**Object Oriented Programming in C++**

**Important point Object-Oriented Programming:-**

* C++ language was designed with the main intention of adding object-oriented programming to C language. The invention of C++ is “**Bjarne Stroustrup”.**
* As the size of the program increases readability, maintainability, and bug-free nature of the program decrease.
* This was the major problem with languages like C which relied upon functions or procedure (hence the name procedural programming language)
* As a result, the possibility of not addressing the problem adequately was high
* Also, data was almost neglected, data security was easily compromised
* Using classes solves this problem by modelling program as a real-world scenario

**Difference between Pop’s and Oop’s:**

**Procedure Oriented Programming:**

* Consists of writing a set of instruction for the computer to follow
* The main focus is on functions and not on the flow of data
* Functions can either use local or global data
* Data moves openly from function to function

**Object-Oriented Programming:**

* Works on the concept of classes and object
* A class is a template to create objects
* Treats data as a critical element
* Decomposes the problem in objects and builds data and functions around the objects

**Basic Concepts in Object-Oriented Programming**

**Classes -** Basic template for creating objects

**Objects –** Basic run-time entities

**Data Abstraction & Encapsulation –** Wrapping data and functions into a single unit

**Inheritance –** Properties of one class can be inherited into others

**Polymorphism –** Ability to take more than one forms

**Dynamic Binding –** Code which will execute is not known until the program runs

**Message Passing –** message (Information) call format

**Benefits of Object-Oriented Programming**

* Better code reusability using objects and inheritance.
* Principle of data hiding helps build secure systems.
* Multiple Objects can co-exist without any interference.
* Software complexity can be easily managed.

#include<bits/stdc++.h>

using namespace std;

class Student{

    string name;    //Private

    public:

    int age;         // Data member of class

    bool gender;

    void setName(string s){

        name = s;

    }

    void printInfo(){

        cout<<"Name = ";

        cout<<name<<endl;

        cout<<"Age = ";

        cout<<age<<endl;

        cout<<"Gender = ";

        cout<<gender<<endl;

    }

};

int main()

{   Student arr[3];

    for(int i=0;i<3;i++){

        string s;

        cout<<"Name = ";

        cin>>s;

        arr[i].setName(s);

        cout<<"Age = ";

        cin>>arr[i].age;

        cout<<"Gender = ";

        cin>>arr[i].gender;

    }

    for(int i=0;i<3;i++){

        arr[i].printInfo();

    }

    return 0;

}

#include<bits/stdc++.h>

using namespace std;

class Student{

    string name;

    public:

    int age;

    bool gender;

    Student(){

        cout<<"Default Constructor"<<endl;

    }

    Student(string s, int a, int g){

        cout<<"Parameter Constructor"<<endl;

        name = s;

        age = a;

        gender = g;

    }

    Student(Student &a){

        cout<<"Copy Constructor"<<endl;

        name = a.name;

        age = a.age;

        gender = a.gender;

    }

    ~Student(){

        cout<<"Distructor called."<<endl;

    }

    void setName(string s){

        name = s;

    }

    void getName(string s){

        cout<< name <<endl;

    }

    void printInfo(){

        cout<<"Name = ";

        cout<<name<<endl;

        cout<<"Age = ";

        cout<<age<<endl;

        cout<<"Gender = ";

        cout<<gender<<endl;

    }

    bool operator == (Student &a){

        if(name==a.name && age==a.age && gender==a.gender){

            return true;

        }

        return false;

    }

};

int main()

{   Student a("Adarsh", 20, 1);

    //a.printInfo();

    Student b("Rahul", 21, 0);

    Student c = a;

    if(b==a){

        cout<<"same"<<endl;

    }

    else{

        cout<<"Not same"<<endl;

    }

    return 0;

}

**21.3 Inheritance:**

It is possible to inherit attribute and methods from one class to another.

