ЛАБОРАТОРНАЯ РАБОТА №3

Курс «Объектно-ориентированное программирование»

Тема: Наследование и полиморфизм. Абстрактные классы и интерфейсы. RTTI.

Цель: Научиться реализовывать на C++ наследование классов, программировать абстрактные классы и интерфейсы, виртуальные методы, а также динамически определять тип объекта во время выполнения программы.

Ход работы

Вариант 5

Задание 1.

1. Написать в ООП-стиле код программы, позволяющей работать с арифметическими выражениями разного вида, оперирующими вещественными числами: вычислять результат выражения, выводить запись выражения на консоль и в файл лога. Например, для вычисления выражений вида (10+4+2+3+7+1) и (1+2.5) будет использоваться класс `Summator`, выражений вида (2*3*7*1) – класс `Multiplier`, и т.д.
   В коде необходимо отразить следующее:
   - Создать интерфейс `ILoggable` с 2 методами (функционал логирования):

     Запись лога всего выражения на консоль:
     ```
     void logToScreen()
     ```
     Добавление записи лога всего выражения в файл лога:
     ```
     void logToFile(const std::string& filename).
     ```
   - Создать абстрактный класс `ExpressionEvaluator`, реализующий интерфейс `ILoggable` и предоставляющий чисто виртуальный метод `double calculate()` для вычисления результата произвольного выражения. Количество операндов должно храниться в отдельном члене класса. Сами операнды `x1,x2,x3` и т.д. должны храниться в члене данного класса – массиве, в куче (динамической памяти).
   - Класс `ExpressionEvaluator` должен предоставлять два конструктора и виртуальный деструктор. В конструкторе без параметров выделять память под 20 операндов и инициализировать их нулями, в конструкторе с параметром `n` – выделять память под `n` элементов и инициализировать нулями. Также необходимо реализовать 2 метода, позволяющие присвоить операндам конкретные значения:

     Присвоить значение *value* одному операнду на позиции *pos*:
     ```
     void setOperand(size_t pos, double value)
     ```
     Заполнить сразу группу из *n* операндов массивом значений *ops*:
     ```
     void setOperands(double ops[], size_t n)
     ```
   - В деструкторе должна освобождаться память, выделенная в конструкторе.
   - Создать два подкласса класса `ExpressionEvaluator`, работающих со стандартными выражениями, в соответствии с вариантом, из четырех возможных:

     `Summator` – сумма всех операндов (`x1 + x2 + x3 + x4 + ...`)
     `Subtractor` – разность всех операндов (`x1 - x2 - x3 - x4 - ...`)
     `Multiplier` – произведение всех операндов (`x1 * x2 * x3 * x4 * ...`)
     `Divisor` – частное всех операндов (`x1/x2/x3/x4/...`), но если хоть один операнд равен 0, то результату выражения присвоить также 0.

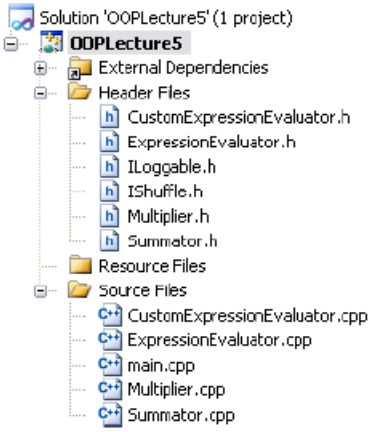
- Создать подкласс `CustomExpressionEvaluator`, работающий со специфическими выражениями, вид которых приведен в варианте.
- Подклассы `ExpressionEvaluator`, для которых порядок следования операндов важен, должны также реализовывать интерфейс `IShuffle`. Данный интерфейс объявляет 2 перегруженных метода (функционал перемешивания операндов):

  Произвольно перемешать операнды:
  ```
  void shuffle()
  ```
  Перемешать операнды, находящиеся на позициях i и j:
  ```
  void shuffle(size_t i, size_t j)
  ```

В функции `main()` необходимо продемонстрировать работу созданных классов:

- Создать массив из трех указателей на класс обработки арифметических выражений.
- В соответствии с вариантом, создать в куче три объекта конкретных подклассов обработки арифметических выражений и установить на них указатели; присвоить их операндам значения двумя способами (поэлементным и групповым).
- Продемонстрировать полиморфизм: организовать проход в цикле по указателям и вывести лог выражения на консоль и в файл (в консоли отобразить еще и сам результат выражения).
- Организовать цикл по указателям, в теле которого средствами C++ проверить, реализует ли текущий объект интерфейс `IShuffle`. Если да, то вызвать один из методов `shuffle()` этого объекта и отобразить на экране запись выражения после перемешивания операндов, а также вычислить и отобразить результат нового выражения.

Ниже приведена желательная структура проекта (например, для случая, когда нужно использовать классы `Summator` и `Multiplier`):



Примечание. Отрицательные операнды при выводе записи выражения на экран и в файл должны автоматически браться в скобки (подробнее см. приложение В).

Вариант 5

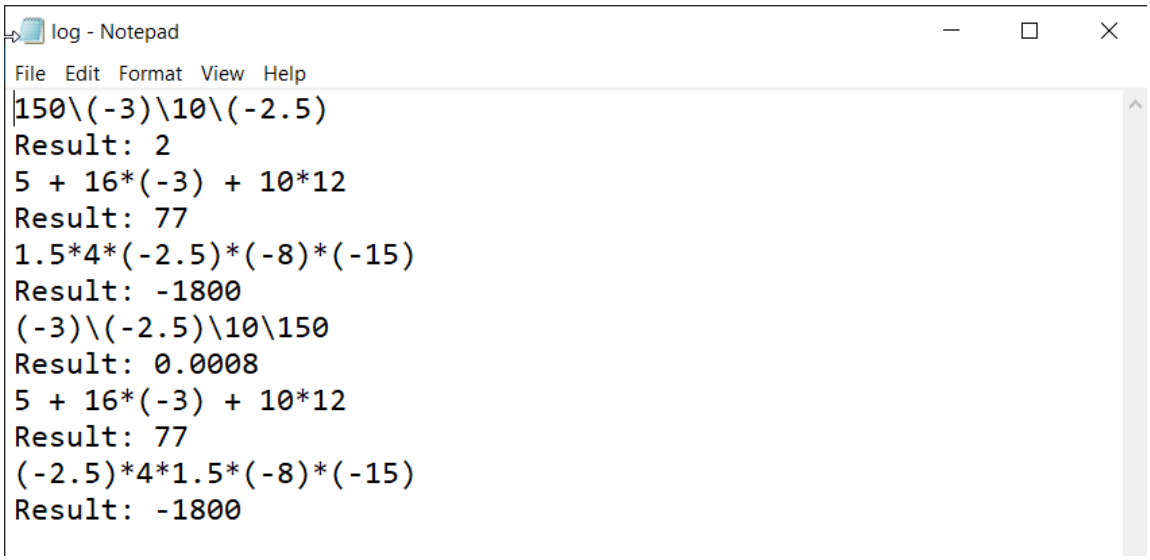
Вид выражения `CustomExpression`: `result = x1 + x2 * x3 + x4 * x5 + ...`

Порядок создания и инициализации объектов подклассов:
`Divisor`: 4 операнда, присвоить поэлементно `150, -3, 10, -2.5`.
`CustomExpressionEvaluator`: 5 операндов, присвоить группой `5, 16, -3, 10, 12`.
`Multiplier`: 5 операндов, присвоить группой `1.5, 4, -2.5 -8, -15`.

