

Object Oriented Programs

Starter

By Prateek Panwar

Index

|  |  |  |
| --- | --- | --- |
| SN | Program | Page No. |
| 1 | Write a program to understand the basic concepts of Object-Oriented Programming | 5 |
| 2 | Write a program display the following output using single cout statement.  Maths = 90, Physics = 74, Chemistry = 76. | 6 |
| 3 | Write a program to read two numbers from the keyboard and display the larger value on the screen. | 7 |
| 4 | Write a program to print ASCII value corresponding to given character. | 9 |
| 5 | Write a program to understand different types of constructors in C++. | 10 |
| 6 | Write a program to understand different types of objects in C++. | 12 |
| 7 | Write a program to understand the concept of metaclass in c++ | 14 |
| 8 | Write a program to demonstrate the concept of association. | 16 |
| 9 | Write a program to demonstrate the concept of aggregation | 18 |
| 10 | Write a program to demonstrate the concept of composition. | 20 |
| 11 | Write a program to demonstrate the concept of delegation. | 22 |
| 12 | Write a program to demonstrate the concept of Namespace. | 23 |
| 13 | Write a program to implement the concept of Inheritance. a. Mutilevel Inheritance b. Multiple Inheritance | 24 |
| 14 | Write a C++ program to understand the following concepts:  a. Method overloading  b. Method overriding and Method hiding | 27 |
| 15 | Design a C++ Class ‘Complex’ with data members for real and imaginary part. Provide default and parameterized constructors. Write a program to perform arithmetic operations of two complex numbers using operator overloading | 30 |
| 16 | Design a base class shape. Use this class to store two double type values that could be used to computer area of figures.  Derive to specific classes called triangle and rectangle from the base shape add to the base a member function getdata() to initialize base class data member another member function display\_area() to compute and display the area of figures. Make display\_area() as a virtual function and redefine it the derived class to suit their requirements. | 33 |
| 17 | Write a program to understand the concept of Abstract Class in C++. | 35 |
| 18 | Write a C++ program to save the data into a file and to display the content of file on the screen. | 36 |
| 19 | Write a program to demonstrate how exceptions are handled in C++. Also write a program that handle user defined exception. | 38 |
| 20 | Write following programs to understand the concept of template:  a. Function template  b. Class template | 41 |

1. Write a program to understand the basic concepts of Object-Oriented Programming.

Ans. Program:

#include <iostream>

#include <conio.h>

using namespace std;

class employees

{

    int id;

    char name[20];

public:

    void getdetails()

    {

        cout << "Enter id: ";

        cin >> id;

        cout << "Enter Name: ";

        cin >> name;

    }

    void showdetails()

    {

        cout << "\nID: " << id;

        cout << "\nName: " << name;

    }

};

int main()

{

    employees e1;

    e1.getdetails();

    e1.showdetails();

}

Output:

Text

Description automatically generated

1. Write a program display the following output using single cout statement. Maths = 90, Physics = 74, Chemistry = 76.

Ans. Program:

#include <iostream>

#include <conio.h>

using namespace std;

int main()

{

    cout << "Maths = " << 90 << "\nPhysics = " << 74 << "\nChemistry = " << 76;

}

Output:

Text

Description automatically generated

1. Write a program to read two numbers from the keyboard and display the larger value on the screen

Ans. Program:

#include <iostream>

#include <conio.h>

using namespace std;

class LargerNum

{

public:

int a, b;

LargerNum()

{

a = 0;

b = 0;

}

void getData()

{

cout << "Enter a number: ";

cin >> a;

cout << "Enter a number: ";

cin >> b;

}

void showData()

{

if (a > b)

{

cout << a << " is larger than " << b;

}

if (b > a)

{

cout << b << " is larger than " << a;

}

if (a == b)

{

cout << a << " and " << b << " are equal";

}

}

};

void main()

{

LargerNum LN;

LN.getData();

LN.showData();

}

Output:

Graphical user interface, text

Description automatically generated

1. Write a program to print ASCII value corresponding to given character

Ans. Program:

#include <iostream>

#include <conio.h>

using namespace std;

class FindASCII

{

public:

char a;

int value;

FindASCII()

{

a = '0';

}

void getData()

{

cout << "Enter a character: ";

cin >> a;

}

void showData()

{

value = a;

cout << "ASCII value: " << value;

}

};

void main()

{

FindASCII LN;

LN.getData();

LN.showData();

}

Output:

Graphical user interface, text, application

Description automatically generated

1. Write a program to understand different types of constructors in C++

Ans. Program:

