МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«РОССИЙСКИЙ БИОТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ (РОСБИОТЕХ)»
Институт промышленной инженерии, информационных технологий и мехатроники
Кафедра «Информатика и вычислительная техника пищевых производств»
Направление: 09.03.01 Информатика и вычислительная техника

ЛАБОРАТОРНАЯ РАБОТА № 7-8

на тему:

« РАЗРАБОТКА И ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ КЛАССОВ В C++ »

Вариант № 14

Выполнил: Студент 1 курса, гр. 24о-090301/ИИ-1

ФИО Капралов Олег Вадимович

Проверил: Ящун Т.В.

Москва, 2025

Цель работы

Изучить принципы разработки и применения шаблонов классов в C++ для создания универсальных, типонезависимых компонентов, способных работать с различными типами данных.

Задания

Шаблонный класс "Слово" (Word)

1. Опишите шаблонный класс Word<FrequencyType, LanguageType>, где:

• FrequencyType - тип данных для частоты употребления (int, float, пользовательский класс

Frequency)

• LanguageType - тип данных для языка (std::string, пользовательский класс Language)

Класс должен содержать:

• Поля:

– слово (std::string word)

– язык (LanguageType language)

– частота употребления (FrequencyType frequency)

– список синонимов (std::vector<std::string> synonyms)

– список антонимов (std::vector<std::string> antonyms)

• конструкторы:

– по умолчанию (пустое слово)

– параметризованный (принимает слово, язык, частоту, синонимы и антонимы)

– конструктор преобразования (из Word<U,V> в Word<T,S>)

– конструктор копирования

– конструктор перемещения

• методы:

– void addSynonym(const std::string&) - добавление синонима

– void addAntonym(const std::string&) - добавление антонима

– void printInfo() - вывод информации о слове

– bool isCommon() - проверка, является ли слово распространенным

– std::string getLanguageFamily() - получение языковой семьи

– int countRelations() - подсчет общего числа связей (синонимы + антонимы)

• перегруженные операции:

– + (объединение списков синонимов)

– += (добавление нового синонима/антонима)

– ++/-- (префиксные и постфиксные - увеличение частоты на 1)

– присваивание (=)

– присваивание перемещением

– сравнение (<, >, == - по частоте употребления)

– ввод/вывод (<<, >>)

• обработка исключений:

– проверка отрицательной частоты употребления

– исключения при добавлении синонима/антонима идентичного слову

– ошибки доступа к пустому списку связей в countRelations()

2. Специализации шаблона:

• для частотных слов (FrequencyType = int):

– добавьте метод bool isHighFrequency() (частота > 1000)

– реализуйте метод std::string getFrequencyCategory() (редкое, среднее, частое)

• для многоязычных слов (LanguageType = std::vector<std::string>):

– добавьте метод bool isInternational()

– реализуйте метод int countTranslations()

• обработка исключений:

– для частотных: исключения в getFrequencyCategory() при FrequencyType < 0

– для многоязычных: ошибки подсчета переводов при пустом LanguageType

3. Шаблонный класс-адаптер WordAdapter<Container>:

• реализуйте класс для преобразования слов в контейнеры: std::vector, std::set, std::map

• выполните анализ списков синонимов и антонимов

• подсчитайте уникальные связи между словами

• обработка исключений:

– исключения при преобразовании в set с конфликтующими языками

– ошибки анализа связей для слов без синонимов/антонимов

4. Протестируйте все созданные методы и операции.

5. Опишите функции для работы со словами:

• сортировка слов по частоте употребления

• поиск слов по языку

• фильтрация слов по количеству связей

• обработка исключений:

– проверка некорректных языковых фильтров

– исключения при сортировке слов с разными типами FrequencyType

Описание работы кода

Шаблонный класс Word хранит слово, его язык, частоту использования и списки синонимов и антонимов. Можно добавлять синонимы и антонимы, менять частоту, сравнивать слова по популярности и выводить информацию. Есть специальные версии для числовой частоты и нескольких языков. Класс-адаптер помогает работать с разными коллекциями слов. Дополнительно есть функции для сортировки, поиска и фильтрации слов. Всё это с проверкой ошибок, чтобы программа работала правильно и удобно.

