cmd :- gcc —.cpp -lstdc++ -o —

* Output :-

cout << "Hello" ;

std :: cout << "Hello" << std:: endl;

* input :-

cin >> a ;

std :: cin >> a ;

Programme :

```
#include <iostream>
int main()
{
  std :: cout << "Hello" << std:: endl;
  int a;
  std :: cin << a;
  std :: cout << "a is value :" << a ;
}

// std -> standard library
// :: scope resolution operator -> for global access
// << insertion operator
// " " -> constant string.
```

* class and object:

```
#include<iostream>
using namespace std;
class car
{
    int num;
    public:
    void set_no(int n) //setter.
    {
        num = n;
    }
    int get_no()   //getter.
    {
        return num;
    }
};

int main()
{
    car Kiaseltos;
    Kiaseltos.set_no(2154);
    cout<<" my car no: "<<Kiaseltos.get_no();
}
```

Square :-

```
#include<iostream>
using namespace std;
class square
{
    int n;
    public:
    void set_n(int val)
    {
        n = val;
    }
    int get_n()
    {
        return n*n;
    }
};
int main()
{
    square sq;
    sq.set_n(3);
    cout<<" square of "<<sq.get_n()<<" is "
        <<sq.dosquare()<<endl;
}
```

```
int dosquare()
{
    return n*n;
}
```

* Constructor - default.

```
# include <iostream>
using namespace std;
class cons
{
  public:
  cons()
  {
    cout << "\n constructor called ....";
  }
};

int main()
{
  cons c1;
}
// 1. class name function.
// 2. constructor can not return.
// 3. no need, to call, invoke automatically when
      object created.
```

* parameterized constructor.

```
cons.(int n)
{
  balance = n;
  cout << "\n\n Balance :" << n;
};
int main()
{
  cons a(100), c2(200), c3(0), c4(1000);
}
// use in initialization.
```

4 // bank programme in c plus plus.

```
#include<iostream>
using namespace std;
class bank.
{
  int bal;
  public:
    int showbalance().
    {
      return bal;
    }

    int deposit (int deposit).
    {
      bal = bal + deposit;
      return bal;
    }

    int withdraw (int withdraw).
    {
      if (bal >= withdraw).
      {
        bal = bal - withdraw;
        return bal;
      }
      else
      {
      cout << " In Abe scale pahele balance to
                dekh !!!!!";
      return 1;
      }
    }
};
```

```
int main().
{
 int ch, dp, wt;
 bank por;
 do.
 {
  cout<<"\n 1. balance..\n"<<"2. Deposit...\n"
      <<"3. withdraw.\n"<<"4. Exit....\n";
  cout<<" select Number :: ";
  cin >> ch;
  switch (ch).
  {
    case 1:
    cout<<"\n your acount balance:"
        << por.showbalance();
    break;

    case 2:
    cout<<"\n Enter your Deposit :: ";
    cin >> dp;
    cout<<"\n your deposit ::"<< dp;
    cout<<"\n your bank balance ::"
        << por. deposite(dp);
    break;

    case 3:
    cout<<"\n Enter your withdraw :: ";
    cin >> wt;
    cout<<"\n your withdraw ::"<< wt;
    cout<<"\n your bank balance ::"
        << por. withdraw(wt);
    break;
```

```
case 4:
    cout << "\n Thank you for visiting Vaishali's
                    Bank ....... ";
    break;

default:
cout << "\n your choice is wrong .... ";
break;
}

} while(ch!=4);

}
```

* Destructor. ~constr().

syntax :-

~constr().
{
cout << "In Destructor called ....";
}

* Constructor overloading :-

1). Cons() // default con.
{
cout << " In default Constructor called... ";
}

2) Cons(int n, int m). // two int parm. Cons.
{
cout << " ..... " << n << m;
}

3) Cons. (double n). // one double parm.
{
cout << " ..... " << n;
}

4). cons. (int n) // one int parm.
{
cout << " - - " << n;
}
~ cons() // destructor. ~ field.
{
cout << " ..... ";
}

int main()
{
Cons C1, C2(2,21), C3(3.14), C4(4);
}

*. String મા space વાળો Name લેવા માટે cin ની બદલે

getline(cin,str);

```
#include<iostream>
using namespace std;
class Functionover
{
  pubic:
    Functionover();  // con. declaration.
    void sum(int, int);
    void sum(int, int, int);
    void sum(int double, double);
};

Functionover::functionover() // con. definition
{
    cout.
};
void functionover::sum(int a, int b)
{
    cout.
};
void functionover::sum(int a, int b, int c)
{
    cout.
};
void functionover::sum(double a, doubleb
{
  cout.
};

int main()
{
  functionover obj1;
  obj1.sum(12,32);
  obj2.sum(12,34,43); obj1.Sum(3.2,3.6);
```