//Area of Cone

#include <iostream>

#include <conio.h>

using namespace std;

class FindArea

{

public:

float pi = 3.14;

float temp, answer; //Answer will be first store in temp variable then copied to answer

//Default Constructor - Parameters are not passed

FindArea()

{

float local\_radius, local\_height;

cout << "\n\nDefault Constructor";

cout << "\nEnter height: ";

cin >> local\_height;

cout << "Enter radius: ";

cin >> local\_radius;

answer = pi \* local\_radius \* local\_radius \* local\_height / 3;

}

//Parameterized Constructors - Parameters are passed

FindArea(float h, float r)

{

temp = pi \* r \* r \* h / 3;

answer = temp; //Copy constructor - a Variable copied from another

}

void display()

{

cout << "Answer = " << answer;

}

};

int main()

{

float r, h;

cout << "Area of Cone";

//Calling default constructor

FindArea default\_construct;

default\_construct.display();

cout << "\n\nParameterized Constructor";

cout << "\nEnter height: ";

cin >> h;

cout << "Enter radius: ";

cin >> r;

//Calling parameterized constructor

FindArea parameterized\_construct(h, r);

parameterized\_construct.display();

return 0;

}

Output:

A screen shot of a computer

Description automatically generated with low confidence

1. Write a program to understand different types of objects in c++.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1   //Used for using strncpy in vs

#include<iostream>

#include<string.h>

using namespace std;

class Object                //Object Class

{

private:

    char obj\_name[30];

public:

    Object(char\* name\_inpt); //Parameterized Constructor declaration

    ~Object();              //destructor declaration

};

Object::Object(char\* name\_inpt)

{

    strncpy(obj\_name, name\_inpt, 29);

    obj\_name[29] = '\0';

    cout << "\nContructor called for " << obj\_name << "\n";

}

Object::~Object()

{

    cout << "Destructor called for " << obj\_name << "\n";

}

Object ext\_obj("External Object");

int main()

{

    cout << "\nMain function's beginning\n";

    Object auto\_obj("Automatic Object");

    static Object st\_obj("Static Object");

    Object \* dyn\_obj = new Object("Dynamic Object");

    delete dyn\_obj;           //Dynamic Object needs manual deletion

    cout << "\nMain function's end\n";

    return 0;

}

Output:

Text

Description automatically generated

1. Write a program to understand the concept of metaclass in c++.

Ans. Program:

#include <iostream>

#include <string.h>

using namespace std;

class base                  //MetaClass

{

public:

    char day[10];

    base();

    virtual void show() {};  //Empty Virtual Function

};

base::base()    //Constructor of base class

{

    strcpy(day, "Monday");

}

class derived : public base //Derived Class

{

    void show()

    {

        cout << "Hello, From derived class\nToday is " << day;

    }

};

int main()

{

    base\* ptr;

    derived d;

    ptr = &d;

    ptr->show();

    return 0;

}

Output:

Text

Description automatically generated

1. Write a program to demonstrate the concept of association.

Ans.

Program:

//strcpy\_s is secure version of strcpy used for visual studio

#include <iostream>

#include <string.h>

using namespace std;

class Manufacturer

{

public:

char name[20];

Manufacturer (char\* obj\_name) //Get value of name[]

{

strcpy\_s(name, obj\_name);

}

char\* getdetails()

{

return name;

}

};

class Model

{

char model[20];

public:

Model(char\* obj\_model) //Get value of name[]

{

strcpy\_s(model, obj\_model);

}

char\* getdetails() //+1 Overloaded

{

return model;

}

};

int main()

{

int n\_entry;

char o\_manufacturer[20][5], o\_model[20];

cout << "Enter number of entries (max 5): ";

cin >> n\_entry;

for (int i = 0; i < n\_entry; i++)

{

cout << "\n\nEntry " << i + 1;

cout << "\nEnter Manufacturer's name: ";

cin >> o\_manufacturer[i];

cout << "Enter Model's name: ";

cin >> o\_model;

Manufacturer mn(o\_manufacturer[i]);

Model md(o\_model);

cout<<"\nDetails:\nManufacturer> "<<mn.getdetails();

cout << ", Model> " << md.getdetails();

}

}

Output:

Text

Description automatically generated

1. Write a program to demonstrate the concept of aggregation.

Ans. Program:

#define \_CRT\_SECURE\_NO\_WARNINGS 1   //Used for using strncpy in vs

#include<iostream>

#include<string.h>

using namespace std;

class specs //Stores details of car

{

public:

    int price, power, seats;

    char engine[30];

    specs(int p, int pw, int s, char\* eng) //"specs" Constructor

    {

        strcpy(engine, eng);

        price = p; power = pw; seats = s;

