**LAB 12**

**PRACTICE TASK 3:  
CODE:**

#include<iostream>

using namespace std;

template<typename t>

class queue

{

private:

t list[10];

public:

queue()

{

for (int i = 0;i < 10;i++)

{

list[i] = t{};

}

}

void sort()

{

for (int i = 0;i < 9;i++)

{

for (int j = 0;j < 9 - 1;j++)

{

if (list[j] > list[j + 1])

{

t temp = list[j];

list[j] = list[j + 1];

list[j + 1] = temp;

}

}

}

}

t max()

{

t max\_val = list[0];

for (int i = 0;i < 9;i++)

{

if (list[i] > max\_val)

{

max\_val = list[i];

}

}

return max\_val;

}

t min()

{

t min\_val = list[0];

for (int i = 0;i < 9;i++)

{

if (list[i] < min\_val)

{

min\_val = list[i];

}

}

return min\_val;

}

void return\_queue()

{

cout << " queue elements: " << endl;

for (int i = 0;i < 9;i++)

{

cout << list[i] << " ";

}

}

};

void main()

{

queue <int> iq;

queue <float> fq;

queue <char> cq;

iq.sort();

cout << " max value in intqueue is: " << iq.max()<<endl;

cout << " min value in intqueue is: " << iq.min()<<endl;

iq.return\_queue();

fq.sort();

cout << " max value in floatqueue is: " << fq.max() << endl;

cout << " min value in floatqueue is: " << fq.min() << endl;

fq.return\_queue();

cq.sort();

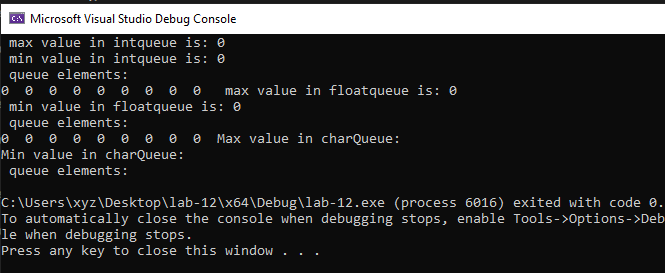
cout << "Max value in charQueue: " << cq.max() << endl;

cout << "Min value in charQueue: " << cq.min() << endl;

cq.return\_queue();

}

**OUTPUT:**

****

**Oop final lab practice:**

/\*#include <iostream>

#include <string>

using namespace std;

// Base class Car

class Car {

protected:

string carName;

bool ignition;

int currentSpeed;

public:

// No-argument constructor

Car() : carName(""), ignition(false), currentSpeed(0) {}

// Parameterized constructor

Car(string name, bool ign, int speed) : carName(name), ignition(ign), currentSpeed(speed) {}

// Setter functions

void setCarName(string name) {

carName = name;

}

void setIgnition(bool ign) {

ignition = ign;

}

void setCurrentSpeed(int speed) {

currentSpeed = speed;

}

// Getter functions

string getCarName() const {

return carName;

}

bool getIgnition() const {

return ignition;

}

int getCurrentSpeed() const {

return currentSpeed;

}

// Member function to set speed

void setSpeed(int speed) {

currentSpeed = speed;

}

};

// Derived class Convertible

class Convertible : public Car {

private:

bool top;

public:

// No-argument constructor

Convertible() : top(false) {}

// Four-argument constructor

Convertible(string name, bool ign, int speed, bool t) : Car(name, ign, speed), top(t) {}

// Setter for top data member

void setTop(bool t) {

top = t;

}

// Display function

void show() const {

cout << "Car Name: " << carName << endl;

cout << "Ignition: " << (ignition ? "On" : "Off") << endl;

cout << "Current Speed: " << currentSpeed << " mph" << endl;

cout << "Top: " << (top ? "Up" : "Down") << endl;

}

};

int main() {

// Instantiate a Convertible object

Convertible convertibleCar("ConvertibleCar", true, 60, true);

// Display initial values

cout << "Initial values:" << endl;

convertibleCar.show();