**Derived Class** (child) :- the class that inherits from another class.

**Base Class** (parent) :- the class being inherited from.

**Type of Inheritance:**

1.Single Inheritance.

class A {

    public:

        void func(){

            cout << "Inherited";

        }

};

class B : public A {

};

int main(){

    B b;

    b.func();

}

**2.Muliple Inheritance.**

class A {

    public:

        void Afunc(){

            cout << "Func A\n";

        }

};

class B {

    public:

        void Bfunc(){

            cout << "Func B\n";

        }

};

class C : public A, public B {

    public:

};

int main(){

    C c;

    c.Afunc();

    c.Bfunc();

}

**3.Multi-level Inheritance.**

class A {

    public:

        void Afunc(){

            cout << "Func A\n";

        }

};

class B : public A {

    public:

        void Bfunc(){

            cout << "Func B\n";

        }

};

class C : public B {

    public:

};

int main(){

    C c;

    c.Afunc();

    c.Bfunc();

}

**4.Hybride Inheritance.**

**5.Hierarchical Inheritance.**

**CBSE 2019 Computer Science.**

class Ground {

    int Rooms;

    protected:

        void put();

    public:

        void get();

};

class Middle : private Ground {

    int Labs;

    public:

        void Take();

        void Give();

};

class Top : public Middle {

    int Root;

    public:

        void In();

        void Out();

};

**1.** Which type of inheritance is this ?

**Ans**: Multi-level Inheritance.

**2.** Write the names of all the member, which are directly accessible by the member function Give() of class Middle.

**Ans**: **Data Members:** Labs

**Member Function:** Put(), Get(), Take(), Give().

**3.** Write the names of all the members, which are directly accessible by the member function Out() of class Top.

**Ans:** **Data Members:** Roof

**Member Functions:** Take(), Give(), In(), Out().

**4.** Write the names of all the members, which are directly accessible by the object **T** of class Top declared in main() function.

**Ans:** Take(), Give(), In(), Out().

**21.4 Polymorphism in Oop’s :**

#include<bits/stdc++.h>

using namespace std;

//Function Overloading

class Apnacollege{

    public:

        void fun(){

            cout<<"I am a function with no arguments"<<endl;

        }

        void fun(int x){

            cout<<"I am a function with int argument"<<endl;

        }

        void fun(double x){

            cout<<"I am a function with double argument"<<endl;

        }

};

int main(){

    Apnacollege obj;

    obj.fun();

    obj.fun(4);

    obj.fun(6.2);

    return 0;

}

**Output:**

I am a function with no arguments

I am a function with int argument

I am a function with double argument

#include<bits/stdc++.h>

using namespace std;

// Operator Overloading

class Complex{

    private:

        int real, imag;

    public:

        Complex(int r = 0, int i = 0){

            real = r;

            imag = i;

        }

    Complex operator + (Complex const &obj){

        Complex res;

        res.imag = imag + obj.imag;

        res.real = real + obj.real;

        return res;

    }

    void display(){

        cout<<real<<" + i"<<imag<<endl;

    }

};

int main(){

    Complex c1(12, 7);

    Complex c2(6, 7);

    Complex c3 = c1 + c2;

    c3.display();

    return 0;

}

**Output**: 18 + i14

#include<bits/stdc++.h>

using namespace std;

// Function Overwriting

class base{

    public:

        virtual void print(){

            cout<<"This is the base class's print Function"<<endl;

        }

        void display(){

            cout<<"This is the base class's display Function"<<endl;

        }

};

class derived : public base{

    public:

        void print(){

            cout<<"This is the derived class's print Function"<<endl;

        }

        void display(){

            cout<<"This is the derived class's display Function"<<endl;

        }

};

int main(){

    base \*baseptr;

    derived d;

    baseptr = &d;

    baseptr -> print();     // not used dot operator

    baseptr -> display();

    return 0;

}

**Output:**

This is the derived class's print Function

This is the base class's display Function