    }

};

class car   //Stores Model information

{

    char model[30];

    specs\* s;

public:

    car(char\* md)   //"car" Constructor

    {

        strcpy(model, md);

    }

    void setAddress(specs\* ptr)    //To set pointer of "specs"

    {

        s = ptr;

    }

    void display()

    {

        cout << "\n\nModel: " << model << ", Price: " << s->price;

        cout << "\nEngine: " << s->engine << ", Power: " << s->power << ", Seats: " << s->seats;

    }

};

int main()

{

    specs s1(550000, 65, 5, "1 l IRDE2 Petrol");

    specs s2(1050000, 115, 5, "1.5 l CRDi Diesel");

    car c1("i10");

    car c2("Verna");

    c1.setAddress(&s1);

    c2.setAddress(&s2);

    c1.display();   c2.display();

    return 0;

}

Output:

Text

Description automatically generated

1. Write a program to demonstrate the concept of composition.

Ans. Program:

#include <iostream>

#include <string>

using namespace std;

class Course //In composition, each class depends on another such that if one of them is destroyed, Other class will not have a purpose

{

public:

string name;

Course()

{

name = "BTech in Computer Science";

}

};

class Subjects //Subject class is dependent on base class

{

public:

string s\_name[5];

Course c;

void setData()

{

s\_name[0] = "Object oriented Programming";

s\_name[1] = "Data Struct & Algo";

s\_name[2] = "Physics";

s\_name[3] = "Maths";

s\_name[4] = "Communication Skills";

}

void Display()

{

cout << "\nCourse Name: " << c.name << "\nSubjects:";

for (int i = 0; i < 5; i++)

{

cout << "\n" << i+1 << ". " << s\_name[i];

}

}

};

int main()

{

Subjects s;

s.setData();

s.Display();

}

Output:

Text

Description automatically generated

1. Write a program to demonstrate the concept of delegation.

Ans. Program:

#include <iostream>

using namespace std;

class base //The class which forwards method is called Dalegate

{

public:

void Display()

{

cout << "Delegate";

}

};

class Print

{

public:

base b;

void Display()

{

b.Display();

}

};

void main()

{

Print p;

p.Display();

}

Output:

Text

Description automatically generated

1. Write a program to demonstrate the concept of Namespace.

Ans. Program:

#include<iostream>

using namespace std;

namespace MySpace   //We'll add 2 numbers using this namespace

{

    int a, b;

    void getData(); //Function to get 2 numbers

    class calc

    {

    public:

        void add(); //Func to add 2 numbers

    };

}

void MySpace::getData()

{

    cin >> a >> b;

}

void MySpace::calc::add()

{

    cout << a + b;

}

using namespace MySpace;

int main()

{

    cout << "Enter 2 numbers: ";

    getData();

    cout << "Answer = ";

    calc obj\_c;

    obj\_c.add();

    return 0;

}

Output:

Graphical user interface, text

Description automatically generated

1. Write a program to implement the concept of Inheritance.

a. Mutilevel Inheritance

b. Multiple Inheritance

Ans. Program:

1. Multilevel Inheritance

#include <iostream>

#include <string>

using namespace std;

class a //a is base class to b (a->b)

{

protected: //Data members accessible by inherited classes only

int salary = 35000;

string job\_dept = "Design";

public:

string name = "Raju", type = "Employee";

};

class b : public a //b is inherited class to a (b->a)

{

public:

void Display()

{

cout << "\nView as Manager:";

cout << "\nName: " << name << ", Working as: " << job\_dept;

}

};

class c : public b //b is also base class to c (b->c)

{

public:

void Display1()

{

cout << "\n\nView as CEO:";

cout << "\nName: " << name << ", Working as: " << job\_dept<<

"\nDepartment: "<<job\_dept<<", Salary: "<< salary;

}

};

void main()

{

b obj\_b;

c obj\_c;

obj\_b.Display();

obj\_c.Display1();

}

1. Output:

Text

Description automatically generated

1. Multiple Inheritance

#include <iostream>

#include <string>

using namespace std;

class a

{

public:

string fname = "Prateek";

};

class b

{

public:

string rname = "Panwar";

};

class c : public a, public b

{ //class a and b both are inherited by class c  (a->c & b->c)

public:

void Display()

{

cout << "Name: " << fname << " " << rname;

}

};

void main()