Метод `shuffle()` – отсортировать все операнды в порядке возрастания.
Метод `shuffle(size_t i, size_t j)` – если хотя бы один из i-ого и j-ого операндов имеет дробную часть, поменять их местами, иначе – не менять.

Формат вывода:
```
5 + 16*(-3) + 10*12 < Total 5 >
< Result 77 >
```

Результат работы программы:



Код:

Main.cpp

```
#include "Divisor.h"
#include "Multiplier.h"
#include "CustomExpressionEvaluator.h"
#include <iostream>

int main() {
    setlocale(LC_ALL, "Russian");

    // Массив указателей
    ExpressionEvaluator* evaluators[3];
```

```
    // Divisor
    Divisor* div = new Divisor(4);

    // Поэлементно
    div->setOperand(0, 150);
    div->setOperand(1, -3);
    div->setOperand(2, 10);
    div->setOperand(3, -2.5);
    evaluators[0] = div;

    // CustomExpressionEvaluator
    CustomExpressionEvaluator* custom = new
CustomExpressionEvaluator(5);

    // Групповое присваивание
    double customOps[] = { 5, 16, -3, 10, 12 };
    custom->setOperands(customOps, 5);
    evaluators[1] = custom;

    // Multiplier
    Multiplier* mult = new Multiplier(5);
    double multOps[] = { 1.5, 4, -2.5, -8, -15 };
    mult->setOperands(multOps, 5);
    evaluators[2] = mult;

    // Логирование
    std::cout << "Логирование\n----------------------" << std::endl;
    for (int i = 0; i < 3; i++) {
        evaluators[i]->logToScreen();
        evaluators[i]->logToFile("log.txt");
        std::cout << std::endl;
    }

    std::cout << "Перемешивание\n-----------------------------------"
<< std::endl;
    for (int i = 0; i < 3; i++) {
    IShuffle* shuffleable = dynamic_cast<IShuffle*>(evaluators[i]);

    if (shuffleable) {

        // заранее всё перемешиваем
        if (i == 0) {
            shuffleable->shuffle();  // для деления
        }
        else if (i == 1) {
            shuffleable->shuffle(0, 1);  // для сложения произведений
        }
        else if (i == 2) {
            shuffleable->shuffle(0, 2);  // для умноженя
        }

        // Вывод
        std::cout << evaluators[i]->getExpression()
                  << " < Total " << evaluators[i]->getNumOperands() << "
>" << std::endl;
```

```
        std::cout << "< Result " << evaluators[i]->calculate() << " >"
<< std::endl;
        std::cout << std::endl;
        evaluators[i]->logToFile("log.txt");
    }
}

    delete div;
    delete custom;
    delete mult;

    return 0;
     }
```

CustomExpressionEvaluator.h

```
#pragma once
#include "ExpressionEvaluator.h"
#include "IShuffle.h"

#include "additionalFunctions.h"

class CustomExpressionEvaluator : public ExpressionEvaluator, public
IShuffle {
public:
    CustomExpressionEvaluator(size_t n);
    double calculate() override;
    std::string getExpression() override;

    void shuffle() override;
    void shuffle(size_t i, size_t j) override;
     };
```

CustomExpressionEvaluator.cpp

```
#include "CustomExpressionEvaluator.h"
#include <iostream>
#include <algorithm>
#include <cmath>

CustomExpressionEvaluator::CustomExpressionEvaluator(size_t n) :
ExpressionEvaluator(n) {}

double CustomExpressionEvaluator::calculate() {
    double result = operands[0];

    for (int i = 1; i < numOperands; i++) {
        if (i % 2 == 1) {  // Если нечетный индекс
            if (i + 1 < numOperands) {
                result += operands[i] * operands[i + 1];
            }
            else {
                result += operands[i];
            }
```

```
        }
    }

    return result;
}

std::string CustomExpressionEvaluator::getExpression() {
    std::string text;

    // Первое число
    std::string firstN = doubleToString(operands[0]);
    // Проверяем, отрицательное ли число
    if (operands[0] < 0) {
        text = "(" + firstN + ")";
    }
    else {
        text = firstN;
    }

    // Остальные числа
    for (size_t i = 1; i < numOperands; i += 2) {
        text += " + ";

        std::string nStr = doubleToString(operands[i]);
        if (operands[i] < 0) {
            text += "(" + nStr + ")";
        }
        else {
            text += nStr;
        }

        if (i + 1 < numOperands) {
            text += "*";

            std::string nextNStr = doubleToString(operands[i + 1]);
            if (operands[i + 1] < 0) {
                text += "(" + nextNStr + ")";
            }
            else {
                text += nextNStr;
            }
        }
    }

    return text;
}
void CustomExpressionEvaluator::shuffle() {
    // Пузырьковая сортировка
    for (int i = 0; i < numOperands; i++) {
        for (int j = 0; j < numOperands - 1; j++) {
            if (operands[j] > operands[j + 1]) {
                double temp = operands[j];
                operands[j] = operands[j + 1];
                operands[j + 1] = temp;
            }
        }
```

```
    }
}

void CustomExpressionEvaluator::shuffle(size_t i, size_t j) {
    // Проверяем, находятся ли i и j в рамках массива операндов
    if (i >= 0 && i < numOperands && j >= 0 && j < numOperands) {

        // если есть дробная часть, то меняем местами
        if ((operands[i] != (int)operands[i]) || (operands[j] !=
(int)operands[j])) {
            double temp = operands[i];
            operands[i] = operands[j];
            operands[j] = temp;
        }
    }
     }
```

