C++

C++ is a object base or partially object oriented high level programming language. Developed by Bjarne Stroustrup in the year 1982 to 1983.

C++ is initialy known as "C with class!" But after adding the feature of OOPS it will be refered as C++.

NOTE:
The file extension of C++ is ".CPP".

OOPS:

Object oriented programming is an approach modular programming style by creating separate partition memory area for data and function, and assign such templet to user and programer on demand.

Application area of OOPS:
① Software developing either it will be windows application, web application or android app.
② Object oriented database.
③ Hardware and networking.
④ AI (Artificial Intelligence) and expert system.
⑤ Cyber security and cyber crime.

Feature of OOPS:

① class
② Object

③ Data Abstraction
④ Data Encapsulation
⑤ Inheritance
⑥ Polymorphism:
It is a greak term which english equivalent is "one name different forum for different purpose".
⑦ Dynamic binding
⑧ Massage passing.

~~① Class:~~

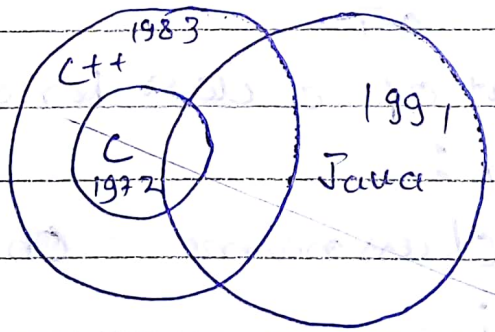
* The differency b/w C and C++

C	C++
① C is a POP (Procedural oriented programming lang)	① C++ is partially Object oriented programming language (OOPS)
② Extension of c file is ".c"	② Extension of C++ file is .CPP.
③ In C we have 32 keywords.	③ In C++ we have more than 64 keywords. (64+11)
④ In C programming there is no access specifiers.	④ But in C++ we have three access specifiers PUBLIC, Private, Protected, Private visiblity mode in C++ is default.
⑤ C doesn't have class as a user defined data type.	⑤ C++ is initially known as "C with class", Actually class is a logical extension of structure

* Difference b/w POP and OOPS

POP.	OOPS
① It stands for Procedural oriented programming language	
② C is the most popular POP (Programming) language.	② C++, JAVA, PYTHON, C#, --- are some popular OOPS languages.
③ In POP data and functions are independent to each other.	③ In OOPS data and functions are bind (wrapped) within a single entity is known as data encapsulation
④ In POP the program execution flow from top to bottom	④ In OOPS the program execution flow bottom to top.
⑤ In POP there is no access specifiers.	⑤ In OOPS there must be access specifier like Public, Private, Protect

* The relationship b/w C, C++ and Java.



① Class:

A class is a user defined data type i.e a logical extension of structure.

A class consist of two things:

① data member:

It can be any valid data type like int, char, float, double etc.

② Member function:

It is a user defined function and any operation on data member. Even initialization must be done inside a member function.

Syntax:

```
class name
{
    Access mode
    <Data member>
    <member function>
};
```

② Object:

A class variable is known as object. ~~or~~

or

Instance of a class is known as object.

or

The run time entity of a class is known as object.

or

Blue print of a class known as object.

Syntax:

```
Class name  Obj_Name;
```

Ex;

```
employee  e;
```

* Advantage of creating Object:
@ Objects are used to access class members.

Q. WAP in C++ to add two no. using class and object.

```
#include <iostream.h>
#include <conio.h>
  class add
  {
  int a, b;   // Private data member
  Public:
   void add1()  // Public number function
  {
  cout<<"\n Enter two nos.";
   cin>>a>>b;
  cout<<"\n sum = "<<a+b;
   }
  };
  void main()
  {
add obj;  // creating the object of class
                and object.
  clrscr();
obj.add1();  // call the number function
                by object of class add.
   getch();
}
```

cout → It stands for output objed and comes under the header file iostream.h

cin : It stands for input object and comes under iostream.h header file.

iostream.h : It is a header file where i/o means input output and stream is a sequence of data.

Q WAP in c++ to check the number is odd or even.

```
#include<iostream.h>
#include<conio.h>
  class chick
  {
    int n;
  public:
  void ~~int~~ check1()
    {
    cout<<"\n Enter no.";
      cin>>n;
  if(n%2==0)
  cout<<"\n even no.";
  else
    cout<<"\n odd no.";
    }
};
    void main()
    {
      ~~main~~ check obj;
        clrscr();
        obj.check1();
    } getch();
```

* Data abstraction:

Hiding unnecessary and complex part of a program and provide only the necessary segment to the user is known as data abstraction.