{

c obj\_c;

obj\_c.Display();

}

b. Output:

Graphical user interface, text

Description automatically generated

1. Write a C++ program to understand the following concepts:
2. Method overloading
3. Method overriding and Method hiding

Ans. Program:

1. Method overloading:

#include <iostream>

using namespace std;

class base

{

public:

    void display(int a)     //Displays integer

    {

        cout <<"Number: "<<a;

    }

    void display(double f)  //Overload +1: Displays float

    {

        cout << "\nFloat: "<<f;

    }

    void display(char const \*s)   //Overload +2: Displays char

    {

        cout <<"\nCharacter: "<<s;

    }

};

int main()

{

    base b;

    b.display(65);

    b.display(65.5);

    b.display("Sixty five");

    return 0;

}

1. Output:

Text

Description automatically generated

1. Method overriding and Method hiding

#include <iostream>

using namespace std;

class base      //Base class

{

public:

    virtual void display()

    {

        cout <<"\nBase Class";

    }

    void func()

    {

        cout << "\nfunc() method of base Class";

    }

};

class d1 : public base

{

public:

    void display()     //Method OverRide +1

    {

        cout << "\nOverrided Method display";

    }

    void func()         //Method Hiding +1

    {

        cout << "\nfunc() method of derieved class";

    }

};

int main()

{

    base b;

    d1 obj\_d1;

    b.display();

    b.func();

    obj\_d1.display();

    obj\_d1.func();

    return 0;

}

b. Output:

Graphical user interface, text

Description automatically generated

1. Design a C++ Class ‘Complex’ with data members for real and imaginary part. Provide default and parameterized constructors. Write a program to perform arithmetic operations of two complex numbers using operator overloading;

Ans. Program:

#include <iostream>

using namespace std;

class Complex

{

    double a, b; //a is real part, b is imaginary part

public:

    Complex()

    {

        a = 0.0, b = 0.0;

    }

    Complex(double a1, double b1)

    {

        a = a1;  b = b1;

    }

    Complex operator+(Complex);

    Complex operator-(Complex);

    Complex operator\*(Complex);

    Complex operator/(Complex);

    void display()

    {

        cout <<a << " + i(" << b<<")";

    }

};

//Each operator is overloaded and 'temp' is object of class 'Complex' to get 2nd data

Complex Complex :: operator+(Complex temp)

{

    Complex obj;

    obj.a = a + temp.a;

    obj.b = b + temp.b;

    return(obj);

}

Complex Complex :: operator-(Complex temp)

{

    Complex obj;

    obj.a = a - temp.a;

    obj.b = b - temp.b;

    return(obj);

}

Complex Complex :: operator\*(Complex temp)

{

    Complex obj;

    obj.a = a \* temp.a;

    obj.b = b \* temp.b;

    return(obj);

}

Complex Complex :: operator/(Complex temp)

{

    Complex obj;

    obj.a = a / temp.a;

    obj.b = b / temp.b;

    return(obj);

}

int main()

{

    double r1, r2, i1, i2;

    Complex c1, c2, ans;

    char opr;

    cout << "Enter data (r = real num),(i = imaginary num): ";

    cout << "\nr1: ";  cin >> r1;

    cout << "i1: ";  cin >> i1;

    cout << "r2: ";  cin >> r2;

    cout << "i2: ";  cin >> i2;

    c1 = Complex(r1, i1);

    c2 = Complex(r2, i2);

    //Getting answers for each operation

    cout << "\nAnswers: ";

    ans = c1 + c2;

    cout << "\nAdd = ";  ans.display();

    ans = c1 - c2;

    cout << "\nSubtract = ";  ans.display();

    ans = c1 \* c2;

    cout << "\nMultiply = ";  ans.display();

    ans = c1 / c2;

    cout << "\nDivide = ";  ans.display();

    return 0;

}

Output:

A screen shot of a computer

Description automatically generated with low confidence

1. Design a base class shape. Use this class to store two double type values that could be used to computer area of figures. Derive to specific classes called triangle and rectangle from the base shape add to the base a member function getdata() to initialize base class data member another member function display\_area() to compute and display the area of figures. Make display\_area() as a virtual function and redefine it the derived class to suit their requirements

Ans. Program:

#include <iostream>

using namespace std;

class shape

{

public:

    double a, b;

    void getdata(double x, double y)

    {

        a = x;  b = y;

    }

    virtual void display\_area() {}

};

class triangle : public shape

{

    void display\_area()

    {

        cout << "\nArea of Triangle is " << 0.5 \* a \* b;