Divisor.h

```
#pragma once
#include "ExpressionEvaluator.h"
#include "IShuffle.h"

#include "additionalFunctions.h"

class Divisor : public ExpressionEvaluator, public IShuffle {
public:
    Divisor(size_t n);
    double calculate() override;
    std::string getExpression() override;

    void shuffle() override;
    void shuffle(size_t i, size_t j) override;
     };
```

Divisor.cpp

```
#include "Divisor.h"
#include <iostream>
#include <algorithm>

Divisor::Divisor(size_t n) : ExpressionEvaluator(n) {}

double Divisor::calculate() {
    double result = operands[0];

    // Все остальные числа
    for (size_t i = 1; i < numOperands; i++) {
        // проверка деления на 0
        if (operands[i] == 0) {
            return 0;
        }
        result /= operands[i];
```

```
    }
    return result;
}

std::string Divisor::getExpression() {
    std::string text;
    for (size_t i = 0; i < numOperands; i++) {

        std::string nStr = doubleToString(operands[i]);

        // Отрицательные числа берём в скобки
        if (operands[i] < 0) {
            text += "(" + nStr + ")";
        }
        else {
            text += nStr;
        }

        // Ставим * перед всеми числами, кроме последнего
        if (i < numOperands - 1) {
            text += "\\";
        }
    }
    return text;
}

void Divisor::shuffle() {
    // Пузырьковая сортировка
    for (int i = 0; i < numOperands; i++) {
        for (int j = 0; j < numOperands - 1; j++) {
            if (operands[j] > operands[j + 1]) {
                double temp = operands[j];
                operands[j] = operands[j + 1];
                operands[j + 1] = temp;
            }
        }
    }
}

void Divisor::shuffle(size_t i, size_t j) {
    // Проверяем, находятся ли i и j в рамках массива операндов
    if (i >= 0 && i < numOperands && j >= 0 && j < numOperands) {

        // если есть дробная часть, то меняем местами
        if ((operands[i] != (int)operands[i]) || (operands[j] !=
(int)operands[j])) {
            double temp = operands[i];
            operands[i] = operands[j];
            operands[j] = temp;
        }
    }
     }
```

ExpressionEvaluator.h

```h
#pragma once
#include "ILoggable.h"
#include <string>
#include <fstream>

class ExpressionEvaluator : public ILoggable {
protected:
    size_t numOperands; // Количество операндов
    double* operands;  // Массив операторов

public:
    ExpressionEvaluator();
    ExpressionEvaluator(size_t n);
    virtual ~ExpressionEvaluator();

    // Присвоить значение одному операнду на позиции pos
    void setOperand(size_t pos, double value);

    // Присвоить значения группе операндов
    void setOperands(double ops[], size_t n);

    virtual double calculate() = 0;
    void logToScreen() override;
    void logToFile(const std::string& filename) override;

    virtual std::string getExpression() = 0;

    size_t getNumOperands() const { return numOperands; }
     };
```

ExpressionEvaluator.cpp

```cpp
#include "ExpressionEvaluator.h"
#include <iostream>
#include <fstream>

// По умолчанию количество операндов ставим 20
ExpressionEvaluator::ExpressionEvaluator() {
    numOperands = 20;
    operands = new double[20];
    for (size_t i = 0; i < 20; i++) {
        operands[i] = 0;
    }
}

ExpressionEvaluator::ExpressionEvaluator(size_t n) {
    numOperands = n;
    operands = new double[n];
    for (size_t i = 0; i < n; i++) {
        operands[i] = 0;
    }
}
```

```cpp
ExpressionEvaluator::~ExpressionEvaluator() {
    delete[] operands;
}


void ExpressionEvaluator::setOperand(size_t pos, double value) {
    operands[pos] = value;
}

void ExpressionEvaluator::setOperands(double ops[], size_t n) {
    for (int i = 0; i < n; i++) {
        operands[i] = ops[i];
    }
}

void ExpressionEvaluator::logToScreen() {
    std::cout << getExpression() << std::endl;
    // Вывод в консоль на английском, потому что так по заданию
    std::cout << "Result: " << calculate() << std::endl;
}

void ExpressionEvaluator::logToFile(const std::string& filename) {
    std::ofstream file(filename, std::ios::app);
    if (file.is_open()) {
        file << getExpression() << std::endl;
        file << "Result: " << calculate() << std::endl;
        file.close();
    }
     }
```

Multiplier.h

```cpp
#pragma once
#include "ExpressionEvaluator.h"
#include "IShuffle.h"

#include "additionalFunctions.h"

class Multiplier : public ExpressionEvaluator, public IShuffle {
public:
    Multiplier(size_t n);
    double calculate() override;
    std::string getExpression() override;

    void shuffle() override;
    void shuffle(size_t i, size_t j) override;
     };
```

Multiplier.cpp

```cpp
#include "Multiplier.h"
#include <iostream>
#include <algorithm>
```

```cpp
#include <cmath>

Multiplier::Multiplier(size_t n) : ExpressionEvaluator(n) {}

double Multiplier::calculate() {
    double result = 1.0;
    for (size_t i = 0; i < numOperands; i++) {
        result *= operands[i];
    }
    return result;
}

std::string Multiplier::getExpression() {
    std::string text;
    for (size_t i = 0; i < numOperands; i++) {

        std::string nStr = doubleToString(operands[i]);

        // Отрицательные числа берём в скобки
        if (operands[i] < 0) {
            text += "(" + nStr + ")";
        }
        else {
            text += nStr;
        }

        // Ставим * перед всеми числами, кроме последнего
        if (i < numOperands - 1) {
            text += "*";
        }
    }
    return text;
}

void Multiplier::shuffle() {
    // Пузырьковая сортировка
    for (int i = 0; i < numOperands; i++) {
        for (int j = 0; j < numOperands - 1; j++) {
            if (operands[j] > operands[j + 1]) {
                double temp = operands[j];
                operands[j] = operands[j + 1];
                operands[j + 1] = temp;
            }
        }
    }
}

void Multiplier::shuffle(size_t i, size_t j) {
    // Проверяем, находятся ли i и j в рамках массива операндов
    if (i >= 0 && i < numOperands && j >= 0 && j < numOperands) {

        // если есть дробная часть, то меняем местами
        if ((operands[i] != (int)operands[i]) || (operands[j] !=
(int)operands[j])) {
            double temp = operands[i];
            operands[i] = operands[j];
```

```
            operands[j] = temp;
        }
    }
    }
```

ILoggable.h

```
#pragma once
#include <string>

// Нельзя создать этот класс напрямую, только через наследников
class ILoggable {
public:
    virtual void logToScreen() = 0;
    virtual void logToFile(const std::string& filename) = 0;
    virtual ~ILoggable() {}
    };
```

