**Classes & Objects :**

Note: Method is a function written inside a class.

* Public can be accessed by main function and others.
* While private is only accessed inside the class.

#include <iostream>

#include <string>

using namespace std;

class Teacher{

    //properties or attributes.

private:

    double salary;

public:

    string name;

    string dep;

    string subject;

    // methods or member functions.

    void changeDep(string newDep){

        dep = newDep;

    }

    void setSalary(double s){

    //We assign salary value to s value.

        salary = s;

    }

   double getSalary(){

        return salary;

    }

};

int main(void){

    Teacher t1;

    Teacher t2;

    //For t1.

    t1.name = "Aslam";

    t1.subject = "Programming";

    t1.dep = "IT";

    t1.setSalary(35000);

    //For t2.

    t2.name = "Akram";

    t2.subject = "OOP";

    t2.dep = "CS";

    t2.setSalary(45000);

    //For t1.

    cout << t1.name << endl;

    cout << t1.subject << endl;

    cout << t1.dep << endl;

    cout << t1.getSalary << endl;

    //For t2.

    cout << t2.name << endl;

    cout << t2.subject << endl;

    cout << t2.dep << endl;

    cout << t2.getSalary << endl;

    return 0;

}

Note: Here, we use getSalary and setSalary to access the Salary value bcz Salary is a private class we access private classes by using public methods or functions.

**Topics in OOPS :**

Encapsulation.

Abstration.

Inheritence.

Polymorphism.

**Encapsulation**

Def: wrapping up of data & members functions in a single unit called class.

#include <iostream>

#include <string>

using namespace std;

class Account{

private:

    double balance;

    string password;

public:

    int accountID;

    string username;

//Setter function.

    void getBalance(double b){

        balance = b;

    }

//Getter function.

    double showBalance(){

        return balance;

    }

};

int main(void){

    Account a1;

    a1.accountID = 457;

    a1.username = "Arsal";

    a1.getBalance(4.5);

    cout << a1.accountID << endl;

    cout << a1.username << endl;

    cout << a1.showBalance() << endl;

    return 0;

}

#include <iostream>

#include <string>

using namespace std;

class Rectangle{

private:

int lenght;

int breath;

public:

    //Setter

    void setLenght(int len){

        lenght = len;

    }

    //Setter

    void setBreath(int brth){

        breath = brth;

    }

    //Getter

    int getlenght(){

        return lenght;

    }

    //Getter

    int getbreath(){

        return breath;

    }

    int getArea(){

        return lenght \* breath;

    }

};

int main(void){

    Rectangle rec1;

    rec1.setLenght(7);

    rec1.setBreath(7);

    cout << "Lenght is: " << rec1.getlenght() << endl;

    cout << "Breath is: " << rec1.getbreath() << endl;

    cout << "Area is: " << rec1.getArea() << endl;

    return 0;

}

**Constructor**

* Parameterized constructor
* #include <iostream>
* #include <string>
* using namespace std;
* class Rectangle{
* private:
* int lenght;
* int breath;
* public:
* Rectangle(int len, int brth){
* lenght = len;
* breath = brth;
* }
* //Getter
* int getlenght(){
* return lenght;
* }
* //Getter
* int getbreath(){
* return breath;
* }
* int getArea(){
* return lenght \* breath;
* }
* };
* int main(void){
* Rectangle rec1(7, 7);
* cout << "Lenght is: " << rec1.getlenght() << endl;
* cout << "Breath is: " << rec1.getbreath() << endl;
* cout << "Area is: " << rec1.getArea() << endl;
* return 0;
* }
* Default constructor/non-parameterized.
* #include <iostream>
* #include <string>
* using namespace std;
* class Student{
* int rollno;
* string name;
* int stdID;
* int semester;
* public:
* Student(){
* cout << "Enter your name " << endl;
* cin >> name;
* cout << "Enter your Rollno " << endl;
* cin >> rollno;
* cout << "Enter your ID " << endl;
* cin >> stdID;
* cout << "Finally enter your Class " << endl;
* cin >> semester;
* }
* void display(){
* cout << name << endl;
* cout << rollno << endl;
* cout << stdID << endl;
* cout << semester << endl;
* }
* };
* int main(void){
* Student s;
* s.display();
* return 0;
* }

Note: this is a special pointer in cpp that points to the current object.

