3- Практическое задание – Создание базового класса в C++

Table of Contents

Цель

Эта лабораторная работа направлена на знакомство студентов с объектно-ориентированным программированием на C++ посредством проектирования и манипуляции классами. Студенты научатся инкапсулировать данные и функциональность внутри классов, создавая объекты для выполнения конкретных операций.

1. Проектирование простого класса Книга

Цель: понять структуру класса, основные переменные-члены и функции.

Описание: спроектировать класс Book с следующими спецификациями:

Переменные-члены: title (string), author (string), publicationYear (int) и price (float).

Функции-члены: Конструктор для инициализации всех переменных-членов, метод для отображения деталей книги и метод для обновления цены книги.

Активность:

- Реализуйте класс с указанными переменными-членами и функциями.
- Создайте объект класса Book в главной функции и инициализируйте его соответствующими значениями.
- Отобразите детали книги, а затем обновите цену, показывая обновленные детали после этого.

Чтобы справиться с этим заданием, мы пройдем процесс шаг за шагом, разбивая компоненты класса C++, как реализовать его переменные-члены и функции, а также как использовать класс в основной программе.

Шаг 1: Понимание класса в C++

Класс в C++ — это чертеж для создания объектов. Он инкапсулирует данные для объекта (через переменные-члены) и методы для манипуляции этими данными (через функции-члены). В данном случае наш класс представляет Книгу с атрибутами, такими как название, автор, год публикации и цена.

Шаг 2: Определение класса Книга

Сначала мы определяем наш класс с требуемыми переменными-членами и функциями. Вот как это делается:

```cpp
#include <iostream>
#include <string>

using namespace std;

class Book {
public:
    // Member Variables
    string title;
    string author;
    int publicationYear;
    float price;

    // Constructor
    Book(string t, string a, int year, float p) : title(t), author(a), publicationYear(year), price(p) {}

    // Member Function to display book details
    void displayDetails() {
        cout << "Title: " << title << endl;
        cout << "Author: " << author << endl;
```

```
        cout << "Publication Year: " << publicationYear << endl;
        cout << "Price: $" << price << endl;
    }

    // Member Function to update the book's price
    void updatePrice(float newPrice) {
        price = newPrice;
    }
};
```

Объяснение:

- **Переменные-члены:** Эти переменные хранят данные книги. У нас есть название, автор, год публикации и цена, как указано.
- **Конструктор:** Особая функция-член с таким же именем, как класс. Она используется для инициализации объектов своего класса. Здесь она принимает параметры для каждой переменной-члена и использует список инициализаторов (: title(t), author(a), publicationYear(year), price(p)) для присвоения этих параметров переменным-членам.
- **displayDetails():** Эта функция печатает детали книги. Мы используем cout для вывода.
- **updatePrice(float newPrice):** Эта функция обновляет цену книги с новым значением, переданным в качестве параметра.

Шаг 3: Использование класса Книга в главной функции

Теперь давайте посмотрим, как создать объект класса Книга, инициализировать его, отобразить его детали, обновить цену и затем отобразить обновленные детали.

```
int main() {
    // Creating an object of Book class and initializing it
    Book myBook("The Great Gatsby", "F. Scott Fitzgerald", 1925, 20.99);

    // Displaying book details
    cout << "Initial book details:" << endl;
    myBook.displayDetails();

    // Updating the book's price
    myBook.updatePrice(15.99);

    // Displaying updated book details
    cout << "\nUpdated book details:" << endl;
    myBook.displayDetails();

    return 0;
}
```

Объяснение:

- Мы создаем объект myBook класса Книга, инициализируя его конкретными значениями для названия, автора, года публикации и цены.
- Затем мы вызываем displayDetails() для вывода начальных деталей книги.
- Мы обновляем цену книги с помощью updatePrice() и передаем новую цену в качестве аргумента.
- Наконец, мы снова вызываем displayDetails(), чтобы показать обновленные детали книги.
- Эта программа вводит концепцию классов, создания объектов и основных операций с данными объекта. Это фундаментальное упражнение для понимания объектно-ориентированного программирования в C++.

2. Реализация класса Student с методами

Для этого задания мы продолжим развивать основные концепции объектно-ориентированного программирования, реализуя класс Student. Этот пример будет сосредоточен на инкапсуляции информации о студентах, выполнении операций с этими данными и важности конструкторов и функций-членов для эффективного управления объектами.

Шаг 1: Понимание требований

Нам нужно создать класс Student, который инкапсулирует детали студента, такие как имя, ID и оценки за экзамены и задания. Кроме того, мы реализуем функциональность для расчета итоговой оценки как простого среднего между оценками за экзамены и задания, а также метод для отображения деталей студента вместе с рассчитанной оценкой.