// Set new values using setter functions

convertibleCar.setCarName("NewConvertibleCar");

convertibleCar.setIgnition(false);

convertibleCar.setCurrentSpeed(45);

convertibleCar.setTop(false);

// Display updated values

cout << "\nUpdated values:" << endl;

convertibleCar.show();

return 0;

}

#include <iostream>

#include <string>

using namespace std;

class Company {

protected:

int companyID;

string companyName;

public:

Company(int id, const string& name) : companyID(id), companyName(name) {}

int getCompanyID() const {

return companyID;

}

void setCompanyID(int id) {

companyID = id;

}

const string& getCompanyName() const {

return companyName;

}

void setCompanyName(const string& name) {

companyName = name;

}

};

class MobilePhone : public Company {

private:

string mobilePhoneName;

int mobileID;

int mobilePrice;

public:

MobilePhone(int id, const string& name, const string& phoneName, int mobID, int price)

: Company(id, name), mobilePhoneName(phoneName), mobileID(mobID), mobilePrice(price) {}

const string& getMobilePhoneName() const {

return mobilePhoneName;

}

void setMobilePhoneName(const string& phoneName) {

mobilePhoneName = phoneName;

}

int getMobileID() const {

return mobileID;

}

void setMobileID(int id) {

mobileID = id;

}

int getMobilePrice() const {

return mobilePrice;

}

void setMobilePrice(int price) {

mobilePrice = price;

}

void display() const {

cout << "Company ID: " << getCompanyID() << endl;

cout << "Company Name: " << getCompanyName() << endl;

cout << "Mobile Phone Name: " << mobilePhoneName << endl;

cout << "Mobile ID: " << mobileID << endl;

cout << "Mobile Price: " << mobilePrice << endl;

}

};

class Laptop : public Company

{

private:

string laptopName;

public:

Laptop(int id, const string& name, const string& lName)

: Company(id, name), laptopName(lName) {}

const string& getLaptopName() const {

return laptopName;

}

void setLaptopName(const string& lName) {

laptopName = lName;

}

void display() const {

cout << "Company ID: " << getCompanyID() << endl;

cout << "Company Name: " << getCompanyName() << endl;

cout << "Laptop Name: " << laptopName << endl;

}

};

int main() {

MobilePhone mobile(1, "Samsung", "Galaxy S21", 101, 999);

Laptop laptop(2, "Dell", "Inspiron");

// Accessing base class members using derived class objects

cout << "Mobile Phone Information:" << endl;

mobile.display();

cout << "\nLaptop Information:" << endl;

laptop.display();

return 0;

}

#include <iostream>

#include <string>

using namespace std;

class CafeService {

protected:

string orderID;

double price;

public:

CafeService() : orderID("ord#0"), price(0.0) {}

CafeService(string orderId, double foodPrice) : orderID(orderId), price(foodPrice) {}

};

class StaffService : public CafeService {

private:

double serviceFee;

int cabinNumber;

public:

StaffService(string orderId, double foodPrice, double fee, int cabin)

: CafeService(orderId, foodPrice), serviceFee(fee), cabinNumber(cabin) {}

double totalCharges() {

return price + serviceFee;

}

void display() {

cout << "Order ID: " << orderID << endl;

cout << "Food Price: $" << price << endl;

cout << "Service Fee: $" << serviceFee << endl;

cout << "Total Charges: $" << totalCharges() << endl;

cout << "Cabin Number: " << cabinNumber << endl;

}

};

int main() {

// Instantiate an object of class StaffService and test the implementation

StaffService staffOrder("ord#123", 25.0, 5.0, 101);

staffOrder.display();

return 0;

}

#include <iostream>

class Calculator {

public:

// Addition

template <typename T>

T add(T a, T b) {

return a + b;

}

// Subtraction

template <typename T>

T subtract(T a, T b) {

return a - b;

}

// Multiplication

template <typename T>

T multiply(T a, T b) {

return a \* b;

}

// Division

template <typename T>

T divide(T a, T b) {

if (b != 0) {

return a / b;