In C++ data abstraction will be achieved by 3 available access specifiers (access mode or visiblity mode):-

(1) Public:-

Members of a class are visible every-where.

(2) Private:-

Members ~~of~~ are visible only inside the class.

(3) Protected:-

Members are visible in same class and also in derived (sub) class.

* Data Encapsulation:

Binding/wrapping of data member and member function together within a single entity is known as data encapsulation.

NOTE:

It will be achieved by class.

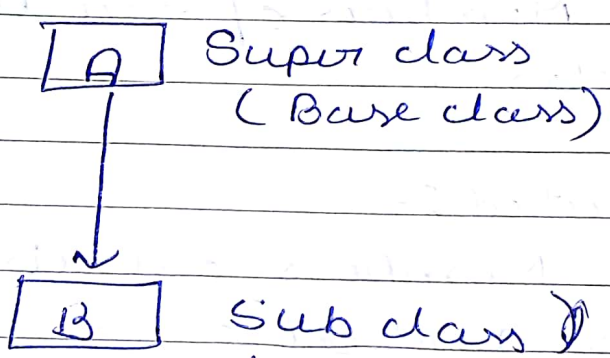
Example:

```
class a
{
int x;
Public:
void get_data();
void put_data();
};
```

5. Inheritance:

It is an important concept of OOPS, where a sub class can inherit some or all properties of its associated super class.

eg.



6. Polymorphism:

It is a greek term whose english equivalent is "One name different form for different purpose.

Polymorphism

Runtime
↳ function overriding
(Virtual function)

compile time
↳ function overloading
↳ operator overloading

7. Dynamic Binding:
Run time polymorphism is also known as dynamic binding or late binding.
Compile time polymorphism is also known as static binding or early binding.

8 Mussage Passing:
Passing arguments to member function is known as message passing.

Consider the following examples:

```
#include <iostream.h>
#include <conio.h>
    class a
{
    int x, y; //Private data member
public:
void add (int a, int b) //public
                        member function
{
```

```
    x = a;
    y = b;
cout<<"\n Sum = "<<x+y;
  }
};

void main()
{
   a obj;
   int a,b;
   clrscr();
cout<<"\n Enter a and b";
   cin >>a>>b;
   obj.add(a,b); //message passing

   getch();
}
```

Date
Page

* Special member function in C++:-
  1. Constructor
  2. Destructor
  3. Static function (class function)
  4. Friend function
  5. Inline function
  6. Virtual function
  7. Pure virtual function.

① constructor:-

A constructor is a special member function and it is special due to following properties-

1. The name of the constructor is same as the class name.
2. A constructor always declare in public.
3. A constructor never returns any value even void.
4. A constructor is auto executed when the associated class objects creates.
5. A constructor construct the class object.

Type of constructor:

① Default constructor (Non parametrized):

② A constructor without any argument is known as default or non parmeterized constructor.

Example:

```
#include <iostream.h>
#include <conio.h>
class a
```

```
{
public:
a() // default constructor
{
cout <<"\n default constructor !!";
}
};
void main()
{
clrscr();
a obj; // call the default constructor
getch();
}
```

(2) Parametrized constructor:

This constructor should have some parameters.

ex:

```
#include <iostream.h>
#include <conio.h>
class a
{
public:
a(int x) // Parametrized const
{
cout <<"\n Parametrized const =" << x;
}
};
void main()
{
```

```
    clrscr();
    a obj(88); // call the parametrized
                  constructor.
    getch();
}
```

③ Copy constructor :-

It is also a special type of parametrized constructor with a limitation i.e. "It always consists class object reference(address) as its parameter."

Ex:

```
#include<iostream.h>
#include<conio.h>
    class a
    {
    Public:
    a()
{
cout<<"\n default";
}

a(a& obj1)
{
cout<<"\n copy constructor";
}
};
void main()
{
clrscr();
```

Date
Page

```
a ob1; // call default constructor
a ob2( ob1); // call the copy constructor
getch();
}
```

* Assignment on constructor:

① Add two number by constructor.

```
#include <iostream.h>
#include <conio.h>
class a
{
  int x,y,z;
  Public:
    a()
  {
cout<<"\n Enter two no.";
  cin>>x>>y;
      z=x+y;
cout<<"\n Sum="<<z;
  }
};

Void main()
{
  clrscr();
a obj; // call default constructor

  Get ch();
}
```