* this pointer

#include <iostream>

#include <string>

using namespace std;

class Test{

private:

    int x;

public:

    void setX(int x){

//left side shows obj and right side shows parenthesis.

        this->x = x;

    }

    void getX(){

        cout << "X= " << x << endl;

    }

};

int main(void){

    Test obj;

    obj.setX(25);

    obj.getX();

    return 0;

}

* Copy constructor

Default copy constructor:

#include <iostream>

#include <string>

using namespace std;

class Test{

private:

    int x;

public:

    void setX(int x){

        this->x = x;

    }

    void getX(){

        cout << "X= " << x << endl;

    }

};

int main(void){

    Test obj1;

    obj1.setX(25);

    obj1.getX();

    Test obj2(obj1); //default copy constructor calls.

    obj2.getX();

    return 0;

}

Custom copy constructor

#include <iostream>

#include <string>

using namespace std;

class Teacher{

private:

    double salary;

public:

    string sub;

    string dept;

    string name;

//Parameterized constructor.

    Teacher(string name, string dept, string sub, double salary){

        this->name = name;

        this->dept = dept;

        this->sub = sub;

        this->salary = salary;

    }

//Copy constructor

//Here, we pass-by-referrence value which means original value of obj is passed not the copy of obj.

    Teacher(const Teacher& Orgobj){

        this->name = Orgobj.name;

        this->dept = Orgobj.dept;

        this->sub = Orgobj.sub;

        this->salary = Orgobj.salary;

    }

    void setSalary(double s){

        salary = s;

    }

    double getSalary(){

        return salary;

    }

    void changeDept(string newDept){

        dept = newDept;

    }

    void showInfo(){

        cout << "Name: " << name << endl;

        cout << "Dept: " << dept << endl;

        cout << "Subject: " << sub << endl;

        cout << "Salary: " << getSalary() << endl;

    }

};

int main(void){

    Teacher t1("Arsal","IT", "OOP", 45000);

    Teacher t2(t1);

    t2.showInfo();

    return 0;

}

**Shallow and Deep Copy**

Shallow Copy:

Note: It copies the reference of the original object.

Note: The main diff b/w them is that shallow copy happens when we don’t use copy constructor. Meanwhile, Deep constructor is when we use copy constructor.

Shallow Copy:

An object is created by simply copying the data of all variables of the original object.



#include <iostream>

using namespace std;

class Boxes{

    private:

        int lenght;

        int breadth;

        int height;

        int \*p;

    public:

    void getDimension(int lenght1, int breadth1, int height1, int x){

        lenght = lenght1;

        breadth = breadth1;

        height = height1;

        p = new int[20];//Dynamically Allocated.

        \*p = x;

    }

    void showDimension(){

        cout << "Lenght: " << lenght << endl;

        cout << "Breadth: " << breadth << endl;

        cout << "Height: " << height << endl;

        cout << "Address of P: " << p << endl;

    }

    ~Boxes(){

        delete []p;

    }

};

int main(void){

    Boxes b1, b2;

    b1.getDimension(45, 33, 10, 50);

    b1.showDimension();

    b2 = b1;

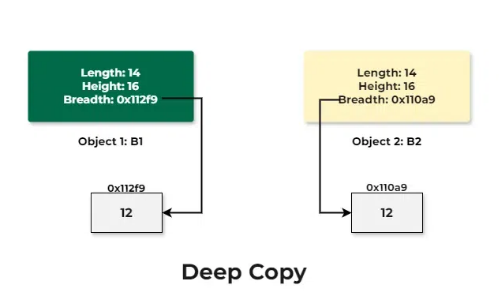
    b2.showDimension();

    return 0;

}

Deep Copy:

An object is created by copying data of all variables, and it also allocates similar memory resources with the same value to the object.



#include <iostream>

using namespace std;

class Boxes{

    private:

        int lenght;

        int \*breadth;

        int height;

    public:

    Boxes(){

        breadth = new int;

    }

    void getDimension(int lenght1, int breadth1, int height1){

        lenght = lenght1;

        \*breadth = breadth1;

        height = height1;

    }

    void showDimension(){

        cout << "Lenght: " << lenght << endl;

        cout << "Breadth: " << breadth << endl;

        cout << "Height: " << height << endl;

    }

    //Copy constructor for implementing deep copy.

    Boxes(Boxes& obj){

        lenght = obj.lenght;

        breadth = new int;

        \*breadth = \*(obj.breadth);

        height = obj.height;

    }

    //Destructor.

    ~Boxes(){

        delete []breadth;

    }

};

int main(void){

    Boxes b1;

    b1.getDimension(45, 33, 10);

    b1.showDimension();

    Boxes b2(b1);//Copy constructor.

    b2.showDimension();

    return 0;

}

Example of Shallow and Deep Constructor.