Buffer clear = fflush(stdin);

* Allways use same variable.

this -> BAL = BAL;

* Copy constructor.

```
#include<iostream>
using namespace std;
class Findage
{
    int age;
    public:
    Findage(int n). // constructor
    {
        age = n;
        cout << "In Age :" << age;
    }
    Findage(Findage &New_age) // copy constructor
    {
        age = New_age.age;
        cout << "In Age :" << age;
    }
};
int main()
{
    Findage person1(20);
    Findage Person2(Person1);
    return 0;
}
```

# object as argument and object as return

```
#include <iostream>
using class namespace std;
class Example
{
    int a, b;
    public: s
void set_ab (int x, int y)
{
    a = x; b = y;
}
void get_ab ().
{
    cout << " in a : " << a;
    cout << " in b : " << b;
}

Example add (Example E1, Example E2)
{
    Example E_temp;

    E_temp.a = E1.a + E2.a;
    E_temp.b = E1.b + E2.b;
    return E_temp;
};

int main ()
{
    Example E1, E2, E3;
    E1.set_ab (10, 20);
    E2.set_ab (11, 21);
    E3 = E3.add (E1, E2);
    E3.get_ab ();
```

Date
Page

*. Friends Function.

```
#include <iostream>
using namespace std;
class Box
{
  private:
     int length;
  public:
     Box() : length(0)
     {
     }
    private:
     friend int printlength(Box);
};
int printlength(Box b)
{
   b.length += 10;
   return b.length;
}
int main()
{
  Box b;
  cout << "length of Box : " << printlength(b)
                                << endl;
   return 0;
}
```

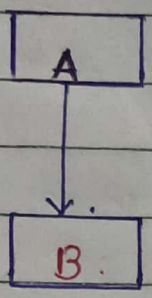
* Friends Function.

```
# include <iostream>
using name space std;
class Box.
{
   private:
      int length;
   public.:
      Box() : length(0).
      {
      }
      Private:
       friend int printlength(Box);
};
int printlength(Box b)
{
    b.length += 10;
    return b.length;
}
int main()
{
   Box b;
   cout << " Length of Box: " << printlength(b) << endl;
   return 0;
}
```

# * Inheritance *

(વારસો).

1) single inheritance.




Base / parent.

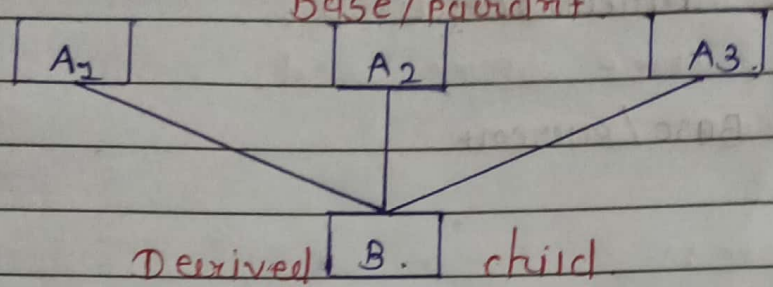
Derived / child.

=> 1). Human નામનો કલાસ (class).
2). Employee class.
Employee class. inherit human class.

=>. 1). Animal. class.
2). Dog. class.
Dog class. inherit. human class.

=> 1). Father class.
2) Son class.
Son class inherit. Father class.

2) Multiple inheritance.



