**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И. РАЗЗАКОВА**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**ОТЧЕТ**

**по лабораторной работе №7**

**Выполнила:** студент группы ПИ-3-21

Джанышбекова Акмарал

**Проверила:** Мусабаев Э. Б.

**Бишкек 2024**

**Задание №1**

Создать базовый класс Array с полями: массив типа unsigned и поле для хранения количества элементов у текущего объекта массива. Максимально возможный размер массива задается статической константой. Реализовать конструктор инициализации, задающий количество элементов и начальное значение (по умолчанию 0). Реализовать в классе Array виртуальную функцию поэлементного сложения массивов. Реализовать два класса, переопределив виртуальную функцию сложения. Вызывающая программа должна продемонстрировать все варианты вызова виртуальных функций.

#include <iostream>

using namespace std;

// Базовый класс Array

class Array {

protected:

static const int MAX\_SIZE = 100; // Максимальный размер массива

unsigned arr[MAX\_SIZE]; // Массив

int size; // Количество элементов

public:

// Конструктор инициализации

Array(int sz, unsigned initialValue = 0) : size(sz) {

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

arr[i] = initialValue;

}

}

// Виртуальная функция поэлементного сложения массивов

virtual void addArrays(const Array& other) {

int minSize = std::min(size, other.size);

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

arr[i] += other.arr[i];

}

}

// Вывод массива

void printArray() const {

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

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

}

cout << endl;

}

};

// Класс ArrayPlusOne, переопределяющий виртуальную функцию

class ArrayPlusOne : public Array {

public:

// Конструктор

ArrayPlusOne(int sz, unsigned initialValue = 0) : Array(sz, initialValue) {}

// Переопределение функции поэлементного сложения массивов

void addArrays(const Array& other) override {

Array::addArrays(other);

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

arr[i]++;

}

}

};

// Класс ArrayMinusOne, переопределяющий виртуальную функцию

class ArrayMinusOne : public Array {

public:

// Конструктор

ArrayMinusOne(int sz, unsigned initialValue = 0) : Array(sz, initialValue) {}

// Переопределение функции поэлементного сложения массивов

void addArrays(const Array& other) override {

Array::addArrays(other);

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

arr[i]--;

}

}

};

int main() {

setlocale(LC\_ALL, "Ru");

// Пример использования

Array arr1(3, 60);

Array arr2(3, 40);

ArrayPlusOne arr3(3, 10);

ArrayMinusOne arr4(3, 20);

cout << "Вывод значении: " << endl;

cout << "Вывод arr1: ";

arr1.printArray();

cout << "Вывод arr2: ";

arr2.printArray();

cout << "Вывод arr3: ";

arr3.printArray();

cout << "Вывод arr4: ";

arr4.printArray();

cout << "\n-------------------------------------------------";

cout << "\nВызов виртуальной функции поэлементного сложения" << endl;

cout << "-------------------------------------------------" << endl;

cout << "arr1 + arr2: ";

cout << "\n60 + 40: ";

arr1.addArrays(arr2);

arr1.printArray();

cout << "arr2 + arr4: ";

cout << "\n40 + 20: ";

arr2.addArrays(arr4);

arr2.printArray();

return 0;

}

**Задание №2**

Создать абстрактный базовый класс Figure с виртуальными методами вычисления площади и периметра. Создать производные класса: Rectangle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие в производных. Площадь трапеции: *S = (a + b) x h/2.*

#include <iostream>

using namespace std;

// Абстрактный базовый класс Figure

class Figure {

public:

// Виртуальные методы для вычисления площади и периметра

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

// Производный класс Rectangle

class Rectangle : public Figure {

protected:

double length;

double width;

public:

// Конструктор инициализации

Rectangle(double l, double w) : length(l), width(w) {}

// Реализация виртуальных методов

double area() const override {

return length \* width;

}

double perimeter() const override {

return 2 \* (length + width);

}

};

// Производный класс Circle

class Circle : public Figure {

protected:

double radius;

public:

// Конструктор инициализации

Circle(double r) : radius(r) {}

// Реализация виртуальных методов

double area() const override {

return 3.14 \* radius \* radius;

}

double perimeter() const override {

return 2 \* 3.14 \* radius;

}

};

// Производный класс Trapezium

class Trapezium : public Figure {

protected:

double sideA;

double sideB;

double height;

public:

// Конструктор инициализации

Trapezium(double a, double b, double h) : sideA(a), sideB(b), height(h) {}

// Реализация виртуальных методов

double area() const override {

return (sideA + sideB) \* height / 2.0;

}

double perimeter() const override {

return sideA + sideB + 2 \* height;

}

};

int main() {

setlocale(LC\_ALL, "Ru");

Rectangle rectangle(5, 3);

Circle circle(4);

Trapezium trapezium(3, 7, 4);

cout << "Результат производного класса Rectangle:" << endl;

cout << "Area: " << "5 \* 3 = " << rectangle.area() << endl;

cout << "Area: " << rectangle.area() << endl;

cout << "\nPerimeter: " << "2 \* (5 + 3) = " << rectangle.perimeter() << endl;

cout << "Perimeter: " << rectangle.perimeter() << endl;

cout << "----------------------------------------\n";

cout << "\nРезультат производного класса Circle:" << endl;

cout << "Area: " << "3.14 \* 4 \* 4 = " << circle.area() << endl;

cout << "Area: " << circle.area() << endl;

cout << "\nPerimeter: " << "2 \* 3,14 \* 4 = " << circle.perimeter() << endl;

cout << "Perimeter: " << circle.perimeter() << endl;

cout << "----------------------------------------\n";

cout << "\nРезультат производного класса Trapezium:" << endl;

cout << "Area: " << "(3 + 7) \* 4 / 2.0 = " << trapezium.area() << endl;

cout << "Area: " << trapezium.area() << endl;

cout << "\nPerimeter: " << "3 + 7 + 2 \* 4 = " << trapezium.perimeter() << endl;

cout << "Perimeter: " << trapezium.perimeter() << endl;

return 0;

}

**Задание №3**

Создать абстрактный базовый класс Currency (валюта) для работы с денежными суммами. Определить виртуальные функции перевода в рубли и вывода на экран. Реализовать производные классы Dollar (доллар) и Euro (евро) со своими функциями перевода и вывода на экран.

#include <iostream>

#include <iomanip>

using namespace std;

// Абстрактный базовый класс

class Currency {

public:

// Виртуальная функция для конвертации в рубли

virtual double toRubles(double amount, double exchangeRate) const = 0;

// Виртуальная функция для отображения типа валюты

virtual void display() const = 0;

// Виртуальный деструктор для правильного удаления объектов

virtual ~Currency() {}

};

// Производный класс Dollar

class Dollar : public Currency {

public:

// Реализация функции конвертации в рубли

double toRubles(double amount, double exchangeRate) const override {

return amount \* exchangeRate;

}

// Реализация функции отображения типа валюты

void display() const override {

cout << "Тип валюты: Dollar" << endl;

}

};

// Производный класс Euro

class Euro : public Currency {

public:

// Реализация функции конвертации в рубли

double toRubles(double amount, double exchangeRate) const override {

return amount \* exchangeRate;

}

// Реализация функции отображения типа валюты

void display() const override {

cout << "Тип валюты: Euro" << endl;

}

};

int main() {

setlocale(LC\_ALL, "Ru");

// Создание объектов классов Dollar и Euro

Dollar dollar;

Euro euro;

double amount, exchangeRate;

cout << "Производный класс Dollar: " << endl;

cout << "\nВведите сумму в долларах: ";

cin >> amount;

cout << "Введите текущий курс доллара к рублю: ";

cin >> exchangeRate;