#include <iostream>

#include <cstring>//We use this library for the strcpy() function.

#include <string>

using namespace std;

class joinName{

private://\* shows it's a pointer to char array.

    char \*name;

public:

//Simple Constructor.

    joinName(const char \*str){

        name = new char[20];//Dynamically allocated memory.

        strcpy(name, str);//Copies the name into str.

    }

//Copy constructor.

    joinName(const joinName &obj){

        name = new char[20]; //Dynamic Allocation.

        strcpy(name, obj.name);//Copies the contents of name into obj.name.

    }

    void concatenate(const char\*str){

        strcat(name, str);//name should be added into the str.

    }

    //Destructor

    ~joinName(){

        delete []name;

    }

    void display(){

        cout << "Full name is: " << name << endl;

    }

};

int main(void){

    //Shallow copy.

    joinName name1("Ali");

    joinName name2(name1);//Copy constructor.

    name1.display();

    name2.display();

    name1.concatenate("Aslam");

    name1.display();

    name2.display();

    return 0;

}

**Destructors**

Note: Below code taken from above program.

    //Destructor

    ~joinName(){

        delete []name;

    }

**Inheritance**

Def: When properties & members of base class(parent) are passed on the derived class(child).

#include <iostream>

#include <string>

using namespace std;

class Person{

    public:

        string name;

        int age;

    // Person(string name, int age){

    //     this->name = name;

    //     this->age = age;

    // }

    Person(){

    }

};

        //Inheritence

class Student : public Person{

    //We req student name, age, rollno.

    public:

        int rollno;

    void showInfo(){

    cout << "Name: " << name << endl;

    cout << "Age: " << age << endl;

    cout << "Roll no: " << rollno << endl;

    }

};

int main(void){

    Student s1;

    s1.name = "Arsal";

    s1.age = 16;

    s1.rollno = 45;

    s1.showInfo();

    return 0;

}

Note: Now we’re create an obj of child class.

* Note: First of all parent class constructor is call then child constructor follows.
* Note: First child class memory is deallocate then the parent class follows.

#include <iostream>

#include <string>

using namespace std;

class Person{

    public:

        string name;

        int age;

    // Person(string name, int age){

    //     this->name = name;

    //     this->age = age;

    // }

    Person(){

        cout << "Parent Constructor. " << endl;

    }

    ~Person(){

        cout << "Parent Destructor. " <<  endl;

    }

};

        //Inheritence

class Student : public Person{

    //We req student name, age, rollno.

    public:

        int rollno;

    Student(){

        cout << "Child Constructor. " << endl;

    }

    void showInfo(){

    cout << "Name: " << name << endl;

    cout << "Age: " << age << endl;

    cout << "Roll no: " << rollno << endl;

    }

    ~Student(){

        cout << "Child Destructor. " << endl;

    }

};

int main(void){

    Student s1;

    s1.name = "Arsal";

    s1.age = 16;

    s1.rollno = 45;

    s1.showInfo();

    return 0;

}

* In case of parameterized constructor.

#include <iostream>

#include <string>

using namespace std;

class Person{

    public:

        string name;

        int age;

    //Parameterized constructor.

    Person(string name, int age){

        this->name = name;

        this->age = age;

    }

    ~Person(){

        cout << "Parent Destructor. " <<  endl;

    }

};

        //Inheritence

class Student : public Person{

    //We req student name, age, rollno.

    public:

        int rollno;

    //Person class contructor calls first then Student class invoks.

    //Here, we explicitly calls parent class constructor then child class constructor.

    Student(string name, int age, int rollno) : Person(name, age){ //When constructor is called

        this->rollno = rollno;                                      //only value are passed not data-type.

    }

    void showInfo(){

    cout << "Name: " << name << endl;

    cout << "Age: " << age << endl;

    cout << "Roll no: " << rollno << endl;

    }

    ~Student(){

        cout << "Child Destructor. " << endl;

    }

};

int main(void){

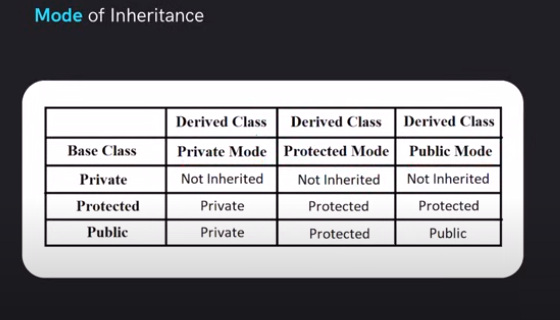
    Student s1("Aslam", 14, 120);

    s1.showInfo();

    return 0;

}

**Modes of inheritance**



Note: The property which is use only for inheritance then it should be in protected mode.