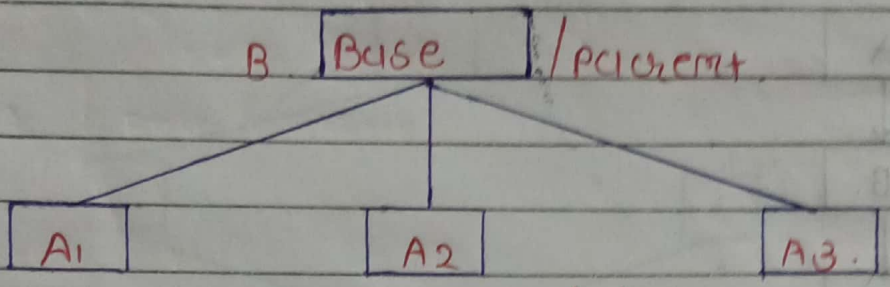

=>. A1, A2, A3 are three Banks.
B as customer.

=>. A1 - Human, A2 - Employee, A3 - Father.
B - Nurse.

=>. A1 = Liquid A2 = Fuel.
B1 = petrol.

=>. A1 = Father A2 = Mother.
B1 = child

3) Hierarchical inheritance.




Derived / child.

=> B = Animal.
A1 = Cat A2 = Dog A3 = Cow.

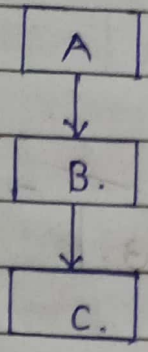
=> B = science class.
A1 = physics, A2 = chemistry. A3 = Biology

=> B = Diamond.
A1 = Ruby diamond. A2 = Blue stone.

=> B = Fuel.
A1 = Petrol A2 = Diesel

=> B = Bank. RBI
A1 = BOI A2 = Syndicate Bank A3 = SBI.

4) Multilevel inheritance.

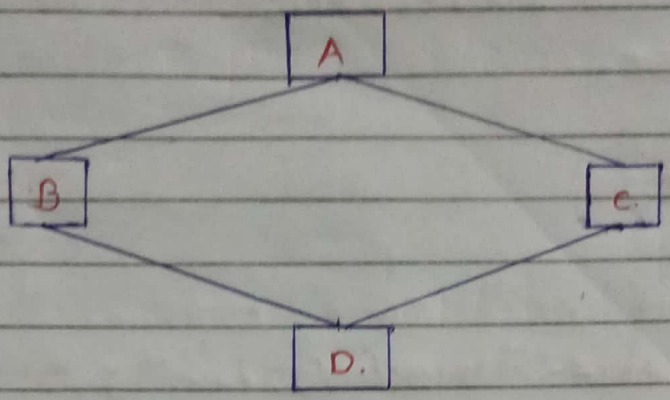



=>. A = grandfather. B = father. C = son.

=>. A = Company. B = Box. C = shipment.
Box weight

=> A = car B = Tata C = Harrier.

5) Hybrid inheritance.




=> A = Animal.
B = Herbivores. (శాకాహారి)
C = Non veg. (మాంసాహారి).
D = Dog, cat.

=> A = SAMSUNG.
B = OLed display. C = Component.
D = Mobail.

=> A = Car.
B = Normal. C = Racing
D = Famous.

# Single inheritance.

```
// single inheritance.
#include<iostream>
using namespace std;
class Base.
{
  public:
  Base().
  {
    cout << "\n Base cons...." ;
  }
  ~Base().
  {
    cout << "\n Bas dest...";
  }
};

class derived : Pulic. Base.
{
  Pulic:
  Derived().
  {
    cout << "\n Derived cons...";
  }
  ~Derived().
  {
    cout << "\n Derived dest....";
  }
};

int main().
{
  Derived d;
}
```

```
Base cons.....
Derived cons....
Derived dest....
Base dest....
```

# Multiple inheritance

```
// Multiple inheritance
#include<iostream>
using namespace std;
class Base
{
    public:
    Base()
    {
        cout<<"\nBcons....";
    }
    ~Base()
    {
        cout<<"\n Base dest...";
    }
};
class Base1
{
    public:
    Base1()
    {
        cout<<"\n Base1 cons....";
    }
    ~Base1()
    {
        cout<<"\n Base1 dest...";
    }
};
```

```
class Derived : public Base , public Base1
{
public:
Derived()
{
    cout << "\n Derived cons...";
}
~Derived()
{
    cout << "\n Derived dest...";
}
};

int main()
{
    Derived d1;
}
```

* Output *

```
Base cons....
Base1 cons...
Derived cons...
Derived dest....
Base1 dest....
Base dest.....
```

