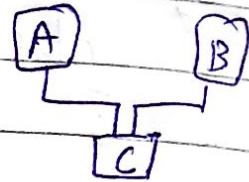


```

    b1. getB();
    do c1. setc();
        c1. getc();
    }

```

3. Multiple inheritance $\Rightarrow$



eg-

```

class A
{
    int a;
    protected:
        void setA()
        {
            a = 20;
        }
    int getA getA()
    {
        return a;
    }
};

class B
{
    int b;
    protected:
        void setB()
        {
            b = 33;
        }
    int getB()
    {
        return b;
    }
};

class C : public A, public B
{
    int c;
    public:
        void setc()
        {
            setA();
            setB();
        }
}

```



```

    c = getA() + getB();
}

int get(c)
{
    return c;
}

int main()
{
    c c1;
    c1.setc();
    c1.get(c);
}

```

→ In multiple inheritance, when two classes ^{or more} are inheritant then they may contain function with same name this cause name ambiguity in the derived class. This ~~conclusion~~ ^{confusion} can be resolved using resolution operation (::).

For example consider a function show existing in both A & B class. Both classes A & B are inheritance in C class. Therefore object of C say c1 when calls show() c1.show() results in ambiguity. To solve this problem show() can be called corresponding to its parent class using scope resolution operator

```

A:: c1.show();
B:: c1.show();

```


4. Multilevel inheritance $\Rightarrow$



```
class A
{
    int a;
protected:
    void setA()
    {
        a = 20;
    }
    int getA()
    {
        return a;
    }
};
```

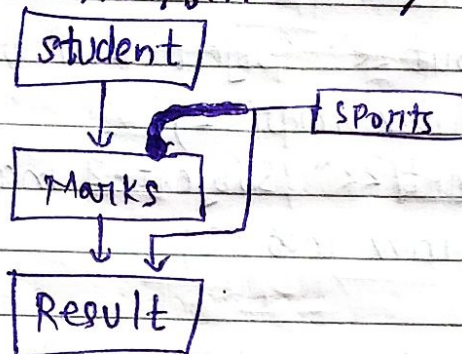
```
class B : public A
{
    int b;
protected:
    void setB()
    {
        b = 33;
    }
    int getB()
    {
        return b;
    }
};
```

```
class C : public B
{
    int c;
protected:
    void setC()
    {
        setA();
        setB();
        c = getA() + getB();
    }
    int getC()
    {
        return c;
    }
};
```


int main()
{

c c1;
c1.setc();
c1.getc();
}

5. Hybrid inheritance $\Rightarrow$



Q. WAP to calculate result of a student from subjects and from sports play.

Solⁿ

class student
{

protected :

int roll;

~~public :~~

void inpt()

{

cout << "Enter roll no. ";

cin >> roll;

}

void show()

{

cout << "roll no. = " << roll;

}

};


```
class subject : public student
{
```

```
protected:
```

```
int mark1;
```

```
int mark2;
```

```
public:
```

```
void subinpt()
```

```
{
```

```
cout << " Subject 1 marks = ";
```

```
cin >> mark1;
```

```
cout << " Subject 2 marks = ";
```

```
cin >> mark2;
```

```
}
```

```
int void showsub()
```

```
{
```

```
cout << " total marks = ";
```

```
return out mark1 + mark2;
```

```
}
```

~~int~~

```
int total()
```

```
{
```

```
return (mark1 + mark2);
```

```
}
```

```
};
```

```
class sports
{
```

```
protected: int mark3;
```

```
public:
```

```
void sport()
```

```
{
```

```
cout << " sport marks = ";
```



```
        cin >> marks;
    }

    int showspart()
    {
        return marks;
    }
};

class result : public subject, public sports
{
    int total;
public:
    void get()
    {
        inpt(); subinpt(); sport();
        showsub();
        showspart();
    }

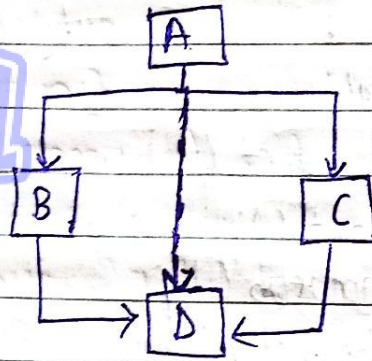
    void display()
    {
        show();
        total = showsub() + showspart();
        cout << "total";
    }
};

int main()
{
    result r1;
    r1.get();
    r1.display();
    return 0;
}
```


★ IMP

Virtual Base Class $\Rightarrow$ When a base class is declared virtual it resolves the ambiguity that occurs with multipath inheritance.

Consider a base class A which is inherited by both classes B & C. Then these two classes are inherited by class D.