IShuffle.h

```
#pragma once

class IShuffle {
public:
    virtual void shuffle() = 0;
    virtual void shuffle(size_t i, size_t j) = 0;
    virtual ~IShuffle() {}
    };
```

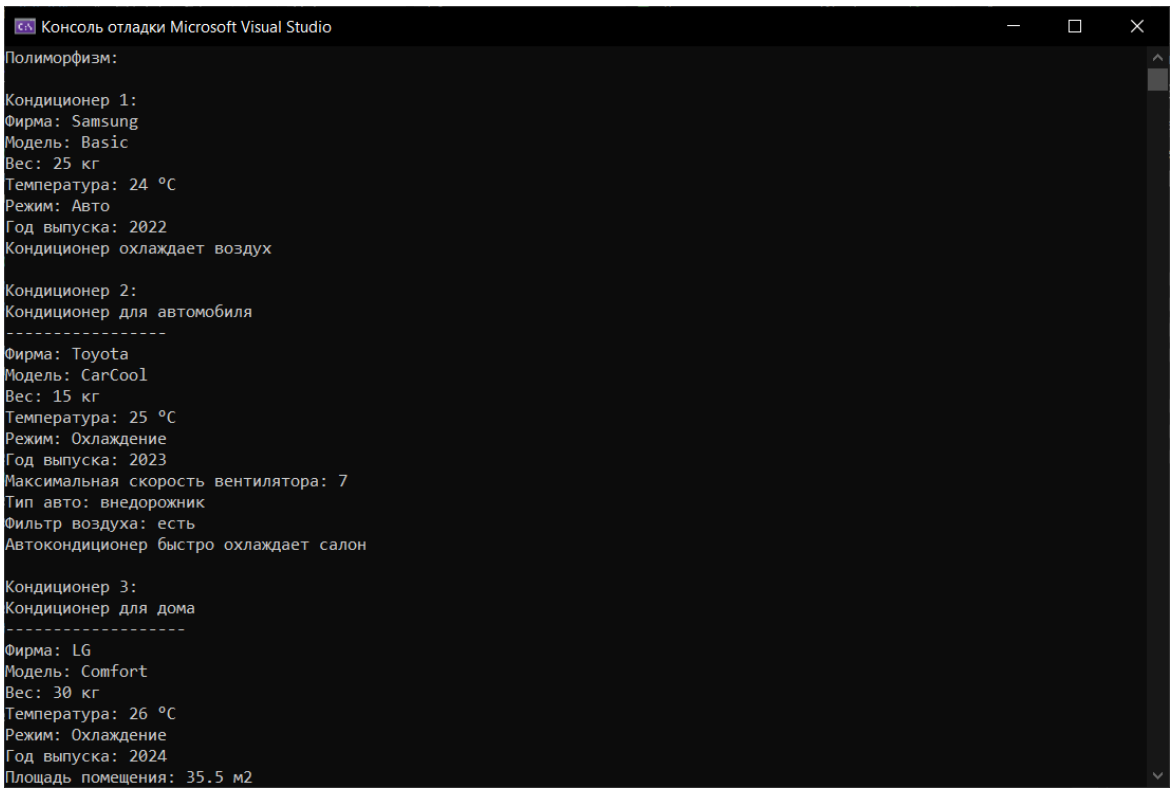
Задание 2.

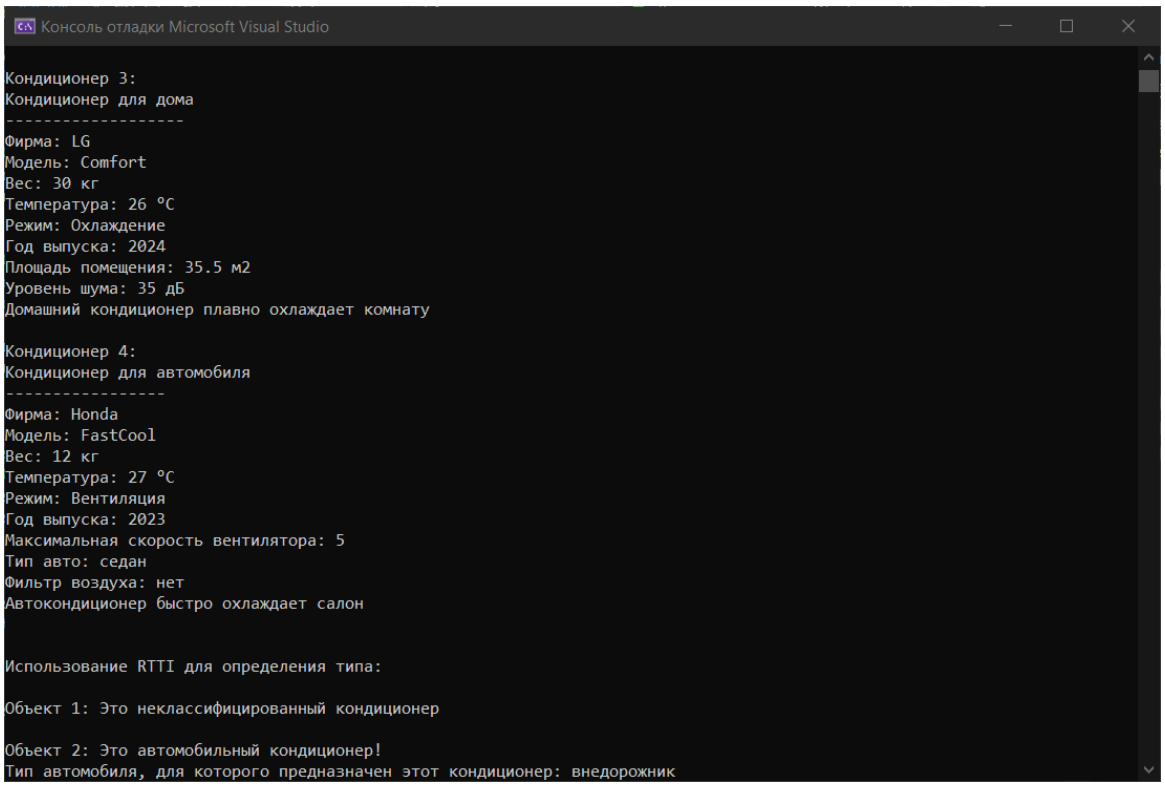
2. Дополнить код задания 3 лабораторной работы № 2, написав еще два класса по предметной области, в соответствии с вариантом. Продумать и корректно реализовать схему наследования классов. В главной функции продемонстрировать применение интерфейса, полиморфизм и RTTI на примере 3-4 объектов классов, по аналогии с заданием 1.
   Примечание. Данное задание творческое. Вы можете свободно вводить различные свойства и метода классов, не указанные явно в задании.