}

else {

std::cerr << "Error: Division by zero." << std::endl;

return static\_cast<T>(0); // Return 0 in case of division by zero

}

}

};

int main() {

Calculator calculator;

// Test with int data type

int intResultAdd = calculator.add(5, 3);

int intResultSubtract = calculator.subtract(8, 2);

int intResultMultiply = calculator.multiply(4, 6);

int intResultDivide = calculator.divide(10, 2);

// Test with double data type

double doubleResultAdd = calculator.add(5.5, 3.3);

double doubleResultSubtract = calculator.subtract(8.6, 2.1);

double doubleResultMultiply = calculator.multiply(4.2, 6.7);

double doubleResultDivide = calculator.divide(10.0, 2.0);

// Display results

std::cout << "Results with int data type:" << std::endl;

std::cout << "Addition: " << intResultAdd << std::endl;

std::cout << "Subtraction: " << intResultSubtract << std::endl;

std::cout << "Multiplication: " << intResultMultiply << std::endl;

std::cout << "Division: " << intResultDivide << std::endl;

std::cout << "\nResults with double data type:" << std::endl;

std::cout << "Addition: " << doubleResultAdd << std::endl;

std::cout << "Subtraction: " << doubleResultSubtract << std::endl;

std::cout << "Multiplication: " << doubleResultMultiply << std::endl;

std::cout << "Division: " << doubleResultDivide << std::endl;

return 0;

}

#include <iostream>

#include <string>

using namespace std;

// Base class

class Animal {

public:

virtual void eat(string food) {

cout << "Animal eats " << food << endl;

}

};

// Derived class Herbivore

class Herbivore : public Animal {

public:

void eat(string food) override {

cout << "Herbivore eats " << food << " (plants)" << endl;

}

};

// Derived class Carnivore

class Carnivore : public Animal {

public:

void eat(string food) override {

cout << "Carnivore eats " << food << " (meat)" << endl;

}

};

// Derived class Omnivore

class Omnivore : public Animal {

public:

void eat(string food) override {

cout << "Omnivore eats " << food << " (plants and meat)" << endl;

}

};

int main() {

Herbivore herbivore;

Carnivore carnivore;

Omnivore omnivore;

herbivore.eat("grass");

carnivore.eat("rabbit");

omnivore.eat("fruits and insects");

return 0;

}

#include <iostream>

#include <string>

using namespace std;

class Shape

{

protected:

string ShapeName;

int area;

public:

Shape(string name = "Undefined", int area = 0) : ShapeName(name), area(area) {}

void displayarea() {

cout << "Shape: " << ShapeName << ", Area: " << area << endl;

}

};

class Circle : public Shape {

public:

Circle(string name, int radius) : Shape(name, 3.14 \* radius \* radius) {}

};

class Polygon : public Shape {

public:

Polygon(string name, int sideLength) : Shape(name, sideLength \* sideLength) {}

};

kl

int main() {

Shape shape("Generic Shape", 0);

Circle circle("Circle", 5);

Polygon polygon("Polygon", 4);

shape.displayarea();

circle.displayarea();

polygon.displayarea();

return 0;

}

#include<iostream>

using namespace std;

class cafeservice

{

protected:

string order\_id;

double price;

public:

cafeservice():order\_id("#ord0"),price(0.0){}

cafeservice(string order, double pr)

{

order\_id = order;

price = pr;

}

};

class staffservice :public cafeservice

{

private:

double service\_fee;

int cabin\_num;

public:

staffservice(string id, double pr, double fee, int cabin):cafeservice(id,pr)

{

service\_fee = fee;

cabin\_num = cabin;

}

double total\_charges()

{

return price + service\_fee;

}

void display()

{

cout << " total charges: " << total\_charges() << endl;

}

};

void main()

{

staffservice obj("#12g", 200, 50, 5);

obj.display();

}

#include<iostream>

using namespace std;

class Calculator

{

public:

template<typename t>

t add(t a, t b)

{

t sum = a + b;

return sum;

}

template<typename t>

t sub(t a, t b)