Date Page

② WAP in C++ to check a no. is prime or not using parametrized constructor.

```
#include <iostream.h>
#include <conio.h>
class a
{
int i,f;
Public:
a(int n)
{
f=0;
for (i=1; i<=n; i++)
{
if (n%.i==0)
f++;
}
if (F==2)
cout<<"\n Prime";
else
cout<<"\n not prime";
};
void main()
{
int n;
cout<<" enter a no.";
cin>>n;

a obj(n);
getch();
}
```

3. WAP in C++ to generate the series of prime b/w a given lower to upper limit, but limits must be the parameter of parametrized constructor?

ex. a(int l, int u)

4. WAP in C++ to check the no. is armstrong or not using copy constructor.

Ex:

Q. WAP in C++ to add two no. inside the copy constructor but the no. should be the class data number.

```
#include<iostream.h>
#include<conio.h>
    class a
     {
     int x, y; //data number.
     public:
     a()
     {
cout<<"\n enter two no.";
 cin>>x>>y;
 }
 a(a &a1)
 {
 cout<<"Sum="<<a1.x+a1.y;
 }
};
```

```
void main()
{
clrscr();
a obj1;
a obj2(obj1);

getch();
}
```

3. Prime no.:

```
#include<iostream.h>
#include<conio.h>
  class S
{
  int i, j;
  public:
  S(int l, int u)
  {
    for(i=l; i<=u; i++)
    {
     for(j=2; j<=i; j++)
     {
       if(i%j==0)
         break;
     }
     if(i==j)
     cout<<"\t"<<i;
    }
  }
};
```

Date
Page

```
void main()
{
  int l, u;
  clrscr();
  cout << "\n lower and upper limit";
  cin >> l >> u;
   s obj(l, u);

  getch();
}
```

4.

Q WAP in C++ to check the no. is armstrong or not using copy constructor.

```
#include<iostream.h>
#include<conio.h>
    class a
    {
     int n,i;
    public:
     a()
    {
    cout<<"\n Enter no.";
     cin>>n;
    }
    a(a &obj)
    {
    int r,s=0;
    int num = obj.n;
     int t = num;
    while(num>0)
    {
      r = num%10;
    s = s + r*r*r;
     num = num/10;
    }
    if(s==t)
    cout<<"\n Armstrong";
    else
    cout<<"\n not armstrong";
    }
};
```

```
void main()
{
  clrscr();
  a obj1;      // call the default constructor
  a obj2(obj1); //call the copy constructor
  getch();
}
```

Sizeof → return the size in bytes.

NOTE;

class occupy 1 bytes, if and only if it doesn't contain any data type.

If class contain data type then it occupy the sum of all the data type size.

* Distructor:

It is also a special member function in C++ programming language but it is just opposite to constructor.

A distructor is used to ~~display~~ destroy the objects and free the resources

Properties of Distructor:

1. The name of the distructor is same as the class name.
2. Distructor is always declared in public.
3. Distructor never return any value even void.
4. A distructor is never be parameterized.

Date
Page

⑤ Distructor always start with tilde (~) sign to make differencial with constructor.
⑥ Distructor is auto created after the constructor.

Q WAP in C++ to implement or explain distructor.

```
#include <iostream.h>
#include <conio.h>
    class a
{
public:
  a()
{
cout<<"\n default";
}
a(int x)
{
 cout<<"\n Paramitrized = "<<x;
}
~ a()
{
 cout<<"\n destroyed \n";
}
};  void main()
{
  clrscr();
  a obj1;
  a obj2(8);
    getch();
}
```

* Scope resolution operator (::):
It is a special operator in C++.

Suppose we have a variable with same name one as a local and another as global. In C programming we cannot make the differentiation in this situation among the variable of same name.

So C++ introduced a new variable to solve the problem known as scope resolution operator (::)

eg:

```
#include <iostream.h>
#include <conio.h>
        int a=10; //global variable
        void main()
{
int a = 9; //local variable
        clrscr();
cout<< a<<"\t"<<::a;
        getch();
}
```

Output 9 10

* In c++ the scope resolution operator (::) used in following concepts:
1. To call the static member function.
2. To create the member function body outside the class.

Date
Page

③ File handling.
④ To make differentiation b/w global and local variable with same name.

⑤ WAP in C++ to check to check the string is pallindrome or not. where the body of the member function should be created outside the class.