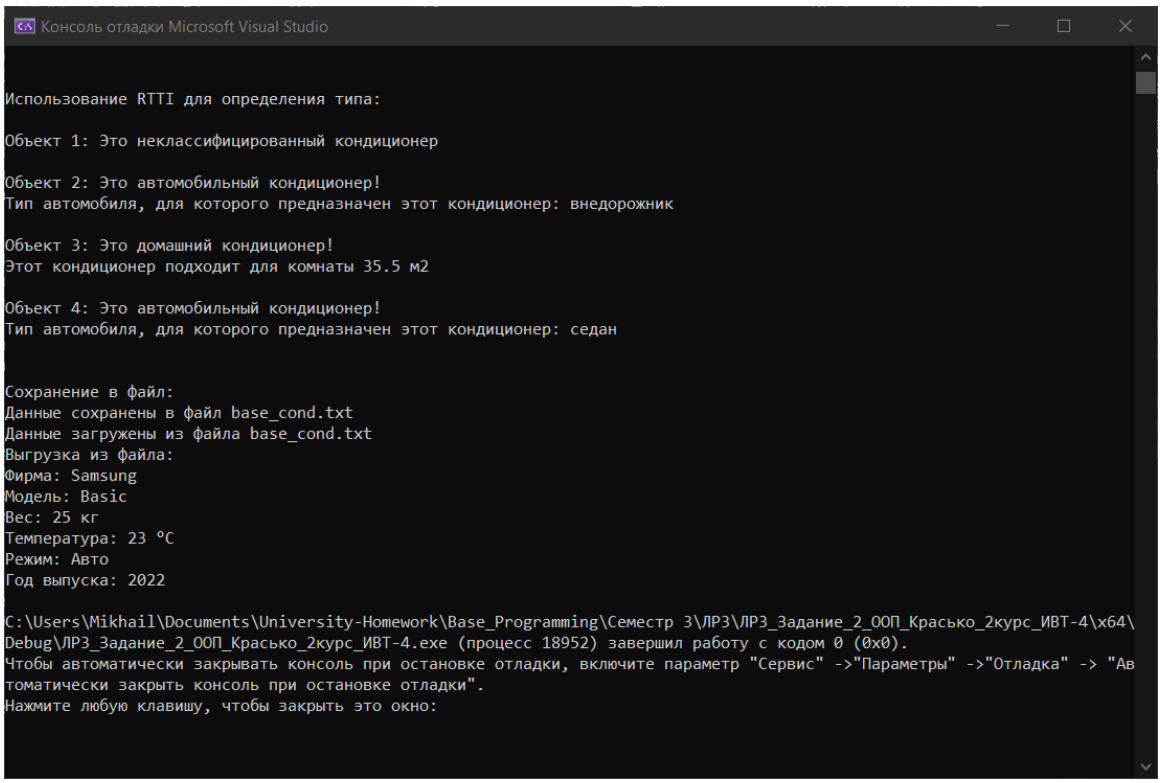
Тема: кондиционер.

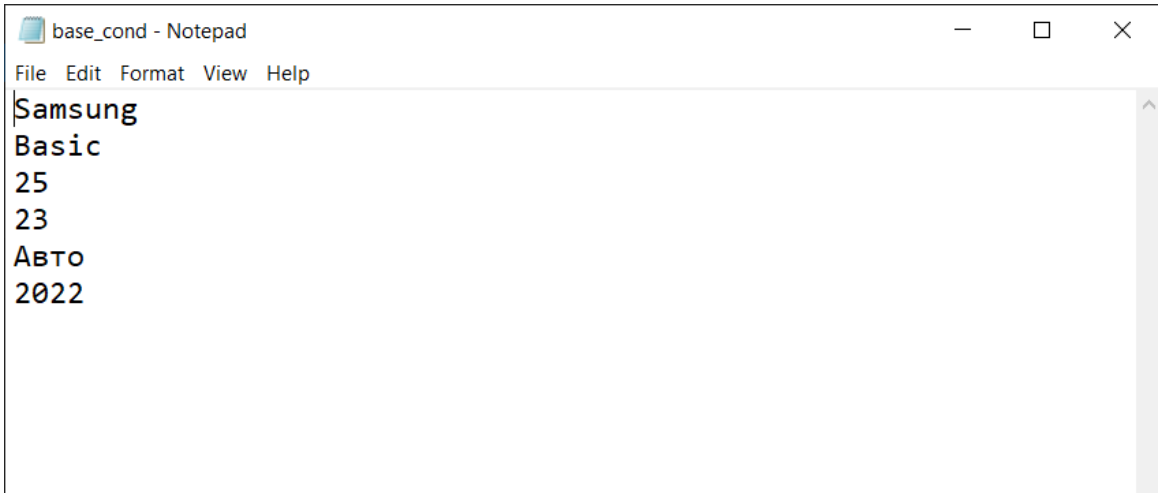
Модификация: добавление классов автомобильных и домашних кондиционеров.

Вывод:


Консоль отладки Microsoft Visual Studio
Полиморфизм:
Кондиционер 1:
Фирма: Samsung
Модель: Basic
Вес: 25 кг
Температура: 24 °C
Режим: Авто
Год выпуска: 2022
Кондиционер охлаждает воздух
Кондиционер 2:
Кондиционер для автомобиля
Фирма: Toyota
Модель: CarCool
Вес: 15 кг
Температура: 25 °C
Режим: Охлаждение
Год выпуска: 2023
Максимальная скорость вентилятора: 7
Тип авто: внедорожник
Фильтр воздуха: есть
Автокондиционер быстро охлаждает салон
Кондиционер 3:
Кондиционер для дома
Фирма: LG
Модель: Comfort
Вес: 30 кг
Температура: 26 °C
Режим: Охлаждение
Год выпуска: 2024
Площадь помещения: 35.5 м2


Консоль отладки Microsoft Visual Studio
Кондиционер 3:
Кондиционер для дома
Фирма: LG
Модель: Comfort
Вес: 30 кг
Температура: 26 °C
Режим: Охлаждение
Год выпуска: 2024
Площадь помещения: 35.5 м2
Уровень шума: 35 дБ
Домашний кондиционер плавно охлаждает комнату
Кондиционер 4:
Кондиционер для автомобиля
Фирма: Honda
Модель: FastCool
Вес: 12 кг
Температура: 27 °C
Режим: Вентиляция
Год выпуска: 2023
Максимальная скорость вентилятора: 5
Тип авто: седан
Фильтр воздуха: нет
Автокондиционер быстро охлаждает салон
Использование RTTI для определения типа:
Объект 1: Это неклассифицированный кондиционер
Объект 2: Это автомобильный кондиционер!
Тип автомобиля, для которого предназначен этот кондиционер: внедорожник