{

t sub = a - b;

return sub;

}

template<typename t>

t multiply(t a, t b)

{

t multiply = a \* b;

return multiply;

}

template<typename t>

t div(t a, t b)

{

if (b != 0)

{

t div = a / b;

return div;

}

else

{

cout << " error! divisible by 0..." << endl;

return 0;

}

}

};

void main()

{

Calculator calculator;

// Test with int data type

int intResultAdd = calculator.add(5, 3);

int intResultSubtract = calculator.sub(8, 2);

int intResultMultiply = calculator.multiply(4, 6);

int intResultDivide = calculator.div(10, 2);

// Test with double data type

double doubleResultAdd = calculator.add(5.5, 3.3);

double doubleResultSubtract = calculator.sub(8.6, 2.1);

double doubleResultMultiply = calculator.multiply(4.2, 6.7);

double doubleResultDivide = calculator.div(10, 0);

// Display results

std::cout << "Results with int data type:" << std::endl;

std::cout << "Addition: " << intResultAdd << std::endl;

std::cout << "Subtraction: " << intResultSubtract << std::endl;

std::cout << "Multiplication: " << intResultMultiply << std::endl;

std::cout << "Division: " << intResultDivide << std::endl;

std::cout << "\nResults with double data type:" << std::endl;

std::cout << "Addition: " << doubleResultAdd << std::endl;

std::cout << "Subtraction: " << doubleResultSubtract << std::endl;

std::cout << "Multiplication: " << doubleResultMultiply << std::endl;

std::cout << "Division: " << doubleResultDivide << std::endl;

}

#include<iostream>

using namespace std;

class Animal

{

public:

virtual void eat(string food)

{

cout << " Animal eats: " << food << endl;

}

};

class Herbivore : public Animal

{

public:

void eat(string food) override

{

cout << " Herbivore eats: " << food<< endl;

}

};

class Carnivore : public Animal

{

public:

void eat(string food)override

{

cout << " Carnivore eats: " << food << endl;

}

};

void main()

{

Animal animal;

animal.eat("Food");

Herbivore herb;

herb.eat("grass");

Carnivore carn;

carn.eat("insects");

}

#include<iostream>

using namespace std;

class shape

{

protected:

string shape\_name;

int area;

public:

shape(string name="undefined", int ar=0):shape\_name(name),area(ar){}

virtual void display\_area()

{

cout << " shape name :" << shape\_name << " and its area: " << area << endl;

}

};

class circle :public shape

{

public:

circle(string name,int radius):shape(name,3.14\*radius\*radius){}

};

class polygon :public shape

{

public:

polygon(string name,int radius):shape(name,radius\*radius){}

};

int main()

{

shape s("generic shape",0);

s.display\_area();

circle c("circle", 4);

c.display\_area();

polygon p("polygon", 5);

p.display\_area();

}

#include<string>

#include<iostream>

using namespace std;

class laboratory

{

protected:

string name;

string location;

public:

laboratory(string n="no\_name", string l="null"):name(n),location(l){}

virtual void input()

{

cout << " Enter name: " << endl;

cin >> name;

cout << " Enter location: " << endl;

cin >> location;

}

virtual void show()

{

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

cout << "location :" << location << endl;

}

};

class wet\_lab :public laboratory

{

private:

int num\_of\_microscopes;

string scientist\_name;

public:

void set\_microscopes(int nom)

{

num\_of\_microscopes = nom;

}

void setscientistname(string s\_n)

{

scientist\_name = s\_n;

}

int get\_nom()

{

return num\_of\_microscopes;

}

string get\_name()

{

return scientist\_name;

}

void input() override

{

laboratory::input();

cout << " Enter number of microscopes: " << endl;

cin >> num\_of\_microscopes;

cout << " enter scientist name:" << endl;

cin >> scientist\_name;

}

void show() override

{

laboratory::show();

cout << " number of microscopes: " << num\_of\_microscopes << endl;

cout << " scientist name: " << scientist\_name << endl;