Here class D inherits A from both B & C causing D to have to ~~two~~ copies of members of A. These causes confusion and can be dealt with declaring a base class as virtual when inheriting. Making A a virtual Base class.

class A

{

}

class B : public virtual A

{

}

class C : public virtual A

{

}

class D : public B, public C

{

}

}

Constructor in derived class $\Rightarrow$

Constructor is called when a class is instantiated ~~created~~ (object created). But In inheritance, only the derived class's object is created meaning the base class is objectless.

But in C++, if the constructor has no argument (base class) then it is the responsibility of derived class constructor to call the base class constructor. This is done automatically, i.e. even if the derived class has no constructor the base class constructor will be called by the derived object.

eg -

```

class A
{
    public:
        A()
        {
            cout << "Hello";
        }
};

class B: public A
{
    public:
        B()
        {
            cout << "Bye";
        }
};

int main()
{
    B b1;
}

```

O/P

HelloBye

If the base class constructor is parameterised then it is mandatory for the derived class to have a parameterised constructor. This constructor

will initialise the parameters of base class constructors

```
eg- class check
{
    int a;
public: check (int n)
{
    a = n;
    void show ()
    {
        cout << "a = " << a ;
    }
};

class mate : public check
{
    int b;
public: mate (int n, int m) : check(n)
{
    m b = m;
    void display ()
    {
        cout << "b = " << b; }
};

int main()
{
    mate m1(5, 9);
    m1.show();
    m1.display();
}
```


Syntax:—

```
derived constructor (arglist) : base constructor (parameter)
{
}

```

Q. WAP to initialise two member of base class using derived class constructor and print their sum in the base class.

Ans—

```
class A
{
    int a, b;
public:
    A(int p, int q)
    {
        a = p;
        b = q;
    }
    void show()
    {
        cout << "a+b=" << a+b;
    }
};

class B : public A
{
    int
public:
    B(int x, int y) : A(p, q)
    {
    }
};

int main()
{
    B b1(2, 3);
    b1.show();
}

```


function overriding :-

```

class A
{
    public:
        void display ()
        {
            cout << "Base class";
        }
};

class B : public A
{
    public:
        void display ()
        {
            // A::display ();
            cout << "Derived class";
        }
};

int main ()
{
    A a1;
    B b1;
    b1.display ();
    // A a1 = new B();
    // a1.display ();
}
    
```

It is a feature in C++ implemented in inheritance which cause a function in base class to be hidden by a function in the derived class with the same name. The base class function in such case is said to be overridden. To access the base class function scope resolution operators can be used in derived class or to access the base function

Using an object a base class reference object ~~may~~ need to the implemented as ~~and~~ a derived class object.

Abstract Class $\Rightarrow$

It is a base class which is generally not instantiated (object created) and is used primarily as serving class for the derived classes.

In c++ , the class can be made abstract by creating at least one pure virtual function inside the base class. This cause the compiler to generate compiler time error whenever an object is created for the base class. [syntax $\rightarrow$ virtual returntype functionname=0;]

Q. WAP to implement abstract class / pure virtual function.

Ans -

```
class Animal
{
public:
    virtual void walk = 0; //i.e. no instantiated
};
class reptile : public animal
{
public:
    void walk()
    {
        cout << "Slither " ;
    }
};
```


Page No. _____
Date: ____/____/____

```
class fish: public animal
```

```
{  
    public:
```

```
        void walk()
```

```
        {
```

```
            cout << "Swim";
```

```
        }
```

```
int main()
```

```
{
```

```
    reptile r1;
```

```
    r1.walk();
```

```
    fish f1;
```

```
    f1.walk();
```

```
    return 0;
```

```
}
```

Q. WAP to print service years left of 2 person by implementing a pure virtual function name service.

Ans -

```
#include <iostream.h>
```

```
class person
```

```
{
```

```
    public:
```

```
        virtual void service = 0;
```

```
};
```

```
class person1: public person
```

```
{
```

```
    int age = 30;
```

```
    int a;
```

```
    public:
```

```
        void inpt (int x)
```

```
        {
```

```
            a = x;
```

```
        }
```



```

        void service ()
        {
            cout << "service year left" << a - age;
        }
    };
class person2: public person
{
    int age = 40;
    int b;
public:
    void inpt (int y)
    {
        b = y;
    }
    void service ()
    {
        cout << "service year left" << b - age;
    }
};

int main ()
{
    person1 p1;
    person2 p2;
    p1.inpt(40);
    p2.inpt(60);
    p1.service();
    p2.service();
    return 0;
}
    
```


Q. WAP to transfer money from account 1 to account 2

```

class account
{
    int balance, acc;
public:
    account ()
    {
        cout << "Enter balance : ";
        cin >> balance;
        cout << "In account no. : ";
        cin >> acc;
    }
};

class trans : public account
{
public:
    void transfer (account &a1, int amt)
    {
        balance = balance - amt;
        a2.balance = a2.balance + amt;
    }

    void show ()
    {
        cout << "new balance : ";
        cout << a2.balance;
        cout << "new balance in account 1 : ";
        cout << balance;
    }
};
    
```



```
int main()
{
```

```
    trans t1, int amount;
```

```
    trans t2;
```

```
    t1 = account1;
```

```
    t2 = account2;
```

```
    cout << "Enter amount";
```

```
    t1 = transfer
```

```
        (int) amount;
```

```
    t1.transfer(t2, amount);
```

```
    t1.show();
```

```
    t2.show();
```

```
}
```