Код:

Conditioner.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

class Conditioner {
protected:
    std::string brand;
    std::string model;
    double weight;
    double temperature;
    std::string mode;
```

```
    int year;

public:
    Conditioner();
    Conditioner(std::string brandInput, std::string modelInput, double
weightInput,
        double temperatureInput, std::string modeInput, int yearInput);

    virtual ~Conditioner() {}

    // Геттеры
    std::string getBrand();
    std::string getModel();
    double getWeight();
    double getTemperature();
    std::string getMode();
    int getYear();

    // Сеттеры
    void setBrand(std::string brandInput);
    void setModel(std::string modelInput);
    void setWeight(double weightInput);
    void setTemperature(double temperatureInput);
    void setMode(std::string modeInput);
    void setYear(int yearInput);

    virtual void printInfo();
    virtual void cool();  // Охлаждение
    virtual void saveToFile(std::string filename = "conditioner.txt");
    virtual void loadFromFile(std::string filename = "conditioner.txt");
     };
```

Conditioner.cpp

```
#include "conditioner.h"

Conditioner::Conditioner() {
    brand = "";
    model = "";
    weight = 0;
    temperature = 0;
    mode = "";
    year = 0;
}

Conditioner::Conditioner(std::string brandInput, std::string modelInput,
double weightInput,
    double temperatureInput, std::string modeInput, int yearInput) {
    brand = brandInput;
    model = modelInput;
    weight = weightInput;
    temperature = temperatureInput;
    mode = modeInput;
    year = yearInput;
```

```
}

std::string Conditioner::getBrand() { return brand; }
std::string Conditioner::getModel() { return model; }
double Conditioner::getWeight() { return weight; }
double Conditioner::getTemperature() { return temperature; }
std::string Conditioner::getMode() { return mode; }
int Conditioner::getYear() { return year; }

void Conditioner::setBrand(std::string brandInput) { brand = brandInput;
}
void Conditioner::setModel(std::string modelInput) { model = modelInput;
}
void Conditioner::setWeight(double weightInput) { weight = weightInput;
}
void Conditioner::setTemperature(double temperatureInput) { temperature
= temperatureInput; }
void Conditioner::setMode(std::string modeInput) { mode = modeInput; }
void Conditioner::setYear(int yearInput) { year = yearInput; }

void Conditioner::printInfo() {
    std::cout << "Фирма: " << brand << std::endl;
    std::cout << "Модель: " << model << std::endl;
    std::cout << "Вес: " << weight << " кг" << std::endl;
    std::cout << "Температура: " << temperature << " °C" << std::endl;
    std::cout << "Режим: " << mode << std::endl;
    std::cout << "Год выпуска: " << year << std::endl;
}

void Conditioner::cool() {
    std::cout << "Кондиционер охлаждает воздух" << std::endl;
    temperature -= 1;
}

void Conditioner::saveToFile(std::string filename) {
    std::ofstream file(filename);
    if (file.is_open()) {
        file << brand << std::endl;
        file << model << std::endl;
        file << weight << std::endl;
        file << temperature << std::endl;
        file << mode << std::endl;
        file << year << std::endl;
        file.close();
        std::cout << "Данные сохранены в файл " << filename <<
std::endl;
    }
}

void Conditioner::loadFromFile(std::string filename) {
    std::ifstream file(filename);
    if (file.is_open()) {
        std::getline(file, brand);
        std::getline(file, model);
        file >> weight;
        file >> temperature;
```

```
        file.ignore();
        std::getline(file, mode);
        file >> year;
        file.close();
        std::cout << "Данные загружены из файла " << filename <<
std::endl;
    }
     }
```

CarConditioner.h

```
#pragma once

#include "conditioner.h"

class CarConditioner : public Conditioner {
private:
    int maxFanSpeed;
    std::string carType;
    bool hasAirFilter;

public:
    CarConditioner();
    CarConditioner(std::string brand, std::string model, double weight,
        double temp, std::string mode, int year,
        int maxSpeed, std::string type, bool filter);

    // Переопределение виртуальных методов
    void printInfo() override;
    void cool() override;

    void setFanSpeed(int speed);
    void showCarType();
     };
```

CarConditioner.cpp

```
#include "carconditioner.h"

CarConditioner::CarConditioner() : Conditioner() {

    // Значения по умолчанию
    maxFanSpeed = 5;
    carType = "легковой";
    hasAirFilter = true;
}

CarConditioner::CarConditioner(std::string brand, std::string model,
double weight,
    double temp, std::string mode, int year,
    int maxSpeed, std::string type, bool filter)
    : Conditioner(brand, model, weight, temp, mode, year) {
```

```
    maxFanSpeed = maxSpeed;
    carType = type;
    hasAirFilter = filter;
}

void CarConditioner::printInfo() {
    std::cout << "Кондиционер для автомобиля\n-----------------" <<
std::endl;
    Conditioner::printInfo();
    std::cout << "Максимальная скорость вентилятора: " << maxFanSpeed <<
std::endl;
    std::cout << "Тип авто: " << carType << std::endl;
    if (hasAirFilter) {
        std::cout << "Фильтр воздуха: есть" << std::endl;
    }
    else {
        std::cout << "Фильтр воздуха: нет" << std::endl;
    }
}

void CarConditioner::cool() {
    std::cout << "Автокондиционер быстро охлаждает салон" << std::endl;
    temperature -= 3;  // в машине кондиционер охлаждает быстрее
}

void CarConditioner::setFanSpeed(int speed) {
    if (speed <= maxFanSpeed) {
        std::cout << "Скорость вентилятора установлена на " << speed <<
std::endl;
    }
    else {
        std::cout << "Слишком большая скорость! Максимум " <<
maxFanSpeed << std::endl;
    }
}

void CarConditioner::showCarType() {
    std::cout << "Тип автомобиля, для которого предназначен этот
кондиционер: " << carType << std::endl;
     }
```

homeConditioner.h

```
#pragma once

#include "conditioner.h"

class HomeConditioner : public Conditioner {
private:
    double roomArea;
    int noiseLevel;

public:
    HomeConditioner();
```

```
    HomeConditioner(std::string brand, std::string model, double weight,
        double temp, std::string mode, int year,
        double area, int noise);

    void printInfo() override;
    void cool() override;

    void showSuitableArea();
     };
```

homeConditioner.cpp

```
#include "homeconditioner.h"

HomeConditioner::HomeConditioner() : Conditioner() {
    roomArea = 20.0;
    noiseLevel = 40;
}