⇒

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class a
{
 char a
{
 char str[10];
 int x;
public:
void check();
};

void a::check()
{
 char str1[10];
cout<<"\n enter string";
 cin>>str;
 strcpy(str1,str);
 strrev(str1);
 x=strcmp(str,str1);
 if(x==0)
```

```
cout<<"\n Pald string ";
else
cout<<"\n Not pald string";
}

void main()
{
  a a1;
  clrscr();
  a1.check();
  getch();
}
```

Q Enter a sentence and display the index of last space.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
  class a
  {
    char a[100];
    int i, count, j;
    public:
    a()
  {
    count = 0;
  cout<<"\n enter a sentence";
    gets(a);
```

```
For(i=0; a[i] != NULL; i++)
 For(j=i-1; a[j] != NULL; j--);
 {
  if (isspace (a[i]))
     count++;
     if (count == 1)
       break;
  }
 cout << j;
  }
};

 void main()
 {
   clrscr();
   a obj;
   getch();
 }
```

Q How to get a line/sentence as an input in C++.

Ans
```
#include <stdio.h>
   char str[100];
   cout << "\n enter a line";
   gets(str)
```

gets function comes under stdio.h.

Q To check string is padld or not.

```
#
#
    class a
    {
    char a[20];
    int i K, l, count;
     public;
    a()
    {
     cout<<count = 0;
     cout<<"\n enter a word";
      cin>>a;
For(i = 0; a[i]!=NULL; i++);
For(K = 0; l = i-1; K<i/2; K++, l--)
 {
  if (a[K] == a[l])
     count++;
 }
if (count == i/2)
  cout<<"\n Padld string";
else
cout<<"\n Not Padld string";
 }
};

   void main()
   {
      clrscr();
     a obj;
```

Date
Page

getch();
{
}

Date
Page

* Friend function:-

Friend is a keyword in C++ programming language and when a function starts with a keyword friend. It is known as friend function.

example:-

```
Friend void show ()
{
- - - - - -
- - - - - - //body of friend function
- - - - - -
}
```

Properties:

① A friend function can declare either public or private.

② A friend function is not in the scope of the class under which it is declare.

③ It is directly called by name, so we don't have to create object in case of calling a friend function

④ It generally consist class object ~~type~~ as argument in case of work with class members.

Advantage:

① Using friend function we can make private member visible outside the class.

* example:

```
#include <iostream.h>
#include <conio.h>
    class a
```

```
{
private:
Friend void display ()
  {
  cout << "\n hello c++";
    }
  };

void main()
  {
    clrscr();
    display ();
    getch();
  }
```

② check the no. is odd or even using friend function but the no. should be the class data member.

```
#
#
  class a
  {
    int x;
  public:
  void get_data()
    {
  cout << "\n enter no";
    cin >> x;
    }
Friend void display (a ob1)
    {
      if (ob1.x % 2 == 0)
```

```
cout<<"\n even";
else
cout<<"\n odd";
}
};

void main()
{
clrscr();
a obj();
xyz.get_data();
display(xyz);

getch();
}
```

* Static Keyword in C++ :-

using the keyword static we can declare :-

1) Static data
2) Static function (class function)

1) Static data :-

A data starts with the keyword is known as static data.

example :-

Static int x;

properties :-

1) A static data is default initialized by zero.
2) In successive function call (recursion) a static data preserve updated value.
3) A single copy of static data is created and share by all the objects of same class.

* Since it is a class data so directly acc... by class name with scope resolution operator (...

* The diff. b/w Static data (class data) and non-stat... data (instance data)

Static data (class data)	Instance data (non-...
i. Static data is default initiali- -zed by zero.	(i) default initialized b... garbage valu...
2. It is a class data.	(ii) It is instance data.
3. directly accessed by class name with scope resolution operator.	(iii) It is access by object with Dot (.) operator.
4. It is associated to class.	(iv) It is associated with object.
5. A single copy is created for Static data and share by all objects of same class.	(v) Multiple copies are created for non-static for individual object.

* Example 1 :

Static data is default initialized by zero.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    Static int x;
    clrscr();
    cout<<"\n"<<x;
}
```

```
int x;
clrscr()
cout<<"\n"<<x;
```

```
getch();
}
output = 0
```

```
getch();
}
output = garbage.
```

② Persist the updated value in successive function call (recursion):

-> Factorial by recursion:

```
# include <iostream.h>
# include <conio.h>
void main()
{
void fact(int );
clrscr();
int n;
cout<<"\n enter a no.";
fact(n);
getch();
}
void fact(int n)
{
Static int x=1, f=1;
if (x<=n)
{
f=f*x;
x++;
fact(n);
}
else
cout<<"\n" << f;
}
```

Date
Page

(3) Static data member is directly accessed by class name with scope resolution operator.

```
#
#
 class a
{
public:
 Static int x; // static data member.
};

int a :: x; //activate the static data member.
 void main()
{
 clrscr();
cout<< "\n" << a :: x; //access by class name
                        with :: operator
 getch();
}
```

Ex 4. Single copy of static data is created and share by all the object of same class.