}

};

class Drylab:public laboratory

{

private:

int num\_of\_comp;

int capacity;

public:

void set\_num(int n)

{

num\_of\_comp = n;

}

void set\_capacity(int cap)

{

capacity = cap;

}

int get\_num()

{

return num\_of\_comp;

}

int get\_cap()

{

return capacity;

}

void input() override

{

laboratory::input();

cout << " number of computers: " << endl;

cin >> num\_of\_comp;

cout << " capacity: " << endl;

cin >> capacity;

}

void show() override

{

laboratory::show();

cout << " Number of computers: " << num\_of\_comp << endl;

cout << " Capacity: " << capacity << endl;

}

};

void main()

{

laboratory obj;

obj.input();

obj.show();

wet\_lab wet;

wet.input();

wet.show();

Drylab dry;

dry.input();

dry.show();

}

#include<iostream>

using namespace std;

class record

{

protected:

int roll\_num;

string course1\_name, course2\_name;

public:

record(){}

record(int roll, const string & c1, const string & c2)

{

roll\_num = roll;

course1\_name = c1;

course2\_name = c2;

}

int get\_roll\_num()

{

return roll\_num;

}

string get\_c1()

{

return course1\_name;

}

string get\_c2()

{

return course2\_name;

}

};

class course\_record :public record

{

protected:

int marks\_c1;

int marks\_c2;

public:

course\_record(){}

course\_record(int roll, const string& c1, const string& c2,int m1, int m2):record(roll,c1,c2)

{

marks\_c1 = m1;

marks\_c2 = m2;

}

int get\_m1()

{

return marks\_c1;

}

int get\_m2()

{

return marks\_c2;

}

};

class course\_result : public course\_record

{

private:

int total\_marks;

public:

course\_result(int roll, const string& c1, const string& c2, int m1, int m2):course\_record(roll,c1,c2,m1,m2){}

int marks\_obtained()

{

total\_marks = marks\_c1 + marks\_c2;

return total\_marks;

}

void display()

{

cout << " Roll number of student: " << roll\_num << endl;

cout << " course 1 name: " << course1\_name << endl;

cout << " course 2 name: " << course2\_name << endl;

cout << " marks of course 1: " << marks\_c1 << endl;

cout << " marks of coure 2: " << marks\_c2 << endl;

cout << " total marks: " << marks\_obtained() << endl;

}

};

void main()

{

course\_result obj3(223020, "oop", "linear", 100, 99);

obj3.display();

}

//diamond problem

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

public:

Person(){}

Person(const string& personName, int personAge) : name(personName), age(personAge) {}

void display() const {

cout << "Name: " << name << ", Age: " << age << endl;

}

};

class Staff : virtual public Person {

protected:

int staffID;

string department;

public:

Staff(const string& personName, int personAge, int staffID, const string& dept)

: Person(personName, personAge), staffID(staffID), department(dept) {}

};

class Professor : virtual public Person {

protected:

int courseID;

string courseName;

public:

Professor(const string& personName, int personAge, int cID, const string& cName)

: Person(personName, personAge), courseID(cID), courseName(cName) {}

};

class Researcher : public Staff, public Professor {

private:

int labID;

int experimentNo;

public:

Researcher(const string& personName, int personAge, int staffID, const string& dept,

int cID, const string& cName, int labID, int expNo)

: Person(personName,personAge),Staff(personName, personAge, staffID, dept), Professor(personName, personAge, cID, cName),

labID(labID), experimentNo(expNo) {}

void display() const {

cout << " person name: " << name << endl;

cout << " Person age: " << age << endl;

cout << "Staff ID: " << staffID << ", Department: " << department << endl;

cout << "Course ID: " << courseID << ", Course Name: " << courseName << endl;

cout << "Lab ID: " << labID << ", Experiment No: " << experimentNo << endl;

}

};

int main() {

Researcher researcher("John Doe", 35, 101, "Physics Department", 501, "Quantum Physics", 201, 3);

// Displaying information using the display function

researcher.display();

return 0;