    }

};

class rectangle : public shape

{

    void display\_area()

    {

        cout << "\nArea of Rectangle is " << a \* b;

    }

};

int main()

{

    double x, y;

    shape\* ptr;

    triangle t; rectangle r;

    cout << "Enter length of base: ";   cin >> x;

    cout << "Enter length of height: ";   cin >> y;

    ptr = &t;

    ptr->getdata(x, y);

    ptr->display\_area();

    ptr = &r;

    ptr->getdata(x, y);

    ptr->display\_area();

    return 0;

}

Output:

Graphical user interface, text

Description automatically generated

1. Write a program to understand the concept of Abstract Class in C++.

Ans. Program:

#include <iostream>

using namespace std;

class base //Abstract class is a class with a method declared but not defined

{

virtual void display() = 0; //virtual function undefined

};

class derived : public base

{

public:

void display () //Here we define function

{

cout << "Virtual function is called from derived class";

}

};

void main()

{

derived d;

d.display();

}

Output:

A picture containing text, black, screenshot

Description automatically generated

1. Write a C++ program to save the data into a file and to display the content of file on the screen

Ans. Program:

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class FileHandler

{

public:

string txt, fname,display\_txt;

FileHandler()

{

txt = "",fname="New Text File";

}

void getData()

{

cout << "File Handler";

cout << "\nEnter text: ";

getline(cin, txt);

cout << "Enter file name: ";

cin >> fname;

}

void WriteFile()

{

ofstream f(fname + ".txt");

f << txt;

f.close();

}

void ReadFile()

{

ifstream f(fname + ".txt");

cout << "\nOpened " << fname + ".txt" << ":\n";

while (getline(f, display\_txt))

{

cout << display\_txt;

}

}

};

void main()

{

FileHandler fh;

fh.getData();

fh.WriteFile();

fh.ReadFile();

}

Output:

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Write a program to demonstrate how exceptions are handled in C++. Also write a program that handle user defined exception.

Ans. Program:

a. Exception Handling

#include <iostream>

#include <string>

using namespace std;

class Division

{

public:

int num1, num2;

Division(int x, int y)

{

num1 = x; num2 = y;

}

void calc()

{

try //Exception handling

{

if (num2 == 0)

throw num2; //Throws exception

else

{

cout << "Answer = " << num1 / num2;

}

}

catch (int)

{

cout << "Caught Exception: Cannot divide by 0";

throw;

}

}

};

void main()

{

int num1, num2;

cout << "Enter 2 numbers: ";

cin >> num1 >> num2;

Division d(num1,num2);

d.calc();

}

Output:

Text

Description automatically generated

b. User Defined Exception

#include <iostream>

#include <string>

using namespace std;

struct ZeroDivisionExcp :public exception //User defined exception

{

const char\* what() const throw()

{

return "Caught Exception: Cannot divide by 0";

}

};

class Division

{

public:

int num1, num2;

Division(int x, int y)

{

num1 = x; num2 = y;

}

void calc()

{

try //Exception handling

{

if (num2 == 0)

throw ZeroDivisionExcp(); //Throws exception

else

{

cout << "Answer = " << num1 / num2;

}

}

catch (ZeroDivisionExcp &z)

{

cout << z.what();

throw;

}

}

};

void main()

{

int num1, num2;

cout << "Enter 2 numbers: ";

cin >> num1 >> num2;

Division d(num1,num2);

d.calc();

}

Output:

Text

Description automatically generated

1. Write following programs to understand the concept of template:
2. Function template b. Class template

Ans. Program:

1. Function template

#include <iostream>

#include <string>

using namespace std;

template <typename t> t sqr (t num) //Template to square number

{

return num \* num;

}

void main()

{

cout << "Integer: "<< sqr(5) << "\n";

cout << "Float: " << sqr(2.5f) << "\n";

cout << "Double: " << sqr(2.5) << "\n";

}

1. Output:

Graphical user interface, text

Description automatically generated

1. Class Template

#include <iostream>

#include <string>

using namespace std;

template <class t> //Template using class to calculate factorial of given number

class fac {

private:

t num, ans;

public:

int calc(int x)

{

ans = 1;

num = x;

for (int i = 1; i <= num; i++)

{

ans = ans \* i;

}

return ans;

}

};

void main()

{

fac<int> f; //Using Integer as input

cout<<"Integer: "<<f.calc(5) << "\n";

fac<float> f1; //Using Float as input

cout <<"Float: "<< f1.calc(4.0f);

}

b. Output:

Graphical user interface, text

Description automatically generated