Шаг 2: Определение класса Student

Давайте определим наш класс Student с необходимыми переменными-членами и функциями:

```cpp
#include <iostream>
#include <string>

using namespace std;

class Student {
public:
    // Member Variables
    string name;
    string id;
    double examScore;
    double assignmentScore;

    // Constructor
    Student(string n, string i) : name(n), id(i), examScore(0), assignmentScore(0) {}

    // Method to input scores
    void inputScores(double exam, double assignment) {
        examScore = exam;
        assignmentScore = assignment;
    }

    // Method to calculate final grade
    double calculateGrade() {
        return (examScore + assignmentScore) / 2.0;
    }

    // Method to display student details and grade
    void displayDetails() {
        cout << "Student Name: " << name << endl;
        cout << "Student ID: " << id << endl;
        cout << "Exam Score: " << examScore << endl;
        cout << "Assignment Score: " << assignmentScore << endl;
        cout << "Final Grade: " << calculateGrade() << endl;
    }
};
```

Объяснение:

- Переменные-члены: name, id, examScore и assignmentScore хранят информацию и оценки студента.
- Конструктор: Инициализирует name и id предоставленными значениями и устанавливает начальные оценки равными 0. Этот подход позволяет установить оценки позже через специфический метод.

- inputScores(double exam, double assignment): Принимает оценки за экзамен и задание в качестве входных данных и присваивает их соответствующим переменным-членам.
- calculateGrade(): Рассчитывает итоговую оценку как среднее между оценками за экзамены и задания. Возвращает это значение как double.
- displayDetails(): Выводит детали студента, включая его рассчитанную итоговую оценку, на консоль.

Шаг 3: Использование класса Student в главной функции

Теперь давайте посмотрим, как использовать класс Student в программе:

```
int main() {
    // Creating a Student object
    Student student1("John Doe", "S1234567");

    // Inputting scores
    student1.inputScores(85.5, 92.3);

    // Displaying student details and final grade
    student1.displayDetails();

    return 0;
}
```

Объяснение:

- Мы создаем объект Student с именем student1 и инициализируем его именем и ID.
- Затем мы используем inputScores() для установки оценок за экзамен и задание для student1.
- Наконец, вызываем displayDetails() для вывода всех деталей student1, включая рассчитанную итоговую оценку.

Это упражнение дополнительно укрепляет концепцию инкапсуляции, сохраняя данные студента и операции с этими данными внутри класса Student, демонстрируя мощь объектно-ориентированного программирования в организации и управлении сложными структурами данных.

3. Создание класса Rectangle для геометрических операций

Цель: исследовать функциональность класса с геометрическими расчетами.

Описание: разработать класс Rectangle для выполнения операций с прямоугольником.

Переменные-члены: длина (double) и ширина (double).

Функции-члены: Конструктор, метод для расчета и возвращения площади, метод для расчета и возвращения периметра и метод для проверки, является ли он квадратом.

Активность:

- Закодируйте класс Rectangle с указанными особенностями.
- Создайте несколько объектов Rectangle в главной функции, выполните операции и отобразите результаты для демонстрации функциональности класса.

В этом задании мы собираемся углубиться в применение классов для геометрических операций, сосредотачиваясь на классе Rectangle. Это покажет, как классы могут инкапсулировать как данные, так и операции, связанные с конкретной сущностью — в данном случае, с геометрической фигурой. Начнем с обзора шагов, которые мы предпримем для определения и использования класса Rectangle.

Шаг 1: Определение класса Rectangle

Нам нужно создать класс Rectangle, который включает:

- Переменные-члены для длины и ширины прямоугольника.
- Конструктор для инициализации этих переменных.
- Методы для расчета площади и периметра.
- Метод для проверки, является ли прямоугольник также квадратом (то есть длина равна ширине).

Вот как может выглядеть определение класса:

```
#include <iostream>
using namespace std;

class Rectangle {
public:
    // Member Variables
    double length;
    double width;

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

    // Method to calculate area
    double calculateArea() {
        return length * width;
    }

    // Method to calculate perimeter
    double calculatePerimeter() {
        return 2 * (length + width);
    }

    // Method to check if it is a square
    bool isSquare() {
```

```
        return length == width;
    }
};
```

Объяснение:

- Конструктор: принимает два параметра (длину и ширину) и инициализирует переменные-члены.
- calculateArea(): Возвращает произведение длины и ширины.
- calculatePerimeter(): Возвращает удвоенную сумму длины и ширины.
- isSquare(): Возвращает true, если длина и ширина равны, указывая на то, что прямоугольник является квадратом; в противном случае возвращает false.