Unit - 4

Poly morphism

Constant members:— It is a keyword returning as const. It is used to make element of the program in-memory table (never change). The following elements can be declared const:—

- (i) Variables
- (ii) Pointers
- (iii) function return
- (iv) function argument
- (v) Data members of class
- (vi) Member function of class
- (vii) Class object.

eg - (i)

```
int main()  
{
```

```
    const int i=5; // fixed  
    cout << i;  
    i=i+2; // error  
    cout << i;  
}
```

eg - (ii)

```
int main()  
{
```

```
    int *const i; // address change  
    const int *i; // Now store first variable  
    int x=5; int z=2; store, further not.  
    i = &x; ✓  
    i = &z; X
```


(iii) ~~Ques~~ Constant arguments:—

```
void find (const int i)
{
    cout << i;
    i = i * 2; // error
}
```

```
int main
{
    const int a = 5;
    find(a)
    a = a + 2 // error
}
```

(iv) Constant return type:—

```
const int find()
{
    int a = 7;
    return 5; ✓
    return a; ✗
}
```

(v) ~~Ques~~ Declaring data member of class as constant ⇒

Data members which are constant can not be initialized during declaration but can be given a value by the constructor in its initializer list. Such data members can not be modified during the entire program.

```
class A
{
```

```
    const int i;
```

```
    A(int x) : i(x)
    { }
}
```

```
    void show()
    {
```

```
        cout << i; ✓
```

```
        i = i + 2 ✗
    }
```

```
int main()
{
```

```
    A a1(5);
    a1.show();
}
```


(vi) Declaring object as constant $\Rightarrow$

If an object of a class is declared as constant, all its data member can not be modified by member function or other function.

Syntax: —

```
const classname objective
const A a1(5);
```

(vii) Declaring member function as constant $\Rightarrow$

If a member function is declared constant then it can not modify any data member within scope.

Syntax:

```
returntype functionname() const
```

eg -

```
void show() const
{
    i = i + 5;
    cout << i;
}
```

```
void disp()
{
    i = i + 5;
    cout << i;
}
```

```
int main()
{
    A a1(5);
    a1.show();
    a1.disp();
}
```

Static Member Function $\Rightarrow$ (i) Static data member

A variable declared as static is always stored as a common copy for any no. of object i.e. it is independent of any object.

Therefore it is shared by all object. Static member are only initialised outside the class do not require an object for accessibility.

syntax:— static datatype variable name;
classname :: static member

Q. WAP to demonstrate static member.

```

1- class Box
{
    public:
    static int ob;
    int b;
    Box ()
    {
        b = 0;
    }
    void show()
    {
        cout << ++ob << ++b;
    }
};

int Box :: ob = 0;
int main()
{
    Box b1, b2, b3;
    b1.show();
    b2.show();
    b3.show();
}
    
```

(ii) Static function $\Rightarrow$ If a function is declared static then it is independent of all the object declared and there will be only one copy for all the object.

Q. WAP to implement static function.

```

Ans- class Box
{
    public:
        static int ob;
        int b;
        Box()
        {
            b = 0; ob = 0;
            ob = 0;
            ++ob;
        }
        void show()
        {
            cout << b;
        }
        static void show1()
        {
            cout << ob;
        }
};
int Box::ob = 0;
static int Box::show1();
int main()
{
    Box b1, b2, b3;
    b1.show();
    b2.show();
    b3.show();
    Box::disp();
    Box::disp();
}
    
```


Dynamic Object $\rightarrow$
pointer to objects:-

Q. WAP to input data of student and create a pointer to print that data.

Ans-
class student
{
 int roll;
 char name[10];
public:
 void setdata()
 {
 cin >> roll >> name;
 }
 void getdata()
 {
 cout << roll << "ln" << name;
 }
};

int main()
{
 student s1;
 student *p; p = &s1;
 p $\rightarrow$ setdata(); // s1.setdata()
 p $\rightarrow$ getdata(); // s1.getdata()
}

When a class is instantiated at run time then the object so made at run time are dynamic object. C++ not only allows to create dynamic object but also initialises them. Therefore providing a variable mem way to create a variable memory object. These object are created using

new operator. (Syntax - new classname;)
Once the dynamic object goes out of scope delete can be used to recover its memory.
Default constructor is automatically called with the new operator so is the destructor with the delete operator

```
class student
{
    int roll;
    char name[10];
public:
    student()
    {
        roll = 10;
    }
    void setdata()
    {
        strcpy(name, "ABC")
    }
    void showdata()
    {
        cout << roll << name;
    }
};

main()
{
    student *p = new student;
    p -> setdata();
    p -> showdata();
    delete p;
}
```

Operator overloading $\Rightarrow$ It is an example of polymorphism and is the process of providing special meaning to a certain operation. That is it extends the current capability of an operator and allows it to work with user define datatype. It can also be used for data conversion.

blw different type of data types.

