// C++ program to demonstrate public

// access modifier

#include<iostream.h>

// class definition

class Circle

{

    public:

        double radius;

        double  compute\_area()

        {

            return 3.14\*radius\*radius;

        }

};

// main function

int main()

{

    Circle obj;

    // accessing public datamember outside class

    obj.radius = 3.5;

    cout << "Radius is: " << obj.radius << "\n";

    cout << "Area is: " << obj.compute\_area();

    return 0;

}

**Output:**

Radius is: 3.5

Area is: 38.465

-🡪 encapsulation

#include <iostream>

using namespace std;

// declaring class

class Circle {

    // access modifier

private:

    // Data Member

    float area;

    float radius;

public:

    void getRadius()

    {

        cout << "Enter radius\n";

        cin >> radius;

    }

    void findArea()

    {

        area = 3.14 \* radius \* radius;

        cout << "Area of circle=" << area;

    }

};

int main()

{

    // creating instance(object) of class

    Circle cir;

    cir.getRadius(); // calling function

    cir.findArea(); // calling function

}

**Output**

Enter radius

5

Area of circle=78.5

-🡪

|  |
| --- |
| // C++ program to illustrate call by value  #include <iostream>  using namespace std;  // Function Prototype  void swap(int x, int y);    // Main function  int main()  {  int x = 10, y = 20;  cout << "In the Caller:\n";  cout << "x = " << x << " y = " << y << endl;  // Pass by Values  swap(x, y);  }    // Swap functions that swaps  // two values  void swap(int x, int y)  {  int t;    t = x;  x = y;  y = t;    cout << "In the Function:\n";  cout << "x = " << x << " y = " << y << endl;  }  **Output**  In the Caller:  x = 10 y = 20  In the Function:  x = 20 y = 10 |

-🡪// C++ program to illustrate Call by Reference

#include <iostream>

using namespace std;

// Function Prototype

void swap(int\*, int\*);

// Main function

int main()

{

int x = 10, y = 20;

cout << "In the Caller:\n";

cout << "x = " << x << " y = " <<y << endl;

// Pass reference

swap(&x, &y);

}

// Function to swap two variables

// by references

void swap(int\* x, int\* y)

{

int t;

t = \*x;

\*x = \*y;

\*y = t;

cout << "In the Callee:\n";

cout << "x = " << \*x << " y = " << \*y << endl;

}

**Output**

In the Caller:

x = 10 y = 20

In the Callee:

x = 20 y = 10

-🡪Inline functions

#include <iostream.h>

inline int cube(int s)

{

return s \* s \* s;

}

int main()

{

    cout << "The cube of 3 is: " << cube(3) << "\n";

    return 0;

}

**Output**

The cube of 3 is: 27

-🡪// CPP Program to demonstrate Default Arguments

// A function with default arguments,

// it can be called with

// 2 arguments or 3 arguments or 4 arguments.

#include <iostream.h>

int sum(int x, int y, int z = 0, int w = 0) //assigning default values to z,w as 0

{

    return (x + y + z + w);

}

int main()

{

    // Statement 1

    cout << sum(10, 15) << endl;

    // Statement 2

    cout << sum(10, 15, 25) << endl;

    // Statement 3

    cout << sum(10, 15, 25, 30) << endl;

    return 0;

}

**Output**

25

50

80

-🡪function overloading

#include <iostream.h>

void add(int a, int b)

{

  cout << "sum = " << (a + b);

}

void add(double a, double b)

{

    cout << endl << "sum = " << (a + b);

}

void main()

{

    add(10, 2);

    add(5.3, 6.2);

 }

**Output**

sum = 12

sum = 11.5

🡪function overriding

#include <iostream>

using namespace std;

class A {

public:

void display() {

cout<<"Base class";

}

};

class B:public A {

public:

void display() {

cout<<"Derived Class";

}

};

int main() {

B obj;

obj.display();

return 0;

}

**Output**

Derived Class

🡪recursive functions

#include <iostream.h>

void add(int a, int b)

{

  cout << "sum = " << (a + b);

}

void add(double a, double b)

{

    cout << endl << "sum = " << (a + b);

}

void main()

{

    add(10, 2);

    add(5.3, 6.2);

 }

**Output**

sum = 12

sum = 11.5

🡪default constructor

// CPP program to illustrate the concept of Constructors

#include <iostream>

using namespace std;

class construct {

public:

    int a, b;

    // Default Constructor

    construct()

    {

        a = 10;

        b = 20;

    }

};

int main()