```
#
#
  class a
 {
  public :
 Static int x ; // Static data member
  void count()
 {
```

```
    x++;
}
void dis ()
{
  cout<<"\n x = "<<x;
}
};

int a::x; //active the static member.
  void main()
  {
    clrscr();
    a a1;
    a a2;
    a a3;
   a1.count();
   a2.count();
   a3.count();
   a1.dis();
   a2.dis();
   a3.dis();
    getch();
  }
```

* Static member function (class member function):-

A function starts with the keyword static is known as static member function.

example:-

```
Static void show()
  {- - - - - - - - // body of static member function
  }.- - -
```

Date
Page

properties:-
① A static member function is directly call by class name with :: operator.
② A static member function can work with static data only.

Q. WAP in C++ to implement static member function.

```
#include<iostream.h>
#include<conio.h>
  class a
  {
  static int x;
  public:
  static void show()
    {
    cout<<"\n hello c++ =" <<x;
    }
  };

  int a::x;
    void main()
  {
    clrscr();
  a::show();
      getch();
  }
```

* Static ~~dat~~ function (class function)	non static function (instance function)
1. Static function starts with keyword Static. ex. Static void show() { - - - - } - - - - //body of static function.	① Non static function never have the static key word. ex: void show() { - - - - } - - - //body of non static function
2. Static function is also refer as class function.	② Non static function are also refer as instance function.
3. A static function is directly call by the class name with scope resolution operator (::).	③ A non static function is call by the object with dot operator (.)
4. Static member function can work with static data member of the class.	④ The non static member function can work with non-static data member and ~~class~~ Static data mem

* Inline function:
Inline is a key word in C++.
Syntax:
* <datatype> <Function>(<parameter>){<Statement>;}
* inline <datatype> <class>::<Function>
(<parameters>){<statements>;}

The first syntax example declare an inline function by default.
The second ~~form~~ syntax example declare an inline function explicitly.

Syntax for declaring an inline function:
inline return-type Function-name(Parameter)
{
--- //Function code.
}

Limitation of inline function (disadvantage):
Inline is only a request to the compiler not a command. Compiler can ignore a request for inline in case of function containing :
a) looping Statement (while, do while, for loop)
b) Static variable.
c) recursive.
d) Function have the return type other than void.
e) Switch case and goto statement.

Advantage:

a) function call overhead does not occur.
b) It also save the overhead of push and pop variable on the stack when function is called.
c) It also saves the overhead of a return call from a function
d) Inline function may be useful (if it is small) & For embedded systems its because it has less code.

NOTE:

C++ provides an inline function to reduce the function call overhead. When the inline function is called the whole code in a line and directly executes without function calling overhead.

Q What is function call overhead.

When the program executes the function call instruction, the CPU store the memory address of the instruction following the function call, copy the arguments of the function on the stack and finally transfer control to the specified function.

All these operations are referred as function calling overhead.

Ex:

```
#
#
class a
{
public:
inline void show()
```

```
{
cout<<"\n hello "<<"\n";
}
};

void main()
{
 a obj;
 clrscr();
obj.show();
 getch();
}
```

* Inheritance in c++:

Inheritance is an important concept of OOPs where a sub class can inherits some or all properties of its associated super class.

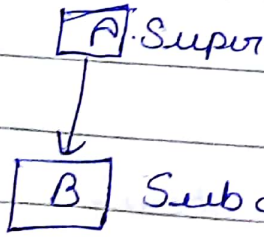
Advantage:

1) Using inheritance we can achieve reusability in OOPS.

2) By reusiblity the length of the program should be reduce as ~~per~~ further result it compiles and execute much faster.

Types:

1) Single Inheritance



Inline · cpp

Date
Page

On single inheritance there is only one super and one sub class.

Q. WAP in C++ to implement Single inheritance.

```
#
#
class a
{
 public :
  int l;
 void getl()
  {
    cout<< "enter l";
    cin>>l;
  }
};
  class b: public a
  {
    int b1;
  public;
  void get b()
  {
  cout<<"\n enter b";
    cin >> b1;
  }
  void area()
  {
  cout<<"area = "<< l * b1;
  }
};
```

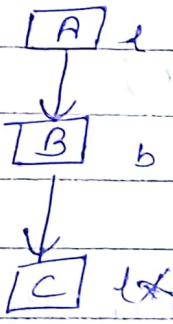
```
void main()
{
  clrscr();
b obj;
 obj.getl();
 obj.getb();
 obj.area();
 getch();
}
```

Q. Using single inheritance find the area of rectangle where both the data in the private.