Types of operator overloading:-

- (i) Unary operator overloading
- (ii) Binary operator overloading

~~Some~~ Almost all operators can be overloaded
except $?:$, $*$, $.$, $::$, $:$ $\rightarrow$

(i) Unary operator overloading $\Rightarrow$ This type of overloading takes no explicit arguments when operator function is a member function and takes one argument when it is a friend function.

Syntax —

Returntype operator opersymbol ()
{
}

eg-

class increment
{

int a;

SAO.K-12[+] public:

increment ()

{

a = 0;

}

void add ()

{

a = a + 1;

~~add~~ }

OR

void operator ++ ()

{

a = a + 1;

}


```

    int show()
    {
        return a;
    }
};

```

```

int main()
{

```

```

    increment i1;
    i1.add();
    i1.show();

```

OR

```

    increment i1, i2;
    i1++; i2++;
    cout << i1.show();
    cout << i2.show();

```

Q. WAP to overload unary minus.

Ans- class minus

```

{
    int a;
    public:

```

```

        minus (int x)
        {
            a = x;
        }

```

```

    void operator-()
    {

```

```

        a = -a;
    }

```

```

    int show()
    {
        return a;
    }
};

```

```

int main()
{

```

```

    minus m1(5), m2(6);
    -m1, -m2;
    cout << m1.show();
    cout << m2.show();
}

```


2. Binary operator Overloading $\Rightarrow$

In this type of overloading a binary operator is overloaded allowing it to know with user datatypes. Its function takes one argument, if it is a member function and two arguments, if it is a friend function.

eg -

```
class convert
{
    int feet, inch;
public:
    void inpt ()
    {
        cin >> feet >> inch;
    }
    void show ()
    {
        cout << feet << " inch";
    }
    void operator + (convert c)
    {
        int a, b;
        a = feet + c.feet;
        b = inch + c.inch;
        if (b >= 12)
        {
            a = a + 1;
            b = b - 12;
        }
        cout << a << " feet" << b << " inch";
    }
};

void main()
```



```

}
    convert c1, c2;
    c1.inpt();
    c2.inpt();
    c1+c2;
}

```

Q. WAP to subtract two complex no. by overload minus.

Ans- `#include <iostream.h>`
`class complex`
`{`
 `int a, b;`
`public:`
 `complex (int x, int y)`
 `{`
 `a = x;`
 `b = y;`
 `}`
 `void show ()`
 `{`
 `cout << "a <= "< a << " + i" << b << " << endl;`
 `}`
`complex & operator - (complex c)`
`{`
 `complex t;`
 `t.a = a - c.a;`
 `t.b = b - c.b;`
 `return t;`
`}`
`}`


```
int main()
{
    complex c1(5, 2);
    complex c2(3, 1);
    complex c3;
    c3 = c1 - c2;           // c3 = c1.operator-(c2)
    c3.show();
}
```

Q. WAP to overload binary multiple.

A -

```
class multi
{
    int a;
public:
    multi(int x) void inpt (int x)
    {
        a = x;
    }
    void show()
    {
        cout << a;
    }
    multi operator* (multi c)
    {
        multi t;
        t.a = a * c.a;
        return t;
    }
};

int main()
{
    int x, y;
```



```

multi complex c1, c2, c3; cin >> x; cin >> y;
{
    1. inpt (x);
    2. inpt (y);
    c3 = c1 * c2;
    c3.show();
}

```

Overloading , Similar to other operators insertion & extraction operator can also be overloaded. They are generally used object cin & cout which belong to istream and ostream class respectively. As both of these object do not belong to the user define class, they can not be declared as members. But to access members of user defined class, they must be declared as friend.

Q. WAP to overload i/p o/p operators

Ans - class complex

```

{
    int real;
    int imag;

```

public :

```

    friend istream &operator >> (istream &inp, complex &c); // extraction
    friend ostream &operator << (ostream &out, complex &c); // insertion
};

```

```

    istream &operator >> (istream &inp, complex &c)
    {
        inp >> c.real >> c.imag;
    }
}

```



```

ostream &operator << (ostream &out, complex& c)
{
    out << c.real << " + i" << c.imag;
}

int main()
{
    complex c1;
    cin >> c1;
    cout << c1;
}

```

Q. WAP to input 2 distance in feet & inch & then

Ans-

```

class distance
{
    int feet;
    int inch;

public:
    friend istream &operator >> (istream &inp, distance d);
    friend ostream &operator << (ostream &out, distance d);
    distance operator + (distance c2)
    {
        distance d;
        d.feet = feet + c2.feet;
        d.inch = inch + c2.inch;
        if (d.inch >= 12)
        {
            d.inch = d.inch - 12;
            d.feet = d.feet + 1;
        }
        return d;
    }
};