{

    // Default constructor called automatically when the object is created

    construct c;

    cout << "a: " << c.a << endl << "b: " << c.b;

    return 0;

}

**Output**

a: 10

b: 20

🡪parameterized constructor

#include<iostream>

using namespace std;

class Point {

private:

    int x, y;

public:

    // Parameterized Constructor

    Point(int x1, int y1)

    {

        x = x1;

        y = y1;

    }

    int getX() { return x; }

    int getY() { return y; }

};

int main()

{

    // Constructor called

    Point p1(10, 15);

    // Access values assigned by constructor

    cout << "p1.x = " << p1.getX()

         << ", p1.y = " << p1.getY();

    return 0;

}

**Output**

p1.x = 10, p1.y = 15

🡪constructor and destructor

// C++ program to demonstrate the execution of constructor and destructor when //multiple objects are created

#include <iostream>

using namespace std;

class Test {

public:

    // User-Defined Constructor

    Test() { cout << "\n Constructor executed"; }

    // User-Defined Destructor

    ~Test() { cout << "\n Destructor executed"; }

};

main()

{

    // Create multiple objects of the Test class

    Test t, t1, t2, t3;

    return 0;

}

**Output**

Constructor executed

Constructor executed

Constructor executed

Constructor executed

Destructor executed

Destructor executed

Destructor executed

Destructor executed

🡪single level inheritance

#include <iostream>

using namespace std;

class base //single base class

{

public:

int x;

void getx()

{

cout << "Enter the value of x = "; cin >> x;

}

};

class derive : public base //single derived class

{

private:

int y;

public:

void gety()

{

cout << "Enter the value of y = "; cin >> y;

}

void product()

{

cout << "Product = " << x \* y;

}

};

int main()

{

derive a; //object of derived class

a.getx();

a.gety();

a.product();

return 0;

} //end of program

**Output**

Enter the value of x = 1

Enter the value of y = 4

Product = 4

**🡪C++ Multilevel Inheritance**

#include <iostream>

using namespace std;

class base //single base class

{

public:

int x;

void getx()

{

cout << "Enter value of x= "; cin >> x;

}

};

class derive1 : public base // derived class from base class

{

public:

int y;

void gety()

{

cout << "\nEnter value of y= "; cin >> y;

}

};

class derive2 : public derive1 // derived from class derive1

{

private:

int z;

public:

void getz()

{

cout << "\nEnter value of z= "; cin >> z;

}

void product()

{

cout << "\nProduct= " << x \* y \* z;

}

};

int main()

{

derive2 a; //object of derived class

a.getx();

a.gety();

a.getz();

a.product();

return 0;

} //end of program

**Output**

Enter value of x= 1

Enter value of y= 4

Enter value

🡪multiple inheritance

#include<iostream>

using namespace std;

class A

{

public:

int x;

void getx()

{

cout << "enter value of x: "; cin >> x;

}

};

class B

{

public:

int y;

void gety()

{

cout << "enter value of y: "; cin >> y;

}

};

class C : public A, public B //C is derived from class A and class B

{

public:

void sum()

{

cout << "Sum = " << x + y;

}

};

int main()

{

C obj1; //object of derived class C

obj1.getx();

obj1.gety();

obj1.sum();

return 0;

} //end of program

**Output**

enter value of x: 4

enter value of y: 5

Sum = 9

🡪hieracheal inhertance

#include <iostream>

using namespace std;

class A //single base class

{

public:

int x, y;

void getdata()

{

cout << "\nEnter value of x and y:\n"; cin >> x >> y;

}

};

class B : public A //B is derived from class base

{

public:

void product()

{

cout << "\nProduct= " << x \* y;

}

};

class C : public A //C is also derived from class base

{

public:

void sum()

{

cout << "\nSum= " << x + y;

}

};

int main()

{

B obj1; //object of derived class B

C obj2; //object of derived class C

obj1.getdata();

obj1.product();

obj2.getdata();

obj2.sum();

return 0;

} //end of program

**Output**

Enter value of x and y:

4 1

Product= 4

Enter value of x and y:

4 5

Sum= 9

🡪hybrid inheritance

#include <iostream>

using namespace std;

class A

{

public:

int x;

};

class B : public A

{

public:

B() //constructor to initialize x in base class A

{

x = 4;

}

};

class C

{

public:

int y;

C() //constructor to initialize y

{

y = 5;

}

};

class D : public B, public C //D is derived from class B and class C

{

public:

void sum()

{

cout << "Sum= " << x + y;

}

};

int main()

{

D obj1; //object of derived class D

obj1.sum();

return 0;

}

**Output**

Sum= 9