Задание 1:

Реализован шаблонный класс Idea<YearType, AuthorType>.
Включает поля (название, автор, год, реализации), все требуемые конструкторы, методы, перегрузки операторов (+, +=, ++, --, ==, <, >, <<, >>), обработку исключений.

```h
#ifndef _KAPRALOV_LR7_8_IDEA_H
#define _KAPRALOV_LR7_8_IDEA_H

#include <iostream>
#include <string>
#include <vector>
#include <stdexcept>
#include <map>
#include <algorithm>
#include <sstream>

using namespace std;

// Предварительное объявление шаблона класса
template <typename YearType, typename AuthorType>
class Idea;

// Предварительное объявление шаблонных функций ввода/вывода
template <typename YearType, typename AuthorType>
ostream& operator<<(ostream& os, const Idea<YearType, AuthorType>& obj);

template <typename YearType, typename AuthorType>
istream& operator>>(istream& is, Idea<YearType, AuthorType>& obj);

template <typename YearType, typename AuthorType>
class Idea {
private:
    string title;
    AuthorType author;
    YearType creationYear;
    vector<string> implementations;

public:
    // Конструкторы
    Idea() : title(""), creationYear(YearType()), implementations() {}

    Idea(const string& t, const AuthorType& a, YearType y, const vector<string>& impl = {})
        : title(t), author(a), creationYear(y), implementations(impl) {
        if (y > 2025) throw invalid_argument("Анахроничный год создания");
    }

    template <typename U, typename V>
    Idea(const Idea<U, V>& other)
        : title(other.getTitle()), author(other.getAuthor()),
          creationYear(other.getCreationYear()),
          implementations(other.getImplementations()) {}

    Idea(const Idea& other) = default;
    Idea(Idea&& other) noexcept = default;

    // Методы доступа
    string getTitle() const { return title; }
    AuthorType getAuthor() const { return author; }
    YearType getCreationYear() const { return creationYear; }
    vector<string> getImplementations() const { return implementations; }

    void setTitle(const string& t) { title = t; }
    void setAuthor(const AuthorType& a) { author = a; }
    void setCreationYear(YearType y) {
        if (y > 2025) throw invalid_argument("Анахроничный год создания");
        creationYear = y;
    }
    void setImplementations(const vector<string>& impl) { implementations = impl; }

    // Методы
    void addImplementation(const string& impl) {
        if (impl.empty()) throw invalid_argument("Пустая реализация");
        implementations.push_back(impl);
    }

    void printInfo() const {
        cout << "Идея: " << title << "\nАвтор: " << author
             << "\nГод: " << creationYear << "\nРеализации: ";
        for (const auto& impl : implementations) cout << impl << ", ";
        cout << endl;
    }

    bool isAncient() const { return creationYear < 1000; }

    int countImplementations() const { return implementations.size(); }

    string getEra() const {
        if (implementations.empty()) throw logic_error("Пустой список реализаций");
        if (creationYear < 0) return "До нашей эры";
        if (creationYear < 500) return "Раннее Средневековье";
        if (creationYear < 1500) return "Средневековье";
        if (creationYear < 1900) return "Новое время";
        return "Современность";
    }

    // Перегруженные операторы
    Idea operator+(const Idea& other) const {
        Idea result = *this;
        result.implementations.insert(result.implementations.end(),
                                      other.implementations.begin(),
                                      other.implementations.end());
        return result;
    }

    Idea& operator+=(const string& impl) {
        addImplementation(impl);
        return *this;
    }

    Idea& operator++() { ++creationYear; return *this; }
    Idea operator++(int) { Idea temp = *this; ++(*this); return temp; }
    Idea& operator--() { --creationYear; return *this; }
    Idea operator--(int) { Idea temp = *this; --(*this); return temp; }
```

Задание 2:

Специализации шаблона:
- Для древних идей (YearType=int): добавлены методы isBeforeChrist, getHistoricalPeriod.
- Для коллективных авторов (AuthorType=vector<string>): методы countCoAuthors, hasAuthor.