```



```
istream &operator >> (istream &inp, distance d)
{
```

```
    inp >> d.feet >> d.inch;
```

```
}
```

```
ostream &operator << (ostream &out, distance d)
{
```

```
    out << d.feet << d.inch;
```

```
}
```

```
int main()
{
```

```
    distance d1, d2, d3;
```

```
    cin >> d1 >> d2;
```

```
    d3 = d1 + d2;
```

```
    cout << d3;
```

```
    return 0;
```

```
}
```


conversion $\Rightarrow$

C++ compiler supports conversion b/w predefined datatypes only. In order to convert b/w user define types a separate conversion function must be provided. This conversion function is implemented depending upon the conversion which can be first.

- (i) Basic type to class type
- (ii) Class type to Basic type
- (iii) One class type to another class type

(i) Basic type to class type $\Rightarrow$

In this case the conversion function is used a simple constructor which takes one arguments only

eg -

```
class hours
{
    float hr;
public:
    hours (int z)
    {
        hr = z;
    }
    void show()
    {
        hr = hr / 60;
        cout << hr;
    }
};
```


int main()
{

hours h1;

int min

cin >> min;

h1 = min;

h1.show();

}

(ii) Class type to Basic type $\Rightarrow$

In this case, the conversion function is an operator function which doesn't have a return type but returns value of desired type.

The operator function must be a member of the class.

Syntax of OF:-
operator datatype()
{
}

eg- class hours
{

int hr

public:

void inpt()
{

cin >> hr;

}

operator int()

{

return hr * 60;

}

};


```
int main()
{
    hours h1;
    int min;
    h1.inpt();
    min = h1;
    cout << min;
}
```

Q. WAP to create a class inch which converts cm to inch & vice-versa.

Ans -

```
class inch
{
    int cm;
public:
    float inch;
    inch (int z) // B to C
    {
        incm = z;
    }
    operator int () // C to B
    {
        return 2.54 * in;
    }
    void show ()
    {
        inch = inch / 2.54;
        cout << inch;
    }
};

int main ()
{
    inch i1;
```


Date: / /

```
int cm;  
cin >> cm;  
i1 = cm; i1.show();  
cm = i1;  
cout << cm;  
return 0;
```

```
}
```


Compile time Polymorphism:- It is the process where a function call is bound to an appropriate function at compile time. It can be seen in function overloading & operator overloading.

Run time Polymorphism:-

It is the process where a function call is bound to an appropriate function at ~~temp~~ run time. It can be seen in virtual function.

* Pointer to object $\Rightarrow$

```
class A
{
    public: int a;
    void show()
    { a=5; cout<<a; }
};

int main()
{
    A a1;
    a1.show();
    A *p;
    p = &a1;
    p->show();
}
```

```
A *p = new A();
p->show();
```

* Pointer to derived class $\Rightarrow$

```
class papa :
{
    public:
    void show()
    { cout<<"I am your father";
    }
};
```



```

class son: public papa
{
    public: void show()
    {
        cout << "No you are not";
    }
};

int main()
{
    papa p1;
    papa *ptr;
    ptr = &p1;
    ptr->show();           // I am your father

    son s1;
    son *p2;
    p2 = &s1;
    p2->show();           // No you are not
    ptr = &s1;
    ptr->show();           // I am your father
}

```

⇒ ~~Each~~ Even though ptr has the address of derived class ptr->show() does not execute show() of derived class.

Because compiler always default to the type of ptr object and not to what's it pointing. This is bound at compile time.

To point to the derived class function ptr must also refer to the derived object. This can be done by making show() in the base class virtual. This causes the bounding of the function show to ptr at run time.


```
class papr
{
    virtual void show() { }
};
```

```
    ptr = &sl;
```

```
    ptr → show();
```

// No you are not

```
    ptr = &p1;
```

```
    ptr → show();
```

// I am your father

Virtual function ⇒

These are the functions which are bound to there reference objects at runtime.

```
class A
```

```
{
    public:
```

```
    int a;
```

```
    void show()
```

```
{
```

```
        cout << a; }
```

```
};
```

```
class B: public A
```

```
{
    public:
```

```
    int b;
```

```
    void show()
```

```
{
```

```
        cout << a << b; }
```

```
};
```

```
int main()
```

```
{
```

```
    A a1;
```

```
    B b1;
```

```
    A *p;
```

```
    B *q;
```

```
    q = &b1;
```

```
    q → b = 5;
```


q → a = 12;

q → show();

q → &a;

q → a = 15;

q → b = 17;

q → show();

p = &b;

p → a = 7;

p → b = 19;

p → show();

}

Unit - 5

Exception Handling

Defination:-

Exception are the unusual condition that occur during the program execution at run time.

There are two type of exception.