}

//inheritance

#include<iostream>

#include<cmath>

using namespace std;

class plane\_point

{

protected:

int X, Y;

public:

plane\_point(){}

plane\_point(int x, int y)

{

X = x;

Y = y;

}

int get\_X() const

{

return X;

}

int get\_Y()const

{

return Y;

}

double plane\_distance(const plane\_point &other)

{

int valX = X - other.X;

int valY = Y - other.Y;

return sqrt(valX \* valX + valY \* valY);

}

};

class space\_point:public plane\_point

{

private:

int Z;

public:

space\_point():plane\_point(0,0)

{

Z = 0;

}

space\_point(int x, int y,int z):plane\_point(x,y),Z(z){}

int get\_Z()

{

return Z;

}

double space\_distance(const space\_point& other)

{

int valX = X - other.X;

int valY = Y - other.Y;

int valZ = Z - other.Z;

return sqrt(valX \* valX + valY \* valY + valZ \* valZ);

}

};

int main()

{

plane\_point point2D1(2, 3);

space\_point point3D1(3, 2, 4);

plane\_point point2D2(1, 5);

space\_point point3D2(2, 4, 6);

cout << " Distance in 2D plane: " << point2D1.plane\_distance(point2D2) << endl;

cout << " Distance in 3D plane: " << point3D1.space\_distance(point3D2) << endl;

}

#include<iostream>

using namespace std;

class polygon

{

protected:

int length;

int width;

public:

void values(int v1, int v2)

{

length = v1;

width = v2;

}

virtual int find\_area()

{

return 0;

}

};

class rectangle :public polygon

{

public:

int find\_area()

{

return length \* width;

}

};

class triangle :public polygon

{

public:

int find\_area()

{

return (length \* width) / 2;

}

};

int main()

{

rectangle rec;

triangle tri;

polygon poly;

polygon\* ptr = &rec;

polygon\* ptr2 = &tri;

polygon\* ptr3 = &poly;

ptr->values(3, 4);

ptr2->values(4, 6);

ptr3->values(3, 1);

cout <<"rectangle: "<< ptr->find\_area() << endl;

cout << "triangle: " << ptr2->find\_area() << endl;

cout << "polygon: " << ptr3->find\_area() << endl;

return 0;

}

#include<iostream>

using namespace std;

class animal

{

protected:

string name;

string zoo;

public:

animal(const string& n, const string& z)

{

name = n;

zoo = z;

}

virtual void show()

{

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

cout << " zoo: " << zoo << endl;

}

};

class birds : public animal

{

private:

bool flying;

public:

birds(const string& n, const string& z, bool fly):animal(n,z)

{

flying = fly;

}

void show()override

{

animal::show();

cout << " can fly? " << (flying ? "yes": "no") << endl;

}

};

class reptiles :public animal

{

private:

int length;

public:

reptiles(const string& n, const string& z,int l):animal(n,z),length(l){}

void show()override

{

animal::show();

cout << " length: " << length <<"meters"<< endl;

}

};

void main()

{

//animal ani("lion", "islamabad");

birds bird("parrot","islamabad",1);

reptiles rep("snake", "islamabad", 5);

// animal\* ptr1 = &ani;

animal\* ptr2 = &bird;

animal\* ptr3 = &rep;

ptr2->show();

ptr3->show();

}

#include<iostream>

using namespace std;

class person

{

protected:

string name;

int age;

public:

person(const string & n,int a):name(n),age(a){}

virtual void print()const = 0;

virtual ~person()

{

}

};

class student :public person

{

protected:

int id;

double cgpa;

public:

student(const string & n, int a,int ID,double CGPA):person(n,a),id(ID),cgpa(CGPA){}

void print() const override

{

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

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

cout << " Id : " << id << endl;

cout << " Cgpa: " << cgpa << endl;

}

};

class regular :public student

{

private:

string school\_name;

int fee;

public:

regular(const string& n, int a, int ID, double CGPA, const string& name, int f) :student(n, a, ID, CGPA)

{

school\_name = name;

fee = f;