Шаг 2: Использование класса Rectangle

Теперь давайте продемонстрируем, как создавать и использовать объекты Rectangle в главной функции:

```
int main() {
    // Creating Rectangle objects
    Rectangle rect1(10.5, 5.5);
    Rectangle rect2(7.0, 7.0);

    // Performing operations on rect1
    cout << "Rectangle 1 Area: " << rect1.calculateArea() << endl;
    cout << "Rectangle 1 Perimeter: " << rect1.calculatePerimeter() << endl;
    cout << "Is Rectangle 1 a Square?: " << (rect1.isSquare() ? "Yes" : "No") << endl;

    // Performing operations on rect2
    cout << "\nRectangle 2 Area: " << rect2.calculateArea() << endl;
    cout << "Rectangle 2 Perimeter: " << rect2.calculatePerimeter() << endl;
    cout << "Is Rectangle 2 a Square?: " << (rect2.isSquare() ? "Yes" : "No") << endl;

    return 0;
}
```

Объяснение:

- Мы создаем два объекта Rectangle: rect1 с длиной 10.5 и шириной 5.5, и rect2 с длиной и шириной 7.0 (что делает его квадратом).
- Для каждого прямоугольника мы рассчитываем и выводим площадь и периметр, проверяем, является ли он квадратом, и отображаем результаты на консоли.

Это задание демонстрирует силу объектно-ориентированного программирования в решении реальных проблем, таких как геометрические расчеты, путем инкапсуляции связанных данных и функциональностей внутри класса. Этот подход упрощает организацию кода, делая его проще для понимания, поддержки и расширения.

4. Создание класса Timer для базового отслеживания времени

Цель: ввести понятие состояния в классе на примере простого приложения для отслеживания времени.

Описание: построить класс Timer, который может запускать, останавливать и сбрасывать таймер, а также отображать прошедшее время в секундах.

Переменные-члены: startTime (int), endTime (int) и isRunning (bool).

Функции-члены: Методы для запуска, остановки, сброса таймера и метод для отображения прошедшего времени.

Активность:

- Реализуйте класс Timer как определено.
- В главной функции создайте объект Timer для демонстрации запуска, остановки и сброса таймера, включая отображение прошедшего времени.

Для задания 4 мы сосредотачиваемся на создании класса Timer, который имитирует базовую функциональность отслеживания времени. Этот класс введет понятие поддержания состояния в объекте через переменные, которые отслеживают время начала, время окончания и то, работает ли таймер в данный момент.

```
#include <iostream>
#include <ctime>
using namespace std;

class Timer {
private:
    clock_t startTime;
    clock_t endTime;
    bool isRunning;

public:
    Timer() : startTime(0), endTime(0), isRunning(false) {}

    void start() {
        if (!isRunning) {
            startTime = clock();
            isRunning = true;
        }
    }

    void stop() {
        if (isRunning) {
            endTime = clock();
            isRunning = false;
        }
    }

    void reset() {
        startTime = 0;
        endTime = 0;
        isRunning = false;
    }

    void displayElapsedTime() {
        double elapsed = (isRunning ? clock() : endTime) - startTime;
        cout << "Elapsed Time: " << (elapsed / CLOCKS_PER_SEC) << " seconds" << endl;
```

```
    }
};

int main() {
    Timer t;

    t.start();
    // Simulate some operation
    for (int i = 0; i < 100000000; i++);
    t.stop();

    t.displayElapsedTime();

    t.reset();
    t.displayElapsedTime();

    return 0;
}
```

Объяснение:

```
#include <iostream>
#include <ctime>
using namespace std;

class Timer {
private:
    clock_t startTime;
    clock_t endTime;
    bool isRunning;
```

Переменные:

- startTime и endTime имеют тип clock_t, который подходит для хранения значений времени процессора.
- isRunning — это булева переменная, которая отслеживает, работает ли таймер в настоящее время.

```
public:
    Timer() : startTime(0), endTime(0), isRunning(false) {}
```

Конструктор: инициализирует startTime и endTime значением 0 и isRunning значением false. Эта настройка гарантирует, что таймер готов к запуску и не работает при создании объекта.

```
    void start() {
        if (!isRunning) {
            startTime = clock();
            isRunning = true;
        }
    }
```

Метод start(): Если таймер еще не запущен (!isRunning), он устанавливает startTime в текущее время процессора с использованием функции clock() и отмечает таймер как работающий. Этот метод гарантирует, что таймер начинает отслеживание времени с этого момента.

```
void stop() {
        if (isRunning) {
            endTime = clock();
            isRunning = false;
        }
    }
```

