**OOPM LAB**

1. **Write a program in C++ to create a class Student with data ‘name, city and age’ along with method printData to display the data. Create the two objects s1, s2 to declare and access the values.**

#include <iostream>

using namespace std;

class student

{

private:

char name[30];

char city[30];

int age;

float ;

public:

//member function to get student's details

void getData(void);

//member function to print student's details

void printData(void);

};

//member function definition, outside of the class

void student::getData (void){

cout << "Enter Name: " ;

cin >> name;

cout << "Enter City: ";

cin >> city;

cout << "Enter Age: ";

cin >>age;

}

//member function definition, outside of the class

void student::printData(void){

cout << "Student details:\n";

cout << "Name:"<< name << ",City:" << city << ", Age:" << age <<endl

}

int main()

{

student std; //object creation

std.getData();

std.printData();

return 0;

}

**OUTPUT:**

Enter name: mike

Enter City: Surat

Enter Age: 22

Student details:

Name:mike, City: Surat, Age: 22

1. **Write a program in C++ using parameterized constructor with two parameters id and name. While creating the objects obj1 and obj2 passed two arguments so that this constructor gets invoked after creation of obj1 and obj2**

#include <iostream>

using namespace std;

class abc

{  
int id;

char b[10] ;

abc(int ID, char B)

{

Id=ID;

b=B;

}

Void display()  
{  
cout<<id<<endl;

Cout<<b<<endl;

}  
};

int main()  
{

abc obj1(20,’shrey’), obj2(30,’yash’);

obj1.display();

obj2.display();

}

**OUTPUT:**

20

shrey

30

yash

1. **Write a program in C++ to implement Copy Constructor**

#include<iostream>

using namespace std;

class Point

{

private:

int x, y;

public:

Point(int x1, int y1)

{

x = x1;

y = y1;

}

// Copy constructor

Point(const Point &p1)

{

x = p1.x;

y = p1.y;

}

int ge­­­­­tX()

{

return x;

}

int getY()

{

return y;

}

};

int main()

{

Point p1(10, 15); // Normal constructor is called here

Point p2 = p1; // Copy constructor is called here

// Let us access values assigned by constructors

cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

return 0;

}

**OUTPUT:**

p1.x = 10, p1.y = 15

p2.x = 10, p2.y = 15

1. **Write a program in C++ to implement destructor.**

#include <iostream>

using namespace std;

class HelloWorld

{

public:

//Constructor

HelloWorld()

{

cout<<"Constructor is called"<<endl;

}

//Destructor

~HelloWorld()

{

cout<<"Destructor is called"<<endl;

}

//Member function

void display()

{

cout<<"Hello World!"<<endl;

}

};

int main()

{

//Object created

HelloWorld obj;

//Member function called

obj.display();

return 0;

}

**OUTPUT:**

Constructor is called

Hello World!

Destructor is called

1. **Write a program in C++ to implement Static class member**

#include <iostream>

using namespace std;

void Test(){

static int x = 1;

x = ++x;

int y = 1;

y = ++y;

cout<<"x = "<<x<<"n";

cout<<"y = "<<y<<"n";

}

int main()

{

Test();

Test();

return 0;

}

**OUTPUT:**

x=2

y=2

x=3

y=2

1. **Write a program in C++ to implement multiple Inheritance.**

#include<iostream>

using namespace std;

class Person

{

// Data members of person

public:

Person(int x)

{

cout << "Person::Person(int ) called" << endl; }

};

class Faculty : public Person

{

// data members of Faculty

public:

Faculty(int x):Person(x)

{

cout<<"Faculty::Faculty(int ) called"<< endl;

}

};

class Student : public Person

{

// data members of Student

public:

Student(int x):Person(x)

{

cout<<"Student::Student(int ) called"<< endl;

}

};

class TA : public Faculty, public Student

{

public:

TA(int x):Student(x), Faculty(x)

{

cout<<"TA::TA(int ) called"<< endl;

}

};

int main() {

TA ta1(30);

}

**OUTPUT:**

Person::Person(int ) called

Faculty::Faculty(int ) called

Person::Person(int ) called

Student::Student(int ) called

TA::TA(int ) called

1. **Write a program in C++ to implement Friend function**

#include <iostream>

using namespace std;

class Box

{

private:

int length;

public:

Box(): length(0)

{

}

friend int printLength(Box); //friend function

};

int printLength(Box b)

{

b.length += 10;

return b.length;

}

int main()

{

Box b;

cout<<"Length of box: "<< printLength(b)<<endl;

return 0;

}

**OUTPUT:**

Length of box: 10

1. **Write a program in C++ to implement Operator Overloading.**

// Overload ++ when used as prefix

#include <iostream>

using namespace std;

class Count

{

private:

int value;

public:

// Constructor to initialize count to 5

Count() : value(5)

{

}

// Overload ++ when used as prefix

void operator ++ ()

{

++value;

}

void display()

{

cout << "Count: " << value << endl;

}

};

int main()

{

Count count1;

// Call the "void operator ++ ()" function

++count1;

count1.display();

return 0;

}

**OUTPUT:**

Count: 6

1. **Write a program in C++ to implement virtual base class.**

#include <iostream>

using namespace std;

class A

{

   public:

   int a;

   A()

{

      a = 10;

   }

};

class B : public virtual A

{

};

class C : public virtual A

{

};

class D : public B, public C

{

};

int main(){

   //creating class D object

   D object;

   cout << "a = " << object.a << endl;

   return 0;

}

**OUTPUT:**

a = 10

1. **Write a program in C++ to implement Exception handling.**

#include <iostream>

using namespace std;

double division(int a, int b)

{

if( b == 0 )

{

throw "Division by zero condition!";

}

return (a/b);

}

int main ()

{

int x = 50;

int y = 0;

double z = 0;

try

{

z = division(x, y);

cout << z << endl;

}

catch (const char\* msg)

{

cerr << msg << endl;

}

return 0;

}

**OUTPUT:**

Division by zero condition!

**Index**

|  |  |
| --- | --- |
| **S no.** | **Description** |
| 1 | Properties of class and object |
| 2 | Properties of parametrize constructor |
| 3 | Program to implement Copy Constructor |
| 4 | Program to implement destructor |
| 5 | Program to implement Static class member. |
| 6 | Program to implement multiple Inheritance |
| 7 | Program to implement Friend function |
| 8 | Program to implement operator overloading |
| 9 | Program to implement virtual base class |
| 10 | Program to implement Exception handling |