# * Hierarchical inheritance *

```
// Hierarchical inheritance
#include <iostream>
class Base
{
  public:
  Base()
  {
      cout << " Base con..... ";
  }
  ~Base()
  {
      cout << " Base dest ..... ";
  }
};

class derived : pubic Base
{
   pubic:
   Derived()
   {
      cout << " Derived con..... ";
   }
   ~Derived()
   {
      cout << " Derived dest..... ";
   }
};
```

```
class Derived1 : Pubic Base
{
    Pubic :
    Derived1()
    {
        cout << " Derived1 cons....";
    }
    ~Derived1(),
    {
        cout << " Derived1 destructor ....";
    }
};

int main()
{
    Derived c1;
    Derived d1;
}
```

* Output *

```
Base cons...
Derived cons...
Base cons....
Derived1 cons...
Derived1 dest...
Base dest...
Derived dest...
Base dest....
```

# * Multi level inheritance *

```
// Multi level inheritance.
# include <iostream>
using namespace std;
class Base.
{
   public:
   Base()
   {
      cout << " In Base con...";
   }
   ~Base()
   {
      cout << " In Base dist...";
   }
};

class Base1 : public. Base.
{
   public:
   Base1()
   {
      cout << " In Base 1 con....";
   }
   ~Base1()
   {
      cout << " In Base 1 dest...";
   }
};
```

```
class Derived : public Base1
{
  public:
  Derived()
  {
    cout << " in Derived con...";
  }
  ~Derived().
  {
    cout << " in Derived. dest...";
  }
};

int main().
{
  Derived c1;
}
```

* output *.

```
Base. con...
Base1 con...
Derived con...
Derived dest...
Base1 dest...
Base dest...
```

# * Hybrid inheritance *

```
// hybrid inheritance.
#include <iostream>
using namespace std;
class Base.
{
  public:
  Base()
  {
    cout << "in Base. cons..";
  }
  ~Base().
  {
    cout << "in Base dest..";
  }
};

class Base1 : public Base.
{
   Public:
   Base1()
   {
     cout << "Base1 cons..";
   }
   ~Base1()
   {
     cout << "Base1 dest..";
   }
};
```

```
class Base2 : public Base
{
  public:
  Base2().
  {
    cout << "\n Base2 cons....";
  }
  ~Base2().
  {
    cout << "\n Base2 dest....";
  }
};

class Derived : public Base1; public Base2
{
    Public:
    Derived().
    {
      cout << "\n Derived. con....";
    }
    ~Derived()
    {
      cout << "\n Derived dest....";
    }
};

int main().
{
   Derived d;
}
```

Date
Page

## * output *

Base con....
Base1 con...
Base con...
Base 2 con...
Derived con....
Derived dis....
Base 2 dis....
Base dis....
Base1 dis...
Base dis....

*. // run time polymorphism // function overridding

```
#include<iostream>
using namespace std;
class Base // interface
{
    public:
    virtual void display().
    {
        cout << " Base class is invoked " << endl;
    }
};

class Derivied : public Base.
{
    public:
    void display().
    {
        cout << " Derived class is invoked " << endl;
    }
};
int main().
{
    Base *p, b;
    Derived d;
    p = &b ;  // Base
    p -> display();  // -> point to.
    p = &d;          // derived.
    p -> display();

},
```

```
//
#include <iostream>
using namespace std;
class Base // abstract class // interface
{
public:
    virtual void display() = 0; // pure virtual
};

class Derived : public Base
{
public:
    void display()
    {
        cout << "Derived class is invoked" << endl;
    }
};

int main()
{
    Derived d;
    d.display();
}
```

```
//
#include <iostream>
using namespace std;
class shape // abstract class
{
  protected:
  int width;
  int hight;

public:
    virtual int getArea()=0; //pure virtual
    //
    void setwidth(int w)
    {
      width = w;
    }
    void setHeight(int h)
    {
      height = h;
    }
};
int
class Rectangle : public shape
{
  public:
    int getArea()
    {
    return (width * hight);
    }
};
```

```
class Triangle : public shape.
{ public:
  int getAreas()
  {
    return (width = height * 1/2;
  }
};

int main()
{
    Rectangle Rect;
    Triangle Tri;

    Rect.setWidth (5);
    Rect.setHeight (7);
    cout << " Total Rectangle. area : "
         << Rect.getArea() << endl;

   Tri.setwidth (5);
   Tri.setwidth (7);
   cout << " Total triangle area: "
        << Tri.getArea() << endl;
   return 0;
}
```