Program to show public, protected, private.

* Public:
* #include <iostream>
* #include <string>
* using namespace std;
* class Parent{
* public:
* int publicVar;
* };
* class Child : public Parent{
* public:
* void display(){
* cout << "Value of public Variable: " << publicVar << endl;
* }
* };
* int main(void){
* Child c1;
* c1.publicVar = 10;
* c1.display();
* return 0;
* }
* Private:
* #include <iostream>
* #include <string>
* using namespace std;
* class Parent{
* public:
* int privateVar;
* };
* class Child : private Parent{
* public:
* void display(){
* cout << "Value of Private Variable: " << privateVar << endl;
* }
* };
* int main(void){
* Child c1;
* c1.privateVar = 10;
* c1.display();
* return 0;
* }

Note: in private mode it cann’t be inherited.

* Protected:

#include <iostream>

#include <string>

using namespace std;

class Parent{

    protected:

        int protectedVar;

    public:

            Parent(int value) : protectedVar(value){};

};

class Child : protected Parent{

    public:

        void display(){

                cout << "Value of Protected Variable: " << protectedVar << endl;

            }

};

int main(void){

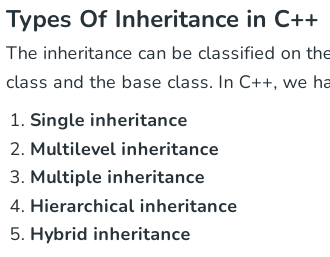
    Child c1(14);

    c1.display();

    return 0;

}

**Types of inheritance**



1-

#include <iostream>

#include <string>

using namespace std;

class Vehicle{

    public:

    Vehicle(){

        cout << "This is a Vehicle! " << endl;

    }

};

class Car: public Vehicle{

public:

    Car(){

        cout << "This Vehicle is a Car." << endl;

    }

};

int main(){

//Creating obj will invokes the constructor of Base class.

    Car obj;

    return 0;

}

2-

#include <iostream>

#include <string>

using namespace std;

class Person{

    public:

        int age;

        string name;

};

class Student : public Person{

    public:

        int rollno;

};

class gradStudent : public Student{

    public:

        string researchTopic;

};

int main(void){

    //These all are objects of gradStudent

    gradStudent g1;

    g1.name = "Ahmed";

    g1.age = 12;

    g1.rollno = 55;

    g1.researchTopic = "Botany";

    //print information.

    cout << "Name: " << g1.name << endl;

    cout << "Age: " << g1.age << endl;

    cout << "Rollno: " << g1.rollno << endl;

    cout << "Research Topic: " << g1.researchTopic << endl;

    return 0;

}

3-

#include <iostream>

#include <string>

using namespace std;

class Student{

    public:

        int rollno;

        string name;

};

class Teacher{

    public:

    string subject;

    double salary;

};

class teacherAss : public Student, public Teacher{

};

int main(void){

    //These all are objects of TeacherAss.

    teacherAss g1;

    g1.name = "Ahmed";

    g1.rollno = 55;

    g1.salary = 45000;

    g1.subject = "CS";

    //print information.

    cout << "Name: " << g1.name << endl;

    cout << "Rollno: " << g1.rollno << endl;

    cout << "Salary: " << g1.salary << endl;

    cout << "Subject: " << g1.subject << endl;

    return 0;

}

4-

#include <iostream>

#include <string>

using namespace std;

class Person{

    public:

        string name;

        int age;

};

class Student : public Person{

    public:

        int rollno;

};

class Teacher : public Person {

    public:

    string subject;

    double salary;

};

int main(void){

    //These all are objects of TeacherAss.

    Teacher g1;

    g1.name = "Ahmed";

    g1.salary = 45000;

    g1.subject = "CS";

    //print information.

    cout << "Name: " << g1.name << endl;

    cout << "Salary: " << g1.salary << endl;

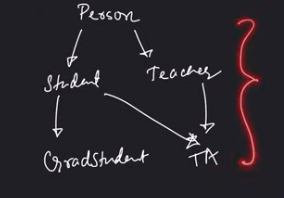
    cout << "Subject: " << g1.subject << endl;

    return 0;

}

5-

Note : all type of inheritance are done here.



**Polymorphism**

Def: is the ability of objects to take on diff forms or behave in diff ways depending on the context in which they’re used.

* Constructor Overloading is an example of polymorphism..