```
#
#
 class a
 {
  int l;
 public:
 int getl()
 {
 cout<<"\n enter l";
 cin>>l;
 return l;
 }
};
class b : public a.
 {
  int b1;
 public:
  void getb()
```

```
{
cout<<"\n enter b";
    cin>>b1;
}
    void area()
      {
cout<<"\n area of rect="<<get1()*b1;
}
};
    void main()
      {
        clrscr();
        b obj;
        obj.getb();
        obj.area();
          getch();
      }
```

② Multilevel Inheritance:



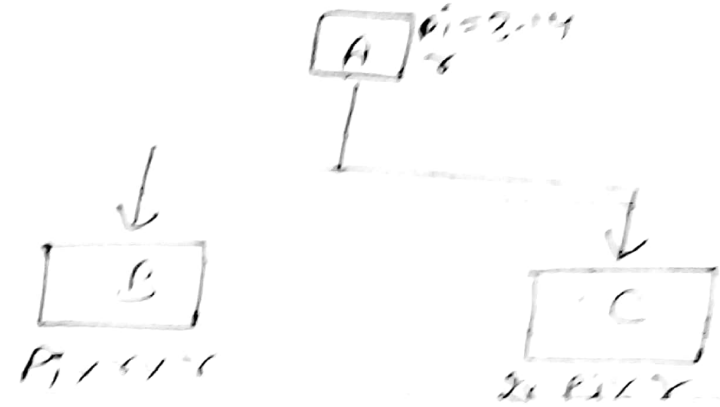
When a sub class derives ~~into~~ another sub class.

Q- WAP in C++ to implement multilevel inheritance.

```
#include<iostream.h>
#include<conio.h>
class A
{
  int l;
  public:
int getl()
  {
  cout<<"\n enter the value of l";
  cin>>l;
  return l;
  };

class B : public A
{
  int b;
  public;
int getb()
  {
  cout<<"\n enter the value of b";
  cin>>b;
  return b;
  };

class C : public B
  {
  public:
  void area()
  {
  cout<<"\n area = "<<l*b;
  }
};
```

```
void main()
{
    clrscr();
    C obj;
    obj.getb();
    obj.getb();
    obj.disp();
    getch();
}
```

3. Write a program in C++ to implement Hierarchical inheritance.



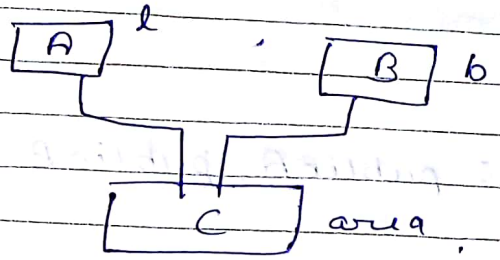
Where a superclass has derived at least two sub class.

```
//
//
    class A
    {
    public:
    float Pi;
    int r;
    void get_data()
    {
```

```
    pi=3.14;
    cout<<"\n enter the radius=";
    cin>>r;
  }
};

class B: public A
  {
    public:
    void area()
    {  get_data();
    cout<<"\n area of circle="<<pi*r*r;
    }
};

class C: public A
  {
    public:
    void circmf()
    {
      get_data();
  cout<<"\n circumference="<<2*pi*r;
    }
};

void main()
{
  clrscr();
  B obj;
  C obj1;
  obj.circum obj.area();
```

```
obj. circumf(c);
  getch();
}
```

*5. Multiple inheritance:

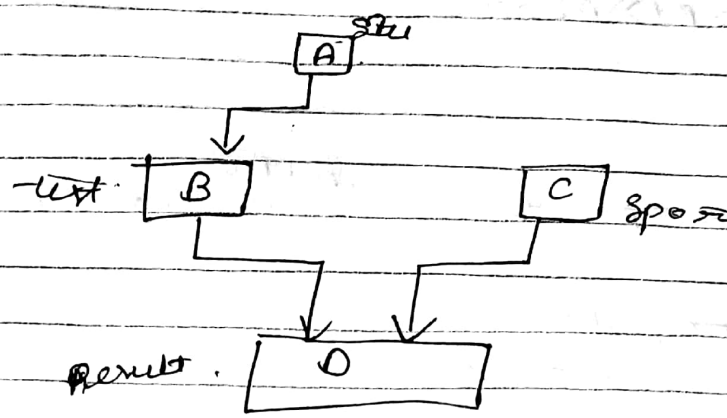


Two or more than two super class are responsible to drive a sub class.