- (i) Synchronous
- (ii) Asynchronous

(i) Synchronous Exception $\Rightarrow$ These exceptions occur due to the events generated from within the program such as out of bounds, divide by zero, underflow and overflow of data structures.

(ii) Asynchronous Exception $\Rightarrow$ These are the exceptions which happen when the fault is beyond the control of the program. Such as disfailure, input-output interrupt etc.

$\rightarrow$ C++ deals with synchronous Exception.

Exception Handling :-

It is a fault handling mechanism to deal with synchronous Exception.

- Detect the problem [try block or hit]
- Report the problem [throw block]
- Report the error-correction [catch block]
- Handle Error

C++ Handle exception using 3 constructs :-
try, throw, catch.

(i) Try ⇒ It is a statement block which may contains an exception.

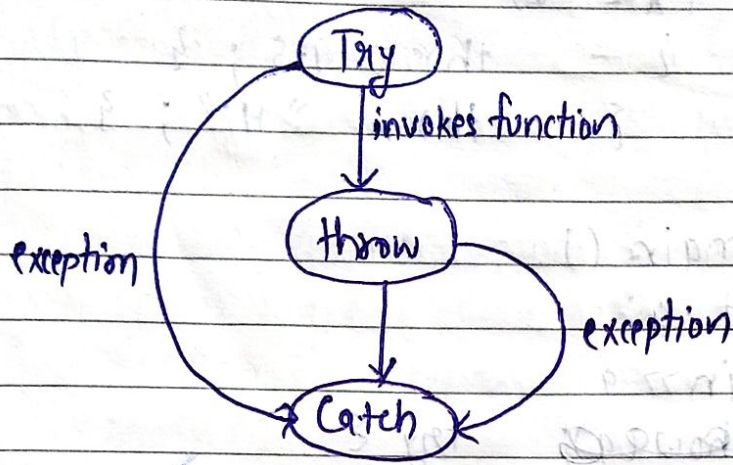
(ii) Throw ⇒ This keyword raises an exception and passes the culprit to the catch block.

(iii) Catch ⇒ It is a block which deals with the exception according to the arguments define.
Catch block generally follows its immediate try block and there can be multiple catch block for a single try.

Syntax:-

```
try {  
    throw (exception object);  
} catch (arguments)  
{  
}
```


If the exception object doesn't have a match in the catch block then the program will immediately terminate



Q. WAP to demonstrate divide by 0 exception

Ans-

```
void main ()
```

```
{
```

```
int a, b;
```

```
cin >> a >> b;
```

```
try {
```

```
if (b == 0)
```

```
{
```

```
throw b;
```

```
}
```

```
else
```

```
cout << a/b;
```

```
}
```

```
catch (int i)
```

```
{ cout << "b is 0"; }
```

```
cout << "Yey";
```

```
cout << "lecture over";
```

```
}
```



```

eg - void show(int x)
{
    if (x == 0)
        throw 0;
    else if (x == 1)
        throw 1.5;
    else
        throw "Hi";
}

```

```

void main()
{
    int a;
    cin >> a;
    try {
        show(a);
    }
}

```

```

catch (int z)
{
    cout << z;
}
catch (float f)
{
    cout << f;
}
catch (char * m)
{
    cout << m;
}
}

```

catch(...)

```

{
    cout << "something";
}

```

Generic Programming $\Rightarrow$ It is an approach which enables to develop generic programs whose parameters can work with a variety of datatypes.

It is implemented using Templates.

Template $\Rightarrow$ It is a feature which allows to create a generic class or functions that can work with different

datatypes. In C++, There are two ways to create

- (i) class template
- (ii) function template

(i) Function template:— These templates can be used to generate generic functions.

syntax:—

```
class < template c > template < class c1 >  
c1 large ( c1 a, c1 b )  
{  
    return a > b ? a : b;  
}
```

eg -

```
void main()  
{  
    int a, b;  
    float c, d;  
    char x, y;  
    cin >> a >> b;  
    cin >> c >> d;  
    cin >> x >> y;  
}
```

```
cout << large (a, b);  
cout << large (c, d);  
cout << large (x, y);
```

eg -

```
template < class c1, class c2 >  
class < template c1, template c2 >  
void large ( c1 a, c2 b )  
{
```

```
    cout << a > b ? a : b;
```

```
}
```

```
void main()
```

```
{  
    int a, b;  
    float c, d;
```

```
    cin >> a >> b >> c >> d;
```

```
    large (a, c);
```

```
}
```


(ii) Class template $\Rightarrow$ It is an template.
When a template is created from a template class it is known as class template.