HomeConditioner::HomeConditioner(std::string brand, std::string model,
double weight,
    double temp, std::string mode, int year,
    double area, int noise)
    : Conditioner(brand, model, weight, temp, mode, year) {
    roomArea = area;
    noiseLevel = noise;
}
void HomeConditioner::printInfo() {
    std::cout << "Кондиционер для дома\n-------------------" <<
std::endl;
    Conditioner::printInfo();
    std::cout << "Площадь помещения: " << roomArea << " м2" <<
std::endl;
    std::cout << "Уровень шума: " << noiseLevel << " дБ" << std::endl;
}
void HomeConditioner::cool() {
    std::cout << "Домашний кондиционер плавно охлаждает комнату" <<
std::endl;
    temperature -= 0.5;  // дома охлаждает медленнее
}
void HomeConditioner::showSuitableArea() {
    std::cout << "Этот кондиционер подходит для комнаты " << roomArea <<
" м2" << std::endl;
     }
```

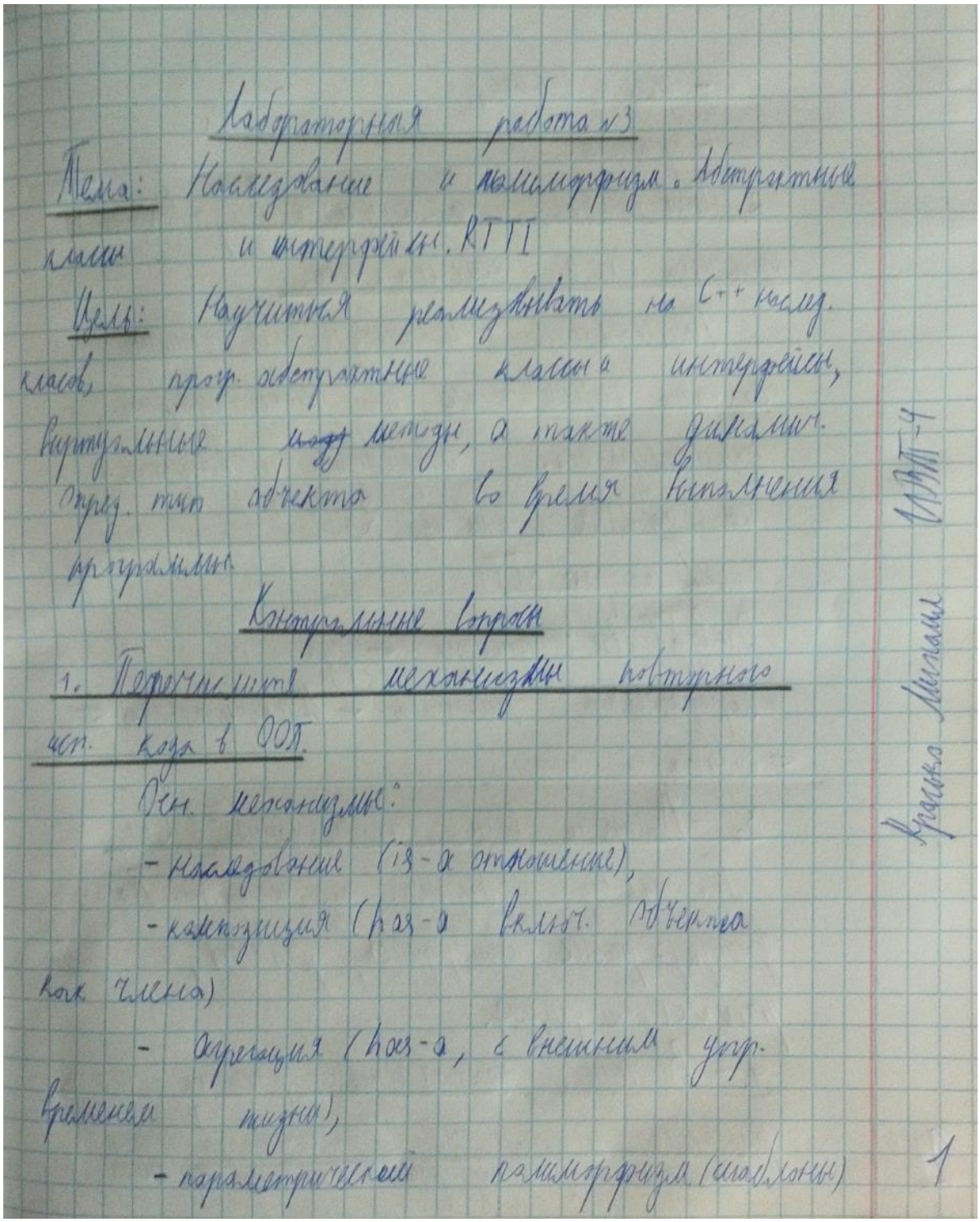
main.cpp

```
#include <iostream>
#include <vector>
#include <typeinfo>  // для RTTI

#include "conditioner.h"
#include "carconditioner.h"
#include "homeconditioner.h"
int main() {
```

```
    setlocale(LC_ALL, "Russian");
    // Создаем массив указателей на базовый класс
    std::vector<Conditioner*> conditioners;
    conditioners.push_back(new Conditioner("Samsung", "Basic", 25, 24,
"Авто", 2022));
    conditioners.push_back(new CarConditioner("Toyota", "CarCool", 15,
25, "Охлаждение", 2023, 7,
        "внедорожник", true));
    conditioners.push_back(new HomeConditioner("LG", "Comfort", 30, 26,
"Охлаждение",
        2024, 35.5, 35));
    conditioners.push_back(new CarConditioner("Honda", "FastCool", 12,
27, "Вентиляция",
        2023, 5, "седан", false));
    std::cout << "Полиморфизм:" << std::endl;
    for (int i = 0; i < conditioners.size(); i++) {
        std::cout << "\nКондиционер " << i + 1 << ":" << std::endl;
        conditioners[i]->printInfo();  // полиморфный вызов
        conditioners[i]->cool();       // полиморфный вызов
    }
    std::cout << "\n\nИспользование RTTI для определения типа:" <<
std::endl;
    for (int i = 0; i < conditioners.size(); i++) {
        std::cout << "\nОбъект " << i + 1 << ": ";

        if (dynamic_cast<CarConditioner*>(conditioners[i])) {
            std::cout << "Это автомобильный кондиционер!" << std::endl;

            // Вызываем методы, которые есть только у автомобильных
кондиционеров
            CarConditioner* carCond =
dynamic_cast<CarConditioner*>(conditioners[i]);
            carCond->showCarType();
        }
        else if (dynamic_cast<HomeConditioner*>(conditioners[i])) {
            std::cout << "Это домашний кондиционер!" << std::endl;