}

void print() const override

{

student::print();

cout << " School name: " << school\_name << endl;

cout << " Fee: " << fee << endl;

}

};

class Private : public student

{

private:

string address;

int fee;

public:

Private(const string& n, int a, int ID, double CGPA, const string add, int f) :student(n, a, ID, CGPA)

{

address = add;

fee = f;

}

void print()const override

{

student::print();

cout << " Address : " << address << endl;

cout << " Fee: " << fee << endl;

}

};

void main()

{

student stu("noor", 20, 223020, 3.87);

regular reg("joe", 19, 2345, 2.9, "itfal", 3000);

Private pr("johm", 24, 7890, 3.9, "fazaia", 5000);

person\* ptr1 = &stu;

ptr1->print();

cout << endl;

ptr1 = &reg;

ptr1->print();

cout << endl;

ptr1 = &pr;

ptr1->print();

}

#include<iostream>

using namespace std;

class geometric\_shape

{

public:

virtual void show() = 0;

virtual int compute\_area() = 0;

~geometric\_shape(){}

};

class rectangle :public geometric\_shape

{

protected:

int length;

int width;

public:

rectangle(int l, int w)

{

length = l;

width = w;

}

void set\_length(int len)

{

length = len;

}

int get\_length()

{

return length;

}

void set\_width(int wid)

{

width = wid;

}

int get\_width()

{

return width;

}

int compute\_area() override

{

return length \* width;

}

void show()override

{

cout << " area of rectangle: " <<rectangle::compute\_area() << endl;

}

};

class cuboid :public rectangle

{

private:

int height;

public:

cuboid(int l, int w,int h):rectangle(l,w),height(h){}

void set\_height(int he)

{

height = he;

}

int get\_height()

{

return height;

}

int compute\_area()override

{

return (length \* width \* height);

}

void show() override

{

cout << " area of cuboid: " << cuboid::compute\_area() << endl;

}

};

int main()

{

rectangle rec(3, 5);

cuboid cub(2, 3, 4);

geometric\_shape\* ptr=&rec;

ptr->compute\_area();

ptr->show();

cout << endl;

ptr = &cub;

ptr->compute\_area();

ptr->show();

}

#include <iostream>

using namespace std;

// Template function to exchange the values of two parameters

template <typename T>

void exchange(T& a, T& b) {

T temp = a;

a = b;

b = temp;

}

int main() {

// Test with int

int intValue1 = 5, intValue2 = 10;

cout << "Before exchange - Integers: " << intValue1 << ", " << intValue2 << endl;

exchange(intValue1, intValue2);

cout << "After exchange - Integers: " << intValue1 << ", " << intValue2 << endl;

// Test with long

long longValue1 = 100, longValue2 = 200;

cout << "\nBefore exchange - Longs: " << longValue1 << ", " << longValue2 << endl;

exchange(longValue1, longValue2);

cout << "After exchange - Longs: " << longValue1 << ", " << longValue2 << endl;

// Test with double

double doubleValue1 = 3.14, doubleValue2 = 2.718;

cout << "\nBefore exchange - Doubles: " << doubleValue1 << ", " << doubleValue2 << endl;

exchange(doubleValue1, doubleValue2);

cout << "After exchange - Doubles: " << doubleValue1 << ", " << doubleValue2 << endl;

// Test with char

char charValue1 = 'A', charValue2 = 'B';

cout << "\nBefore exchange - Chars: " << charValue1 << ", " << charValue2 << endl;

exchange(charValue1, charValue2);

cout << "After exchange - Chars: " << charValue1 << ", " << charValue2 << endl;

return 0;

}

#include<iostream>

using namespace std;

template<typename t>

class calculator

{

private:

t num1, num2;

public:

calculator(t n1, t n2)

{

num1 = n1;

num2 = n2;

}

t addition()

{

return num1 + num2;

}

t subtraction()

{

return num1 - num2;

}

t multiplication()

{

return num1 \* num2;

}

double division()

{

if (num2 != 0)

{

return static\_cast<double>(num1) / num2;

}

else

{

cout << " error!! " << endl;

return 0.0;