Syntax: — `template <class T>`
`classname <type> object (arguments)`

eg - `template <class T>`
`class check`
`{`

`T a;`

`T b;`

`check (T x, T y)`

`{ a = x;`

`b = y;`

`}`

`T show ()`

`{`

`return (a + b);`

~~`return 0;`~~

`}`

`};`

`void main()`

`{`

`check <int> c1 (5, 2);`

`check <float> c2 (7.2, 5.3);`

`cout << c1.show();`

`cout << c2.show();`

`}`

Q. WAP to swap two integer, two floats & ~~two~~ integer & char.

Ans -

```
template < class c1, c2 >
void swap ( c1 a , c2 b )
{
```

```


    c1 e;
    e = a;
    a = b;
    b = e;


```

```
a = a + b;
```

```
b = a - b;
```

```
a = a - b;
```

```
cout << a << b;
```

```

}

void main ()
{

```

```
    int a, b;
```

```
    float c, d;
```

```
    char e;
```

```
    cin >> a >> b >> c >> d >> e;
```

```
    swap ( a, b );
```

```
    swap ( c, d );
```

```
    swap ( a, e );
```

```

}

```

Q. WAP to create a class template which is used to perform arithmetic calculation.

Ans -

```
template < class T1 , class T2 >
class calc
```

```
{
```

```
    T1 a;
```

```
    T2 b;
```

```
    calc ( T1 x , T2 y )
```

```
    {
        a = x;
```

```
        b = y;
```

```
    }

```


~~void add~~

```
void add(T1 a, T2 b)
```

```
{
```

```
    cout << a + b;
```

```
}
```

```
void sub(T1 a, T2 b)
```

```
{
```

```
    cout << a - b;
```

```
}
```

```
void multi(T1 a, T2 b)
```

```
{
```

```
    cout << a * b;
```

```
}
```

```
}
```

```
void main()
```

```
{
```

```
    int x, y; float z, p;
```

```
    calc < int, float > c1(x, z);
```

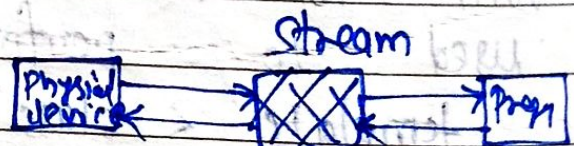
```
    c1.add(x, y);
```

```
    c1.sub(x, y);
```

```
    c1.multi(z, p);
```

```
}
```

Streams $\Rightarrow$



It is a sequence of bits (medium) which enables to send data to the program and to receive data from the program. It provides an interface which takes data from the input stream and delivers data to the output stream.

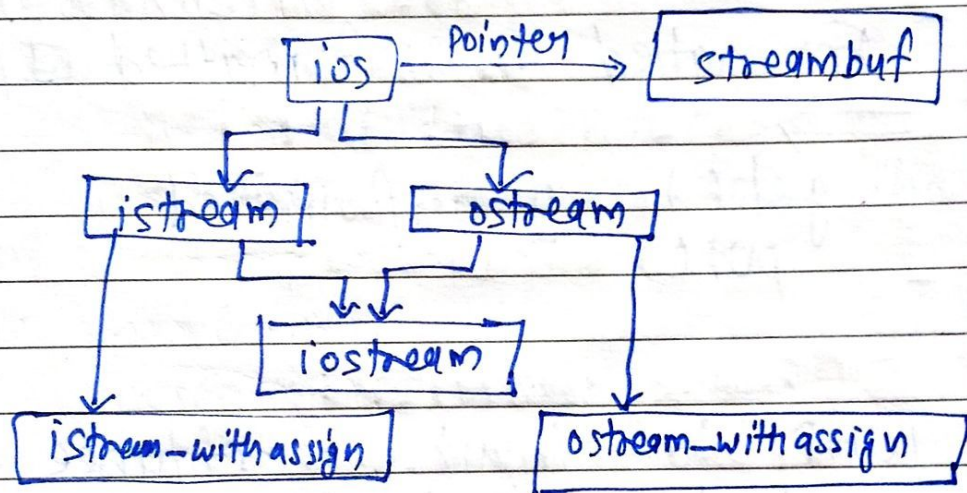
Input Stream :—

It receives data from the source.

Output Stream :—

It delivers data to the destination.

Stream class $\Rightarrow$ It is a hierarchy of classes which specify the implementation of various input/output streams.



console I/O in stream classes

IOS :— This base class contains all the basic facilities of io unit utilize by child classes.

istream :— This class contains all basic input functions. It also defines $\gg$ operator.

ostream :— It contains output functions. It also defines $\ll$ operator.

istream - with assign and ostream - with assign

⇒ Similar to istream & ostream but provides the facility of assignment operator.

iostream ⇒

This class inherits functionality of both istream and ostream. They all are defined in iostream.h.