# Static

```
//
#include<iostream>
using namespace std;
class Bank
{
    int accno, balance;
    string myname;
    public:
    static int count;
    Bank(int accno, int balance, string myname)
    {
        this->accno = accno;
        this->balance = balance;
        this->myname = myname;
        count++;
    }
    void getdata()
    {
        cout<<"In"<<myname<<"-"<<accno
            <<"-"<<balance;
    }
};

int Bank::count;
int main()
{
    Bank q(101, 1500, "varushaleh"),
         M(102, 170, "Keyaar"),
         q1(101, 1500, "Varushaleh");
         M1(102, 170, "Neet");
    q.getdata();
    m.getData();
```

```
cout << "\n No. of Accounts : " << Bank::
                                    count;
return 0;
}

// Static variable must be declare globally
// static variable initial value is 0.
// static variable cannot be private.
```

## * Temp. overloading *.

```
# include <iostream>
using namespace std;
template <class T>
void display (T t1)
{
    cout << " Displaying Template : " << t1 <<
                                    "\n";
}
template <class T1 , class T2 >
void display (T1 t1 , T2 t2 )
{
   cout << " Displaying 2 Template : "
        << t1 << "\t"
        << t2 << "\n" ;
}
// void display (int t1).
// {
//    cout << " Explicitly display : "
//         << t1 << "\n";
// }
```

```
int main()
{
    display (200);
    display (12.40);
    display ('G');
    display ('G', 1.25);
    display ('x', 25);
    display (25, 1.25);
    return 0;
}
```

# * File *

```
//ofstream : stream class to write on files.
// ifstream : stream class to read from files
// fstream : stream class to both read and
              write from/to file.

#include<iostream>
#include<fstream>
using namespace std;
int main().
{
   ofstream filestream("file1.txt");
  if (filestream.is_open()).
  {
   filestream << "welcome to skillcode";
   filestream << "\n after execution";

   filestream.close();
  }
  else.
  {
   cout << "File opening is fail.";
  }
  return 0;
}
```

```
# include <iostream>
# include <fstream>
using namespace std;
int main()
{
    String sorg;
    if stream Filestream("File1.txt");
    if (Filestream.is_open())
    {
        while (getline (filestream, sorg))
        {
            cout << sorg << endl;
        }
        Filestream.close();
    }
else.
{
    cout << "File opening is fail." << endl;
}
return 0;
}.
```

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    char input[75];
    ofstream os;
    os.open("file3.txt");
    cout << "Writing to a text file:";
    cout << "Please Enter your name:";
    cin.getline(input, 100);
    os << input << endl;
    cout << "please Enter your contact:";
    cin >> input;
    cin.ignore();
    os << input << endl;
    os.close();
    ifstream is;
    string line;
    is.open("file3.txt");
    cout << "Reading from a text file:";
    while (getline(is, line))
    {
        cout << line << endl;
    }
    is.close();
    return 0;
}
```

Buffer

// exception handling.

```
#include <iostream>
using namespace std;
double zerodivision (int x, int y).
{
    if (y==0)
    {
        throw "division by zero!";
    }
    return (x/y);
}
int main().
{
    cout << "in begin main";
    int a = 77;
    int b = 0;
    double c = 0;
    try {
        c = zerodivision (a,b);
        cout << c << endl;
    }
    catch ( const char* message)
    {
        cerr << "in if" << message << endl;
    }
    cout << "in end main";
    return 0;
}
```

```
// exception handling
#include <iostream>
using namespace std;
int main()
{
    int p,c,m, error = 0;
    string name;
    do
    {
        try
        {
            cout << " enter student name : " << endl;
            cin >> name;
            cout << " enter physics marks : " << endl;
            cin >> p;
continue - 3
sub.        if (!p>=0 && p<=100)
            {
                throw (p);
            }
            error = 0;
    }
    catch (int e)
    {
        cout << "Invalid marks ! " << endl;
        error = 1;
    }
    while (error);
}
```