```
#
#
  class A
  {
    public:
    {
     int l;
    void getl()
    {
    cout<<"\n enter l";
     cin>>l;
     }
  };
class B
{

  public:
  {
```

```
int b1;
void getb()
 {
  cout<<"\n enter b";
  cin>>b1;
 }
};

class C: public A, public B
{
 public:
 void area()
 {
 cout<<"\n area = "<<l*b1;
 }
};

void main()
{
  clrscr();
  C obj;
  obj.getl();
  obj.getb();
  obj.area();
  getch();
}
```

5. WAP in c++ to implement hybrid implementation



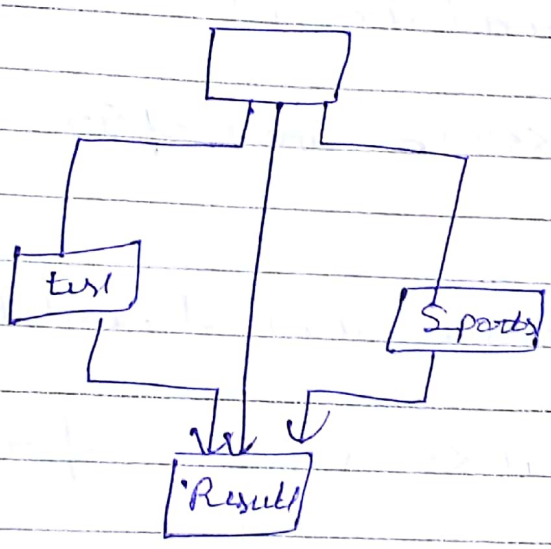
It is a combination of single multiple and multilevel inheritance.

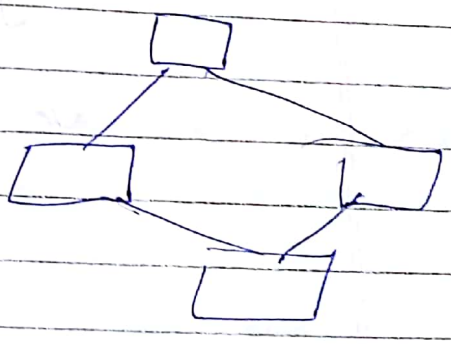
```
#
#
class stu
{
public:
int roll;
void get_roll()
{
cout<<"\n enter rollno";
cin>>roll;
}
};
class test : public stu { virtual
{
public;
int p1, p2, p3;
void get_marks()
{
```

```
cout<<"\n enter marks of P1,P2,P3";
cin>>P1>>P2>>P3;
}
};

class sports                    public virtual stu
{
  public:
  int sw;
  void get_sw()
  {
  cout<<"\n enter sports wight";
  cin>>sw;
  }
};

class Result: public test, public sports
{
  public:
  void disp()
  {
  cout<<"\n P1="<<
  cout<<"\n roll="<<roll;
cout<<"\n paper1="<<P1<<"\n"<<"paper2="
  <<P2<<"\npaper3="<<P3<<"\n";
cout<<"\n sports wight="<<sw;
  int t;
  t= P1+P2+P3;
  cout<<"\n total="<<t;

}
};
```

```
void main()
{
  clrscr();
  Result obj;
  obj.get_oral();
  obj.get_marks();
  obj.get_sw();
  obj.disp();
  getch();
}
```

6. WAP in C++ to implement multipath inheritance (diamond problem)





datduplicacy (Ambiguous)

Combination of all.

Due to these multiple paths the lowest class result get the duplicate value from the common base class (test and sports).

So to avoid such duplicacy we make the common base class as virtual base class, using the keyword virtual.

Now a single copy will be pass to the lowest class result.

Date
Page

Function Overloading

It is an example of compile time polymorphism where a group of functions with same name but diff: argument list.

Now consider the following examples:

```
#include <iostream.h>
#include <conio.h>
  class a
  {
   public:
   void area(int r)
    {
    cout<<"\n area of cir = "<<3.14 * r*r;
    }

   void area(int l, int b)
    {
    cout<<"\n area of rect = "<<l*b;
    }
  };

   void main()
    {
     int r, l, b;
     clrscr();
     a obj;
    cout<<"\n enter radius";
     cin>>r;
     cout<<"\n
     obj.area(r);
```

