МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт информационных технологий |
| Кафедра математического и программного обеспечения |
| Объектно-ориентированное программирование |
|  |

ЛАБОРАТОРНАЯ РАБОТА №4

|  |  |
| --- | --- |
| Исполнитель: | студент  группы 1ПИб-02-3оп-23  Богданов  Ренат Алексеевич |
| Руководитель: | Кустов  Максим Александрович |
| Оценка: |  |
| Подпись: |  |

2024 год

**Цель работы**

Изучить способы определения шаблонов, принцип и механизм создания шаблонных классов и шаблонов функций, получить практические навыки работы с шаблонными классами.

## Задания

1. Модифицируйте абстрактный тип данных, реализованный по заданию лабораторной работы 3, согласно варианту задания (см. раздел III УМП\_ООП\_1.doc), используя шаблоны определения класса и шаблоны определения функции.

2. Проверьте работоспособность АТД на тестовом наборе данных для разных типов.

Код программы:

Ide.h

#pragma once

#include <string>

#include <iostream>

enum class IdeType {

TextEditor,

FullFeatured,

CloudBased

};

template <typename TName, typename TVersion>

class Ide {

public:

Ide();

Ide(const TName& name, const TVersion& version, IdeType type);

Ide(const Ide& other);

~Ide();

void showInfo() const;

Ide\* getAdress() const;

IdeType getType() const;

const TName& getName() const;

const TVersion& getVersion() const;

static int getId();

// Перегруженный оператор присваивания (=)

Ide& operator=(const Ide& other) {

if (this != &other) { // Проверка на самоприсваивание

name = other.name;

version = other.version;

}

return \*this; // Возвращаем ссылку на текущий объект

}

// Перегруженный оператор сравнения (==)

bool operator==(const Ide& other) const {

return (name == other.name) && (type == other.type) && (version == other.version);

}

// Перегруженный оператор сравнения (!=)

bool operator!=(const Ide& other) const {

return !((name == other.name) && (type == other.type) && (version == other.version));

}

// Перегруженный оператор сложения (+)

Ide operator+(const Ide& other) const {

return Ide(name + " + " + other.name, version, type);

}

// Перегруженный оператор вычитания (-)

Ide operator-(const Ide& other) const {

return Ide(name + " - " + other.name, version, type);

}

// Перегрузка логического И (&&)

bool operator&&(const Ide& other) const {

return (type == other.type) && (version == other.version);

}

// Перегрузка оператора побитового сдвига (<<)

friend std::ostream& operator<<(std::ostream& out, const Ide& ide) {

out << "Name: " << ide.name << std::endl;

out << "Version: " << ide.version << std::endl;

out << "Type: " << (int)ide.type << std::endl;

return out;

}

private:

static int id;

Ide\* adress;

IdeType type;

TName name;

TVersion version;

void initialize();

};

template <typename TName, typename TVersion>

int Ide<TName, TVersion>::id = 0;

// Конструктор по умолчанию

template <typename TName, typename TVersion>

Ide<TName, TVersion>::Ide() : name("Noname IDE"), version("0.1"), type(IdeType::FullFeatured) {

initialize();

}

// Конструктор с параметрами

template <typename TName, typename TVersion>

Ide<TName, TVersion>::Ide(const TName& name, const TVersion& version, IdeType type) :

name(name), version(version), type(type) {

initialize();

}

// Конструктор копирования

template <typename TName, typename TVersion>

Ide<TName, TVersion>::Ide(const Ide& other) :

name(other.name), version(other.version), type(other.type) {

initialize();

}

// Деструктор

template <typename TName, typename TVersion>

Ide<TName, TVersion>::~Ide() {}

// Метод для вывода информации

template <typename TName, typename TVersion>

void Ide<typename TName, typename TVersion>::showInfo() const {

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

std::cout << "Version: " << version << std::endl;

std::cout << "Type: ";

switch (type) {

case IdeType::TextEditor:

std::cout << "Text Editor" << std::endl;

break;

case IdeType::FullFeatured:

std::cout << "Full Featured" << std::endl;

break;

case IdeType::CloudBased:

std::cout << "Cloud Based" << std::endl;

break;

default:

std::cout << "Unknown Type" << std::endl;

break;

}

std::cout << "ID: " << id << std::endl;

std::cout << "Address: " << adress << std::endl;

}

// Методы доступа к полям

template <typename TName, typename TVersion>

const TName& Ide<typename TName, typename TVersion>::getName() const { return name; }

template <typename TName, typename TVersion>

Ide<typename TName, typename TVersion>\* Ide<typename TName, typename TVersion>::getAdress() const { return adress; }

template <typename TName, typename TVersion>

const TVersion& Ide<typename TName, typename TVersion>::getVersion() const { return version; }

template <typename TName, typename TVersion>

IdeType Ide<typename TName, typename TVersion>::getType() const { return type; }

template <typename TName, typename TVersion>

int Ide<typename TName, typename TVersion>::getId() { return id; }

template <typename TName, typename TVersion>

void Ide<typename TName, typename TVersion>::initialize()

{

++id;

adress = this;

}

Main.cpp

#include <iostream>

#include "ide.h"

int main() {

setlocale(LC\_ALL, "RU");