#include <iostream>

#include <string>

using namespace std;

class Student{

    public:

        string name;

    Student(){

        cout << "Non-parameterized! " << endl;

    }

    Student(string name){

        this->name = name;

        cout << "Parameterized! " << endl;

    }

};

int main(void){

    //When s1 is call without parenthesis then non-parameterized is call.

    //When s1 is call with parenthesis then parameterized is call.

    Student s1;

    return 0;

}

**KINDS**

* Compile Time.
* Run Time.
* Compile Time Examples:

1. Constructor Overloading:

Note : Code is above.

1. Function Overloading:

Note: is occurs when two functions occurs in same class with diff parameters.

#include <iostream>

#include <string>

using namespace std;

class Print{

    public:

        void show(int x){

            cout << "Integer: " << x << endl;

        }

        void show(char ch){

            cout << "Character: " << ch << endl;

        }

};

int main(void){

    Print p1;

    //At compile time it detects that the data-type is weather char or int and then give output accordingly.

    p1.show('A');

    return 0;

}

1. Operator overload.

#include <iostream>

#include <string>

using namespace std;

class MinusOverload{

    private:

        int a;

        int b;

    public:

        void Distance(){

            a = 0;

            b = 0;

        }

        //Constructor.

        MinusOverload(int i, int j){

            int c;

            a = i;

            b = j;

            c = a - b;

            cout << "C: " << c << endl;

        }

        void Display(){

            cout << "A: " << a << "B: " << b << "C: " << c << endl;

        }

        MinusOverload operator-(){

        // - is the operator overload.

            a = -a;

            b = -b;

            return MinusOverload(a, b);

        }

};

int main(void){

    MinusOverload m1(6, 7), m2(8, 4);

    -m1;

    m1.Display();//Displays the negate value of m1.

    -m2;

    m2.Display();//Displays the negate value of m2.

    return 0;

}

**Function Overriding**

Def: When parent and child both contain the same function with diff implementation.

Note: Child class function overrides the parent class function.

#include <iostream>

using namespace std;

class Parent{

    public:

        void getInfo(){

            cout << "This is a Parent class. " << endl;

        }

};

class Child : public Parent{

public:

    void getInfo(){

            cout << "This is a Child class. " << endl;

        }

};

int main(void){

    Child c1;

    c1.getInfo();

    return 0;

}

**Virtual Functions**

Def: It is a member function that you expect to be redefined in the derived class.

Note: These function are written in Parent class and are & overridden in a Child class.

#include <iostream>

using namespace std;

class Parent{

    public:

        void getInfo(){

            cout << "This is a Parent class. " << endl;

        }

        virtual void greeting(){

            cout << "Hello from Parent function. " << endl;

        }

};

class Child : public Parent{

public:

    void getInfo(){

            cout << "This is a Child class. " << endl;

        }

    void greeting(){

            cout << "Hello from child function. " << endl;

        }

};

int main(void){

    Child c1;

    c1.greeting();

    return 0;

}

**Abstraction**

Def: Hidden all unnecessary details and show only the important part.

Note: The class having pure virtual function automatically becomes Abstract class.

#include <iostream>

using namespace std;

class Shape{ // Due to Pure virtual func it became a Abstract class.

    virtual void draw() = 0; // Pure virtual function.

};

class Circle : public Shape{

    public:

        void draw(){

            cout << "A circle is drawing! " << endl;

        }

};

int main(void){

    Circle c1;

    c1.draw();

    return 0;

}

**Static Keyword**

* Static Variable:

In case of function: Variable declare as a static in a function are created & initialized once for lifetime of the program.

#include <iostream>

using namespace std;

void zero(){

    static int x = 0; //It is initialized once then it's value increases.

    cout << "Value of x= " << x << endl;

    x++;

}

int main(void){

    zero();

    zero();

    zero();

    return 0;

}

In case of class:

#include <iostream>

using namespace std;

class Sample{

    public:

        int x;

    void increment(){

        x += 1;

    }

};

int main(void){

    Sample obj;

    obj.x = 0; // Declare the value of x = 0.

    cout << obj.x << endl// print 0

    obj.increment();

    cout << obj.x << endl;// print 1

    return 0;

}

* Static Objects.

Note: These static obj will exsits until the life of program.

#include <iostream>

using namespace std;

class Case{

    public:

        Case(){

            cout << "Constructor! " << endl;

        }

        ~Case(){

            cout << "Destructor! " << endl;

        }

};

int main(void){

    if(true){

        static Case obj;

    }

    cout << "End of Main body. \n";

    return 0;

}