Formatted & unformatted I/O functions

⇒
(i) get() } unformatted
(ii) put()

Q. WAP to input a sentence & print and count
Ans- #include <iostream.h> count letters.
~~int~~ void main()
{

```
    int char ch[20];  
    for (int i=0; i<20; i++)  
    {  
        cin.get(ch[i]);  
    }  
    for (i=0; i<20; i++)  
    {  
        cout << ch[i];  
    }
```

}

or

```
void main()
{
    char c[200];
    c = cin.get();
    while (c != '\n')
    {
        cout << put(c);
        c = cin.get();
    }
}
```

or

```
void main()
{
    int i = 0, int p;
    char sentence[1000]; char ch;
    do
    {
        cin.get(ch);
        sentence[i] = ch;
        i++;
    } while (ch != '\n');
    cout << put(i);
    for (p = 0; p < i; p++)
    {
        cout << put(sentence[p]);
    }
}
```

(iii) ~~at~~ getline() & write() $\Rightarrow$ getline(variable, size)

Q. WAP to input a sentence

Ans-
void main()
{
 char sentence[100];


```
cin.getline(sentence, 100);
cout.write(sentence, 100);
```

3

Manipulators: —

width, precision, fill

```
float f; int n; int a = 12;
```

eg- $f = 3.14 * n * n;$

~~then~~

```
cout.precision(3);
```

```
cout << f;
```

// 3.140

```
cout.fill('*');
```

```
cout.width(5);
```

```
cout << a;
```

*	*	*	1	2
---	---	---	---	---

```
cout << setw(3) << a;
```

```
cout << setprecision(5) << f;
```

] → for only line

File handling: —

Q. WAP to write data in a file.

Ans - #include <fstream.h>

```
int main()
```

```
{
```

```
ofstream fout("Hello");
```

```
cin >> a;
```

```
fout << "Go";
```

```
fout << a;
```

```
fout << "This is my file";
```

```
fout.close();
```

3

or
ofstream fout;
fout.open("Hello");

Q. WAP to input a string and store it in data.txt.

Ans-
#include <fstream.h>
int main()
{
 ofstream fout;
 fout.open("data.txt");
 char ch[100];
 cin.getline(ch, 100);
 ~~fout~~ fout.write(ch, 100);
 fout.close();
}

Q. WAP to ~~print~~ counting from 1 to 100 to the file.

Ans-
#include <fstream.h>
int main()
{
 ofstream fout;
 int i;
 fout.open("count.txt");
 for(i=1; i<=100; i++)
 {
 fout << i;
 }
 fout.close();
}


```

Q. #include <fstream.h>
int main()
{
    ifstream fin;
    int i;
    fin.open("count.txt");
    while (fin
    fin >> i; // i = fin.get();
    while (fin != 0)
    {
        cout << i; // cout.put(i);
        fin >> i; // i = fin.get();
    }
    fin.close();
}

```

Q. WAP to Read no. of lines in a ~~program~~ file.

Ans -

```

#include <fstream.h>
{
    ifstream fin; char ch[20];
    fin.open("count.txt");
    int a = 0;
    while (fin != 0) // or while(fin)
    {
        fin.getline(ch, 20);
        a++;
    }
    cout << a;
    fin.close();
}

```


Modes of file $\Rightarrow$

ios :: io

ios :: out

ios :: ate

ios :: trunc

ios :: no create

ios :: no replace

Q. WAP to ~~no~~ write student data to a file.

Ans- #include <iostream.h>

#include <fstream.h>

class student

{

int roll;

char name[10];

public:

void inpt()

{

cin >> roll;

cin >> name;

}

void show()

{

cout << roll;

cout << name;

}

void main()

{


```

student s[100];
fstream fs;
int i;
fs.open("Hello", ios::in | ios::out | ios::binary);

```

```

for (i=0; i<100; i++)
{

```

```

    s[i].input();
    fs.write ((char*) &s[i], sizeof(s[i]));

```

```

    fs.seekg(0);

```

```

    i=0

```

```

    while (fs)
    {

```

```

        fs.read ((char*) &b[i], sizeof(b[i]));

```

```

        i++;
    }

```

```

    for (i=0; i<100; i++)
    {

```

```

        b[i].show();
    }

```

```

}

```

```

}

```

```

}

```

Q. WAP to copy contain of one file to other.

Ans - #include <fstream.h>

```

int main()
{

```

```

    ifstream fin;
    ofstream fout;

```

```

}

```



```
fin.open ("Hello.txt", ios::app);  
fout.open ("copy.txt", ios::app);
```

```
char ch;
```

```
ch = fin.get();
```

```
while (fin)
```

```
{
```

```
    fout << ch;
```

```
    ch = fin.get();
```

```
}
```

```
return 0;
```

```
}
```


Class

Class is a blueprint of object.

Class is a logical entity.

Class is declared once.

Class doesn't allocate memory when it is created.

~~where~~ we can define a class by using class keyword

Object

Object is an instance of a class

Object is a real time entity

Object is created many times as per requirement.

Object allocates memory when it is created.

New keyword, newinstance(), clone() & deserialization

Data Hiding

→ Process that ensures exclusive data access to class members and provides object integrity by preventing unintended or intended changes

→ Focuses on protecting data.

→ Helps to secure the data.

Abstraction

An OOP concept that hides the implementation details and shows only the functionality to the user.

Focuses on hiding the complexity of the system.

Helps to hide implementation details & display only the functionalities to the user.