            HomeConditioner* homeCond =
dynamic_cast<HomeConditioner*>(conditioners[i]);
            homeCond->showSuitableArea();
        }
        else {
            std::cout << "Это неклассифицированный кондиционер" <<
std::endl;
        }
    }

    // Демонстрация работы с файлами
    std::cout << "\n\nСохранение в файл:" << std::endl;
    conditioners[0]->saveToFile("base_cond.txt");

    Conditioner loadedCond;
    loadedCond.loadFromFile("base_cond.txt");
    std::cout << "Выгрузка из файла:" << std::endl;
    loadedCond.printInfo();
```

```
// Очистка памяти
for (int i = 0; i < conditioners.size(); i++) {
    delete conditioners[i];
}

return 0;
 }
```

Контрольные вопросы



Лабораторная работа №3

Тема: Наследование и полиморфизм. Абстрактные классы и интерфейсы. RTTI

Цель: Научиться реализовывать на C++ наслед. класс, прогр. абстрактные классы и интерфейсы, виртуальные ~~функ~~ методы, а также динамич. опред. типа объекта во время выполнения программы.

Контрольные вопросы

1. Перечислите механизмы повторного исп. кода в ООП.

Осн. механизмы:

- наследование (is-a отношение),
- композиция (has-a включ. объекта как члена)
- агрегация (has-a, с внешним упр. временем жизни),
- параметрический полиморфизм (шаблоны)

Кравцов Михаил ИВТ-4

1

- делегирование,
- примеси.

2. Что такое наследование? Приведите 3 примера схем иерархий классов.

Наследование - механизм, позволяющий создать новый класс на основе существующего, заимствуя его атрибуты и поведение, и добавляя или изменяя их.

Примеры:

Однозначное: фигура ← круг

Иерархическое: Животные ← Млекопитающие ← ← Собака

3. Типы и виды наследования в C++

По кол-ву базовых: од. и множ. По доступу: public (интерфейс), protected, private (реализация).

4. В чем заключается разница между ранними и поздними связываниями?

Ранние: компилятор опр. ф-цию по типу переменной. Поздние:

во время вызывания по реальному типу объекта через виртуальные методы.

5. Что такое полиморфизм? Виртуальные методы.

Полиморфизм - вызывание разного поведения через общий интерфейс.

Виртуальные методы (virtual) обеспечивают позднее связывание.

6. Что такое абстрактный класс? Чисто виртуальные методы

Абстрактный класс нельзя инстанцировать. Содержит ≥ 1 чисто виртуальный метод

7. Что из себя представляет класс-интерфейс? Для чего он исп.?

Класс-интерфейс - абстрактный класс только с чисто виртуальными методами. Определяет контракт для реализации.

8. Порядок создания и удаления подклассов. Зачем нужен виртуальный деструктор?

Красота Милана ОВВТ-4

3

Создание: база → члены предка → конструктор предка.

Удаление: деструктор производного → члены → база.

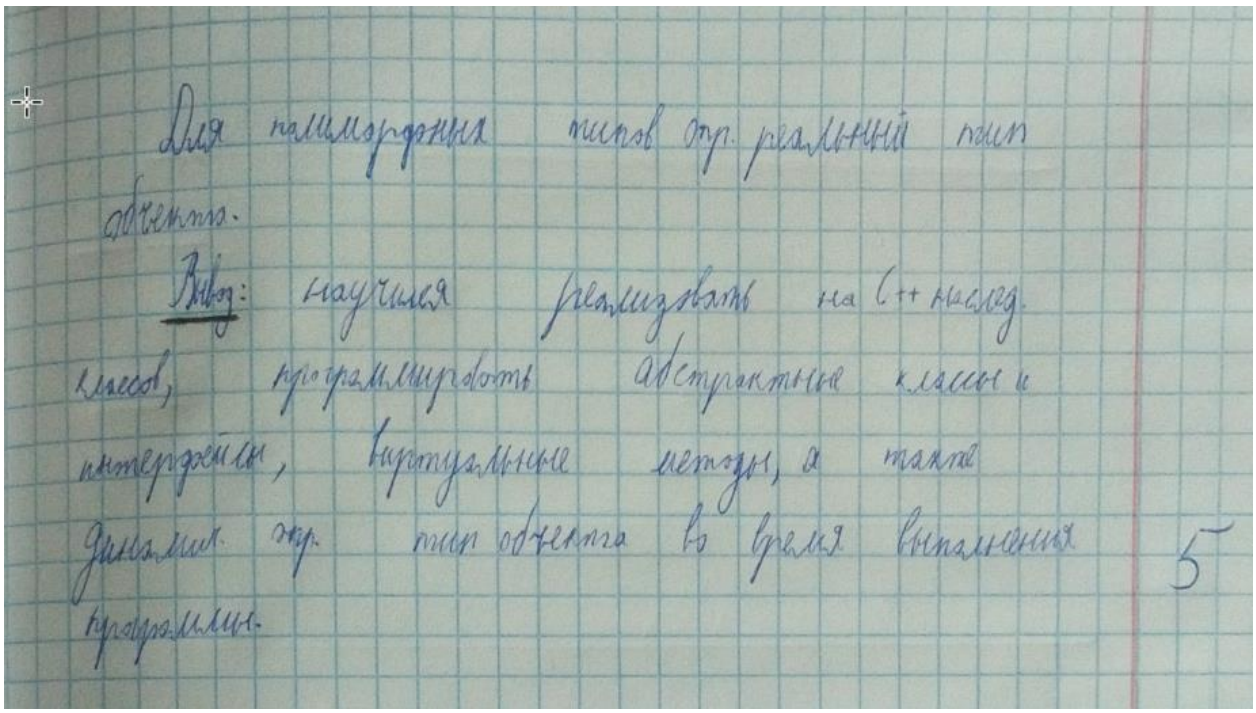
Виртуальный деструктор нужен для корректного удаления через указатель на базовый класс.

9. Что означает RTTI? Какие есть возможности RTTI в C++?

RTTI - идентификация типа во время выполнения.

Возможности:

- dynamic_cast<Derived*>(basePtr): безопасное приведение по иерархии с проверкой реального типа. Возвращает nullptr или ссылку в случае неудачи.
- typeid(expression): возвращает константную ссылку на объект std::type_info, содерж. инф. о типе.



Вывод: научились реализовывать на C++ наследование классов, программировать абстрактные классы и интерфейсы, виртуальные методы, а также динамически определять тип объекта во время выполнения программы.