// Создание экземпляров с разными типами шаблонных параметров

Ide<std::string, std::string> ide1("Visual Studio", "16.10", IdeType::FullFeatured);

Ide<std::string, int> ide2("Sublime Text", 4, IdeType::TextEditor);

Ide<std::string, double> ide3("Eclipse", 4.17, IdeType::CloudBased);

// Вывод информации о каждом экземпляре

std::cout << "IDE 1:" << std::endl;

ide1.showInfo();

std::cout << std::endl;

std::cout << "IDE 2:" << std::endl;

ide2.showInfo();

std::cout << std::endl;

std::cout << "IDE 3:" << std::endl;

ide3.showInfo();

std::cout << std::endl;

// Пример использования конструктора копирования

Ide<std::string, std::string> ide1Copy = ide1;

std::cout << "IDE 1 Copy:" << std::endl;

ide1Copy.showInfo();

std::cout << std::endl;

Ide<std::string, int> ide2Copy(ide2);

std::cout << "IDE 2 Copy:" << std::endl;

ide2Copy.showInfo();

std::cout << std::endl;

Ide<std::string, double> ide3Copy = ide3;

std::cout << "IDE 3 Copy:" << std::endl;

ide3Copy.showInfo();

std::cout << std::endl;

// Сравнение IDE

std::cout << "ide1 == ide2: " << (ide1 == ide1Copy) << std::endl;

// Сравнение с ошибкой из-за разных типов полей

//std::cout << "ide2 == ide3: " << (ide1 == ide3) << std::endl;

// Пример использования методов доступа к полям

std::cout << "Name of IDE 1: " << ide1.getName() << std::endl;

std::cout << "Address of IDE 1: " << ide1.getAdress() << std::endl;

std::cout << std::endl;

std::cout << "Name of IDE 2: " << ide2.getName() << std::endl;

std::cout << "Address of IDE 2: " << ide2.getAdress() << std::endl;

std::cout << std::endl;

std::cout << "Name of IDE 3: " << ide3.getName() << std::endl;

std::cout << "Address of IDE 3: " << ide3.getAdress() << std::endl;

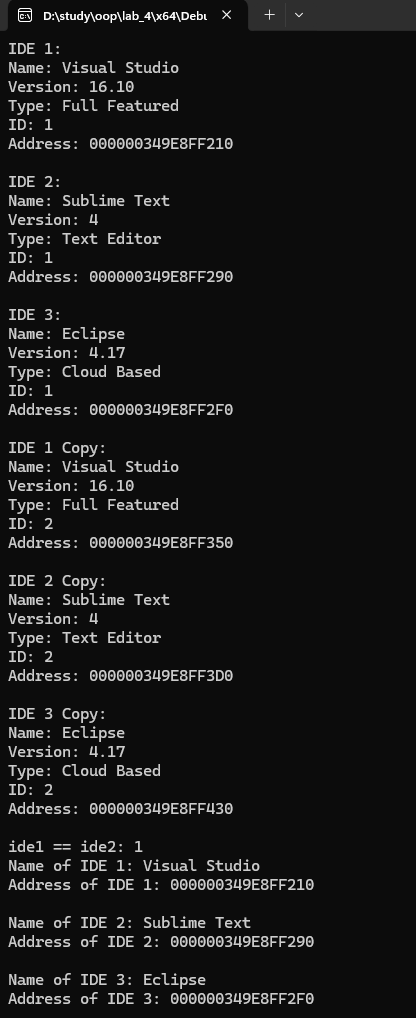
std::cout << std::endl;

std::cin.get();

return 0;

}

Результат работы программы



Ответы на контрольные вопросы

1. *Для чего используется ключевое слово template?*

Для реализации параметризованных типов используется ключевое слово template. Шаблоны определения класса и шаблоны определения функции позволяют многократно использовать код простым способом, безопасным по отношению к типу данных, который разрешает компилятору автоматически реализовать этот тип данных.

1. *В чем заключаются особенности параметров по умолчанию для шаблонов?*

Параметры по умолчанию для шаблонов позволяют указать значения для аргументов шаблона, если они не заданы явно. Это упрощает использование шаблонов, позволяя инстанцировать их с минимальным количеством аргументов.

1. *Для чего используются шаблоны функций?*

Шаблоны функций используются для создания обобщенных функций, которые могут работать с различными типами данных. Это позволяет избежать необходимости переписывать код функции для каждого типа данных.

1. *Назовите разновидности дружественных функций шаблонного класса.*

Шаблонные классы могут содержать «друзей». Дружественные функции могут быть **универсальными** и могут не использовать спецификацию шаблона (для всех инстанцирований шаблонного класса имеется единственный экземпляр дружественной функции), а также могут быть **специфичными** и могут задействовать аргументы шаблона для каждого инстанцирования шаблонного класса.

1. *Дайте определение специализации шаблона.*

Специализация шаблона — это способ определить специализированный вариант шаблона для определенного типа данных или набора аргументов. Это позволяет изменить поведение шаблона для конкретных случаев, не затрагивая его общий код.

**Вывод**

В ходе выполнения лабораторной работы были изучены способы определения шаблонов, принцип и механизм создания шаблонных классов и шаблонов функций. В результате были получены практические навыки работы с шаблонными классами.