// Отображение результата для долларов

cout << "\nРезультат для Долларов:\n";

dollar.display();

cout << "Сумма в рублях: " << dollar.toRubles(amount, exchangeRate) << endl;

cout << endl << "----------------------------------\n";

cout << "Производный класс Euro: " << endl;

cout << "\nВведите сумму в евро: ";

cin >> amount;

cout << "Введите текущий курс евро к рублю: ";

cin >> exchangeRate;

// Отображение результата для евро

cout << "\nРезультат для Евро:\n";

euro.display();

cout << "Сумма в рублях: " << euro.toRubles(amount, exchangeRate) << endl;

return 0;

}

**ВОПРОСЫ К ЗАЩИТЕ ЛАБОРАТОРНОЙ РАБОТЫ №7**

1. *Что такое виртуальная функция (метод) при наследовании?*

Это функция, объявленная в базовом классе и предназначенная для переопределения в производных классах, обеспечивая полиморфизм.

1. *Чем отличается переопределение метода от его перегрузки при наследовании?*

При перегрузке метода меняется прототип метода, то есть создаются несколько методов с одним и тем же именем, но разными параметрами. При переопределении метода прототип остается неизменным, но реализация функции может быть изменена в производном классе.

1. *Что такое чисто виртуальная функция? Для чего она используется*?

Чисто виртуальной функцией называется такая функция, которая не имеет определения в базовом классе, или иными словами, не имеет никакой реализации в этом классе. Её присутствие обозначает, что класс является абстрактным, и его целью является предоставление интерфейса для производных классов, которые должны предоставить свою конкретную реализацию этой функции.

1. *Синтаксис виртуальной функции?*

Синтаксис виртуальной функции в C++ выглядит следующим образом:

virtual тип\_возвращаемого\_значения имя\_функции(параметры) = 0;

class Base {

public:

virtual void virtualFunction() = 0;

};

Здесь **virtual** указывает на то, что функция является виртуальной, а = 0 делает её чисто виртуальной, тем самым делая класс абстрактным.

1. *Как называется класс, содержащий хотя бы одну чисто виртуальную функцию?*

Класс, содержащий хотя бы одну чисто виртуальную функцию, называется абстрактным классом.

1. *Приведите пример иерархической структуры при наследовании?*

class Shape {

public:

virtual void draw() const = 0;

};

class Circle : public Shape {

public:

void draw() const override {

// реализация для круга

}

};

class Square : public Shape {

public:

void draw() const override {

// реализация для квадрата

}

};

class Triangle : public Shape {

public:

void draw() const override {

// реализация для треугольника

}

};

В данном примере класс Shape является абстрактным классом с чисто виртуальной функцией draw(). От него наследуются классы Circle, Square и Triangle, предоставляя свои конкретные реализации метода draw(). Это представляет собой иерархию форм (геометрических фигур) с общим интерфейсом.

1. *Как использовать виртуальные методы в многоуровневом наследовании?*

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

public:

Person(const std::string& n) : name(n) {}

virtual void display() const {

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

}

};

class Student : public Person {

private:

int studentID;

public:

Student(const std::string& n, int id) : Person(n), studentID(id) {}

void display() const override {

Person::display();

std::cout << "Student ID: " << studentID << std::endl;

}

};

class Teacher : public Person {

private:

std::string subject;

public:

Teacher(const std::string& n, const std::string& subj) : Person(n), subject(subj) {}

void display() const override {

Person::display();

std::cout << "Subject: " << subject << std::endl;

}

};

int main() {

Student student("John Doe", 12345);

Teacher teacher("Dr. Smith", "Computer Science");

Person\* ptrPerson1 = &student;

Person\* ptrPerson2 = &teacher;

ptrPerson1->display();

std::cout << "---------------------\n";

ptrPerson2->display();

return 0;

}

В этом примере Person - абстрактный базовый класс, Student и Teacher - производные классы. Оба производных класса переопределяют виртуальную функцию display(). Затем в функции main() создаются объекты классов Student и Teacher, и их адреса присваиваются указателям на базовый класс Person. Вызов метода display() через указатели на базовый класс приводит к вызову соответствующих переопределённых методов в производных классах.