Worksheet4

Task1

Ans-

#include <iostream>

using namespace std;

class Student {

private:

int code; // Student's identifier.

int courses; // Number of courses.

float\* grd; // Pointer to dynamically allocated memory for grades.

public:

// Constructor

Student() : code(0), courses(0), grd(nullptr) {}

// Destructor to clean up dynamically allocated memory.

~Student() {

delete[] grd;

}

// Copy operations are disabled.

Student(const Student&) = delete;

Student& operator=(const Student&) = delete;

// Function to input student data

void input() {

cout << "Enter student code and number of courses: ";

cin >> code >> courses;

grd = new float[courses];

cout << "Enter grades for each course: ";

for (int i = 0; i < courses; i++) {

cin >> grd[i];

}

}

// Function to compare number of passed courses between two students

static int compare(const Student& s1, const Student& s2) {

int count1 = 0, count2 = 0;

for (int i = 0; i < s1.courses; i++) {

if (s1.grd[i] >= 5) count1++;

}

for (int i = 0; i < s2.courses; i++) {

if (s2.grd[i] >= 5) count2++;

}

if (count1 > count2) return 1;

if (count2 > count1) return 2;

return 3;

}

// Accessor for code

int getCode() const {

return code;

}

};

int main() {

Student s1, s2;

s1.input(); // Input data for first student

s2.input(); // Input data for second student

int result = Student::compare(s1, s2);

switch (result) {

case 1:

cout << "Student with code " << s1.getCode() << " has succeeded in more courses." << endl;

break;

case 2:

cout << "Student with code " << s2.getCode() << " has succeeded in more courses." << endl;

break;

default:

cout << "Both students have succeeded in the same number of courses." << endl;

break;

}

return 0;

}

Task 2

Ans-

#include <iostream>

using namespace std;

class Circle {

protected:

float rad;

public:

Circle(float radius) : rad(radius) {}

virtual Circle& operator-(int n) {

rad -= n;

if (rad < 0)

throw 10;

return \*this;

}

virtual void show() const {

cout << "Radius: " << rad << endl;

}

};

class Ellipse : public Circle {

private:

float axis;

public:

Ellipse(float radius, float ax) : Circle(radius), axis(ax) {}

virtual Ellipse& operator-(int n) {

Circle::operator-(n); // Subtract from radius and check for exception

axis -= n;

if (axis < 0)

throw 20;

return \*this;

}

void show() const override {

Circle::show();

cout << "Axis: " << axis << endl;

}

};

void f(Circle &c, int n) {

try {

Circle& tmp = c - n; // Performing the subtraction, which may throw an exception

tmp.show(); // If no exception is thrown, show the dimensions

} catch (int ex) {

if (ex == 10) {

cout << "Exception: Radius is negative!" << endl;

} else if (ex == 20) {

cout << "Exception: Axis is negative!" << endl;

}

throw; // Rethrow the exception to handle it further if needed

}

}

int main() {

try {

Circle cir(5); // Circle with radius 5

Ellipse ell(10, 6); // Ellipse with radius 10 and axis 6

Circle &r1 = cir, &r2 = ell;

f(r1, 3); // Should display Radius: 2

f(r2, 1); // Should display Radius: 9 and Axis: 5

f(r2, 10); // Should throw an exception because axis becomes negative

} catch (int) {

cout << "Terminating program due to an exception." << endl;

return 1;

}

return 0;

}

Task 3

Ans-

#include <iostream>

#include <string>

using namespace std;

class School {

private:

string name;

public:

School(const string& n) : name(n) {}

virtual ~School() {}

virtual void show() const {

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

}

};

class Programming : virtual public School {

private:

int progCourses;

public:

Programming(const string& n, int pc) : School(n), progCourses(pc) {}

int getProgCourses() const {

return progCourses;

}

void show() const override {

School::show();

cout << "Programming courses: " << progCourses << endl;

}

};

class Network : virtual public School {

private:

int netCourses;

public:

Network(const string& n, int nc) : School(n), netCourses(nc) {}

int getNetCourses() const {

return netCourses;

}

void show() const override {

School::show();

cout << "Network courses: " << netCourses << endl;

}

};

class Student : public Programming, public Network {

private:

string studentName;

int code;

public:

Student(const string& schoolName, const string& name, int c, int pc, int nc)

: School(schoolName), Programming(schoolName, pc), Network(schoolName, nc),

studentName(name), code(c) {}

void show() const override {

School::show();

cout << "Programming courses: " << getProgCourses() << endl;

cout << "Network courses: " << getNetCourses() << endl;

cout << "Student Name: " << studentName << endl;

cout << "Code: " << code << endl;

}

};

int main() {

Student s("TBC", "Santosh", 100, 7, 8);

School& p = s;

p.show();

return 0;

}