}

}

};

int main()

{

calculator <int>intcal(2, 3);

calculator <float>floatcal(3.3, 0.0);

cout<<" addition: "<<intcal.addition() << endl;

cout << " division: " << intcal.division() << endl;

}

#include<iostream>

using namespace std;

template<typename t>

t common(t &arr, int size)

{

for (int i = 0;i < size;i++)

{

return (arr + i) \* (arr + i);

}

}

#include<iostream>

using namespace std;

template<typename t>

t area(t radius)

{

return (3.14 \* radius \* radius)/2.0;

}

template<typename t>

t area(t length, t width)

{

return (3)\*(length \* width);

}

int main()

{

cout << " circle area: " << area(4) << endl;

cout << " rectangle area: " << area(2, 3) << endl;

return 0;

}

#include<iostream>

using namespace std;

template<class t>

class queue

{

private:

t arr[5];

public:

queue()

{

for (int i = 0;i < 5;i++)

{

arr[i] = 0;

}

}

void add(t& ar)

{

for (int i = 0;i < 5;i++)

{

arr[i] = \*(arr+i);

}

}

void sort()

{

for (int i = 0;i < 5;i++)

{

for (int j = 0;j < 5 - i - 1;j++)

{

if (arr[j] > arr[j + 1])

{

t temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

cout << " Sorted array: " << endl;

for (int i = 0;i < 5;i++)

{

cout << arr[i] << " ";

}

}

t max()

{

t max = arr[0];

for (int i = 0;i < 5;i++)

{

if (arr[i]>max)

{

max = arr[i];

}

}

return max;

}

t min()

{

t min = arr[0];

for (int i = 0;i < 5;i++)

{

if (arr[i] < max)

{

max = arr[i];

}

}

return min;

}

};

int main()

{

queue <int>obj;

int arr[5] = { 3,7,0,1,7 };

obj.add(\*arr);

obj.sort();

cout<<" max: "<<obj.max()<<endl;

cout << " min: " << obj.min() << endl;

}

#include<iostream>

using namespace std;

template<class t>

class check

{

private:

t var1;

//u var2;

public:

void sum(t a, t b)

{

cout << " sum: " << a + b << endl;

}

};

void main()

{

check <int>obj;

obj.sum(2, 3);

}

#include<iostream>

using namespace std;

class abc

{

public:

static float i;

static int update()

{

static int j = 2;

i++;

return j += 2;

}

};

float abc::i = 5.5;

void main()

{

abc obj1, obj2;

for (int i = 0;i < 5;i++)

{

cout << obj1.i << " " << obj1.update() << endl;

cout << obj2.i << " " << obj2.update() << endl;

}

}\*/

#include<iostream>

using namespace std;

class calculator

{

public:

static double sum(double a, double b)

{

return a + b;

}

static double sub(double a, double b)

{

return a - b;

}

static double mul(double a, double b)

{

return a \* b;

}

static double div(double a, double b)

{

if (b != 0.0)

{

return a / b;

}

else

{

cout << " error!" << endl;

return 0;

}

}

static double mod(double a, double b)

{

if (b != 0.0)

{

return fmod(a,b);

}

else

{

cout << " error!" << endl;

return 0;

}

}

static double sin(double angle)

{

return ::sin(angle);

}

static double cos(double angle)

{

return ::cos(angle);

}

static double tan(double angle)

{

return ::tan(angle);

}

};

void main()

{

cout << " result of sum: " << calculator::sum(2.3, 5.6) << endl;

cout << " result of sub: " << calculator::sub(2.3, 5.6) << endl;

cout << " result of mul: " << calculator::mul(2.3, 5.6) << endl;

cout << " result of div: " << calculator::div(2.3, 5.6) << endl;

cout << " result of mod: " << calculator::mod(2.3, 5.6) << endl;

cout << " result of sin: " << calculator::sin(2.3) << endl;

cout << " result of cos: " << calculator::cos(2.3) << endl;

cout << " result of tan: " << calculator::tan(2.3) << endl;

}