Метод stop(): Если таймер работает, он захватывает текущее время процессора в endTime и отмечает таймер как не работающий. Это действие останавливает таймер.

```
void reset() {
        startTime = 0;
        endTime = 0;
```

```
        isRunning = false;
    }
```

Метод reset(): Сбрасывает startTime и endTime на 0 и отмечает таймер как не работающий. Этот метод подготавливает таймер к новой сессии измерения времени.

```
    void displayElapsedTime() {
        double elapsed = (isRunning ? clock() : endTime) - startTime;
        cout << "Elapsed Time: " << (elapsed / CLOCKS_PER_SEC) << " seconds" << endl;
    }
```

Метод displayElapsedTime(): Рассчитывает прошедшее время путем вычитания startTime из текущего времени (clock()), если таймер все еще работает, или из endTime, если остановлен. Результат делится на CLOCKS_PER_SEC для перевода из единиц времени процессора в секунды, а затем отображается.

CLOCKS_PER_SEC: В предоставленном примере класса Timer, CLOCKS_PER_SEC — это константа в C++, определенная в заголовочном файле <ctime>. Она представляет количество тактов часов в секунду. Функция clock(), также определенная в <ctime>, возвращает время процессора, потраченное программой с момента начала выполнения программы. Значение, возвращаемое функцией clock(), измеряется в "тактах часов", которые являются единицами времени постоянной, но системно-специфической длины, определяемой CLOCKS_PER_SEC. Таким образом, для преобразования разницы между двумя показаниями clock() (которые дают вам количество тактов часов) в секунды, вы делите на CLOCKS_PER_SEC. Это преобразование необходимо, потому что clock() не возвращает время непосредственно в секундах. Формула, используемая в методе displayElapsedTime() класса Timer. Эта константа обеспечивает переносимость между различными платформами, абстрагируя системно-специфическую частоту тактов часов. Это гарантирует, что программа вычисляет время в секундах одинаково на различных компьютерных системах, несмотря на потенциальные различия в том, как эти системы внутренне измеряют время процессора.

```
int main() {
    Timer t;

    t.start();
    // Simulate some operation
    for (int i = 0; i < 100000000; i++);
    t.stop();

    t.displayElapsedTime();

    t.reset();
    t.displayElapsedTime();

    return 0;
}
```

1. Создает объект таймера t.
2. Запускает таймер, имитирует длительную по времени операцию с помощью цикла, а затем останавливает таймер.
3. Отображает прошедшее время, показывая, сколько времени заняла операция.
4. Сбрасывает таймер, а затем сразу же снова отображает прошедшее время, которое должно быть равно 0 или близко к 0, поскольку таймер был сброшен и остановлен до того, как произошла любая значимая операция.

Этот пример демонстрирует, как управлять состоянием внутри класса на практике, вводя базовые концепции измерения времени и оценки производительности.

Результат, который вы видите, вероятно, обусловлен очень быстрым временем выполнения симулированных операций внутри циклов for, в сочетании с гранулярностью функции clock(), используемой в классе Timer.

Функция clock() измеряет время ЦПУ, использованное программой, которое может быть очень маленьким для простых операций и может не точно отражать реальное "стенное" время для коротких интервалов, особенно в быстро выполняющемся цикле, который не выполняет много работы.

Почему результаты могут показаться нелогичными:

Время ЦПУ против реального времени: Функция clock() измеряет время ЦПУ, а не реальное прошедшее время. Если ЦПУ тратит очень мало времени на ваши циклы (потому что они просты и выполняются очень быстро), измеренное время может быть около нуля.

Гранулярность clock(): Разрешение clock() ограничено (часто до миллисекунд), и быстрые операции могут быть недостаточно длинными, чтобы зарегистрировать заметную разницу, особенно если они не потребляют значительных ресурсов ЦПУ.

Решения для повышения точности:

Чтобы получить более значимые результаты в демонстрационной или тестовой среде, рассмотрите возможность симуляции более длительных или более интенсивных с точки зрения ЦПУ операций. В качестве альтернативы, в образовательных целях, вы можете ввести искусственные задержки (например, используя функции sleep), чтобы сделать прошедшее время более заметным. Однако помните, что введение задержек не является реалистичным представлением фактического времени обработки, но может помочь в иллюстрации поведения таймера на воспринимаемых продолжительностях.

Самостоятельная задача

Разработка класса Калькулятор для базовых арифметических операций
Описание: Создайте класс Calculator, который может выполнять базовые арифметические операции (+, -, *, /) с двумя операндами.
Переменные-члены: не требуются.
Функции-члены: Методы для каждой операции (+, -, *, /)