```
cout<<"\n enter l and b";
cin >> l >> b;
obj: area( l, b );

getch();
}
```

In the above example the class a have two functions both having same name area, one with a single integer argument for finding the area of circle and another with double integer ~~as~~ a argument for finding area of rect.

So we can say that one function area overload the another function area by same name but diff: argument list.

Operator Overloading: (compile time poly

"Assigning some special meaning or additional task to an operator is known as operator overloading."

For ex: The '+' operator concatinate two string in Java. So it has an additional task that means it is an example ~~of~~ of operator overloading.

The operator function:

The operator function is used to overload an operator in C++ programming language.

Syntax:

return type operator op (Args)

{

- - - - -

- - - - - //body of operator function

- - - - -

}

*) operator is a keyword.

*) op is the operator that is going to ove

Note: Following operator never be overlo
in C++.

1) ::

2) ?:

3) sizeof

4) member ~~access~~ selection

5) Type id

6) member selection with pointer to member. [.*

Q WAP in C++ to overload unary ++ operator (Use argument operator)

```
#include <iostream.h>
#include <conio.h>
class a
{
int x,y;
public:
void get_data()
{
x=10;
y=10;
}
void display()
{
cout<<"\n x="<<x<<"\t"<<y="<<y;
}
void operator ++()
{
x=++x;
y=++y;
}
};
void main()
{
a obj;
clrscr();
obj.get_data();
obj.display();
obj++;
```

```
obj.display();
getch();
}
```

Q WAP in C++ to overload binary + operator.

```
#
#
class a
{
int x,y;
public:
void get_data()
  {
   x = 10;
   y = 11;
  }
void display()
  {
  cout<<"\n x = "<<x<<"\t "<<"y = "<<y;
  }
a operator + (a obj)
  {
   a obj1;
   obj1.x = obj.x + x;
   obj1.y = obj.y + y;
   return obj1;
  }
};
  void main()
   {
```

```
a a1, a2, a3;
  clrscr();
  a1.get_data();
  a1.display();
  a2.get_data();
  a2.display();
  a3 = a1+a2;
  a3.display();
  getch();
}
```

* Virtual function (Run time polymorphism):

Function overriding -

A group of function with same name, same returntype and also the same no. of arguments its list is known as function overriding that will be solved by using virtual function.

Virtual function:

A function starts with the keyword virtual is known as virtual function.

Ex:

```
virtual void show()
{
  - - - -
  - - - -  //body of virtual function
  - - - -
}
```

NOTE: The main purpose of virtual function is to solve the function overriding problem and to

achieve run time polymorphism:

```
#include<iostream.h>
using namespace std;
  class a
{
  public:
  virtual void show() // virtual functi
  {
  cout<<"\n In super class";
  }
};
  class b : public a
  {
  public:
  void show()
  {
   cout<<"\n In Sub class";
  }
  };

main()
{
a *ptr; // base pointer, a pointer of
                type base class.
a  a1;
b  b1;
ptr = &a1; // assign the address of objid
                class a
ptr => show(); // call the show() functi
of base class a (base class)
```

ptr = &b1; // assign the address of object of class b

ptr -> show(); // call the show() function of class b (Sub class)

}

* Pure Virtual function (Do nothing function):

A Do nothing function or a function that initialized by zero is known as pure virtual function.

```
#include<iostream.h>
#include<conio.h>
  class a
  {
  public:
  virtual void show()=0; //pure virtual
                          function.
```

A pure virtual function (or abstract function in C++ is a virtual function for which we don't have implementation; we only declare it. A pure virtual function is declared by assigning 0 in declaration.

* ~~Contain~~ Containership (Nested class):

A class inside a class is known as nested class and this concept ~~with~~ is known as containership.

Now consider the following example

```
#include <iostream.h>
#include <conio.h>
    class a
{
    public:
    int l;
    void getl()
    {
    cout<<"\n enter l";
      cin>>l;
      }
    void area()
     {
       b obj;
    obj.getb();
   cout<<"\n area of rect="<<l*obj.b1;
     }
    class b
     {
      public:
       int b1;
      void getb()
          {
            cout<<"\n enter b";
             cin>>b1;
        }
      };
     };
```

```
void main()
{
  a ob;
  clrscr();
  ob.getb();
  ob.area();
  getch();
}
```

~~* This p~~: In the above example the class a consider as a container class because it consist a inner class b.

So a is a outer class, b is a inner class under the outer class a. So it will be refer as nesting of class or nested class.

* this pointer:

'this' is a keyword that marks as a implicit pointer to store the address (reference) of current class object.

Uses:

To access data member and member function using object reference.

Ex:

```
#include<iostream.h>
#include<conio.h>
  class a
 {
  public:
   int x;
 void show()
```

```
{
 x=10;
cout<<"\n hello";
 }
};

class b: public a
 {
  public:
  void display ()
  {
   cout<<"\n";
this -> show(); //call the member function
                  by this pointer.
cout<<"\n x=" <<this -> x; //access the
                  data number by this
                  function.
  }
};

void main()
 {
   b b1;
   clrscr();
b1.display ();
   getch();
}
```