Обработка исключений по условиям задания.

```cpp
// Специализация для коллективных авторов
template <typename YearType>
class Idea<YearType, vector<string>> {
private:
    string title;
    vector<string> author;
    YearType creationYear;
    vector<string> implementations;

public:
    Idea() : title(""), creationYear(YearType()), implementations() {}

    Idea(const string& t, const vector<string>& a, YearType y, const vector<string>& impl = {})
        : title(t), author(a), creationYear(y), implementations(impl) {
        if (y > 2025) throw invalid_argument("Анахроничный год создания");
        if (a.empty()) throw invalid_argument("Пустой список авторов");
    }

    template <typename U, typename V>
    Idea(const Idea<U, V>& other)
        : title(other.getTitle()), author(other.getAuthor()),
          creationYear(other.getCreationYear()),
          implementations(other.getImplementations()) {}

    Idea(const Idea& other) = default;
    Idea(Idea&& other) noexcept = default;

    // Методы доступа
    string getTitle() const { return title; }
    vector<string> getAuthor() const { return author; }
    YearType getCreationYear() const { return creationYear; }
    vector<string> getImplementations() const { return implementations; }

    void setTitle(const string& t) { title = t; }
    void setAuthor(const vector<string>& a) {
        if (a.empty()) throw invalid_argument("Пустой список авторов");
        author = a;
    }
    void setCreationYear(YearType y) {
        if (y > 2025) throw invalid_argument("Анахроничный год создания");
        creationYear = y;
    }
    void setImplementations(const vector<string>& impl) { implementations = impl; }

    // Методы
    void addImplementation(const string& impl) {
        if (impl.empty()) throw invalid_argument("Пустая реализация");
        implementations.push_back(impl);
    }

    void printInfo() const {
        cout << "Идея: " << title << "\nАвторы: ";
        for (const auto& a : author) cout << a << ", ";
        cout << "\nГод: " << creationYear << "\nРеализации: ";
        for (const auto& impl : implementations) cout << impl << ", ";
        cout << endl;
    }

    bool isAncient() const { return creationYear < 1000; }

    int countImplementations() const { return implementations.size(); }

    string getEra() const {
        if (implementations.empty()) throw logic_error("Пустой список реализаций");
        if (creationYear < 0) return "До нашей эры";
        if (creationYear < 500) return "Раннее Средневековье";
        if (creationYear < 1500) return "Средневековье";
        if (creationYear < 1900) return "Новое время";
        return "Современность";
    }

    int countCoAuthors() const { return author.size(); }

    bool hasAuthor(const string& name) const {
        return find(author.begin(), author.end(), name) != author.end();
    }

    // Перегруженные операторы
    Idea operator+(const Idea& other) const {
        Idea result = *this;
        result.implementations.insert(result.implementations.end(),
                                      other.implementations.begin(),
                                      other.implementations.end());
        return result;
    }
    Idea& operator+=(const string& impl) {
        addImplementation(impl);
        return *this;
    }

    Idea& operator++() { ++creationYear; return *this; }
    Idea operator++(int) { Idea temp = *this; ++(*this); return temp; }
    Idea& operator--() { --creationYear; return *this; }
    Idea operator--(int) { Idea temp = *this; --(*this); return temp; }

    Idea& operator=(const Idea& other) = default;
    Idea& operator=(Idea&& other) noexcept = default;

    bool operator<(const Idea& other) const { return creationYear < other.creationYear; }
    bool operator>(const Idea& other) const { return creationYear > other.creationYear; }
    bool operator==(const Idea& other) const { return creationYear == other.creationYear; }

    friend ostream& operator<< <YearType, vector<string>>(ostream& os, const Idea<YearType, vector<string>>& obj);
    friend istream& operator>> <YearType, vector<string>>(istream& is, Idea<YearType, vector<string>>& obj);
};
```

Задание 3:

Шаблонный класс-адаптер IdeaAdapter<Container>.
Преобразует реализации идеи в разные контейнеры (vector, set, map), поддерживает группировку по эпохам и обработку ошибок.

```
template <template <typename...> class Container>
class IdeaAdapter {
private:
    Container<string> implementations;
public:
    IdeaAdapter(const Idea<int, string>& idea) {
        auto impls = idea.getImplementations();
        implementations = Container<string>(impls.begin(), impls.end());
    }
    void print() const {
        cout << "Реализации в контейнере: ";
        for (const auto& impl : implementations) cout << impl << ", ";
        cout << endl;
    }
};
```

Задание 4:

Тестовые функции для проверки всех методов и операторов класса Idea и его специализаций.
Покрывают создание, вывод, работу адаптера, исключения.

```cpp
void testAncientIdea() {
    try {
        Idea<int, string> ancient("Пирамиды", "Фараон", -2500, {"Каменные блоки"});
        cout << "Древняя идея:\n";
        ancient.printInfo();
        cout << "До нашей эры: " << (ancient.isAncient() ? "Да" : "Нет") << endl;
        cout << "Исторический период: " << ancient.getEra() << endl;
    } catch (const exception& e) {
        cerr << "Ошибка: " << e.what() << endl;
    }
}

void testCollectiveAuthors() {
    try {
        vector<string> authors = {"Иванов", "Петров"};
        Idea<int, vector<string>> idea("Кооператив", authors, 2020, {"Прототип"});
        cout << "Идея с коллективными авторами:\n";
        idea.printInfo();
        cout << "Количество соавторов: " << idea.countCoAuthors() << endl;
        cout << "Есть автор Иванов: " << (idea.hasAuthor("Иванов") ? "Да" : "Нет") << endl;
    } catch (const exception& e) {
        cerr << "Ошибка: " << e.what() << endl;
    }
}

void testAdapter() {
    try {
        Idea<int, string> idea("Тест", "Автор", 2020, {"Имп1", "Имп2"});
        IdeaAdapter<vector> vecAdapter(idea);
        IdeaAdapter<list> listAdapter(idea);
        IdeaAdapter<set> setAdapter(idea);
        cout << "Вектор:\n";
        vecAdapter.print();
        cout << "Список:\n";
        listAdapter.print();
        cout << "Множество:\n";
        setAdapter.print();
    } catch (const exception& e) {
        cerr << "Ошибка: " << e.what() << endl;
    }
}
```

Задание 5:

Функции для работы с коллекцией идей:

- Сортировка по году
- Поиск по автору
- Фильтрация по количеству реализаций

Обработка исключений (пустые массивы, некорректный ввод).

```cpp
void sortIdeas() {
    vector<Idea<int, string>> ideas = {
        {"Идея1", "Автор1", 2000, {"Имплл1"}},
        {"Идея2", "Автор2", 1990, {"Имплл2"}},
        {"Идея3", "Автор3", 2010, {"Имплл3"}}
    };
    sort(ideas.begin(), ideas.end());
    cout << "Отсортированные идеи по году:\n";
    for (const auto& idea : ideas) idea.printInfo();
}

void findIdeaByAuthor() {
    vector<Idea<int, string>> ideas = {
        {"Идея1", "Иванов", 2000, {"Имплл1"}},
        {"Идея2", "Петров", 1990, {"Имплл2"}}
    };
    string author;
    if (validateInput(author, "Введите автора для поиска")) {
        bool found = false;
        for (const auto& idea : ideas) {
            if (idea.getAuthor() == author) {
                idea.printInfo();
                found = true;
            }
        }
        if (!found) cout << "Идеи автора " << author << " не найдены.\n";
    }
}

void filterIdeasByImplCount() {
    vector<Idea<int, string>> ideas = {
        {"Идея1", "Автор1", 2000, {"Имплл1"}},
        {"Идея2", "Автор2", 1990, {"Имплл2", "Имплл3"}},
        {"Идея3", "Автор3", 2010, {"Имплл4", "Имплл5", "Имплл6"}}
    };
    int minImpls;
    if (validateInput(minImpls, "Введите минимальное количество реализаций")) {
        cout << "Идеи с реализациями >= " << minImpls << ":\n";
        for (const auto& idea : ideas) {
            if (idea.countImplementations() >= minImpls) idea.printInfo();
        }
    }
}
```

Тесты(сделано с помощью баш скрипта)

Тест Фильтрация по реализациям
Ввод 1: 8
Ввод 2: 2
Ввод 3: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Введите минимальное количество
реализаций: Идеи с реализациями >= 2:
Идея: Идея2
Автор: Автор2
Год: 1990
Реализации: Импл2, Импл3,
Идея: Идея3
Автор: Автор3
Год: 2010
Реализации: Импл4, Импл5, Импл6,
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден

```
Тест Создание идеи по умолчанию
Ввод 1: 1
Ввод 2: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Идея по умолчанию:
Идея:
Автор:
Год: 0
Реализации:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден
```

Тест Создание идеи с параметрами
Ввод 1: 2
Ввод 2: ТестИдея
Ввод 3: Автор
Ввод 4: 2020
Ввод 5: Импл1
Ввод 6: Импл2
Ввод 7:
Ввод 8: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Введите название идеи: Введите автора:
Введите год создания: Введите реализации (пустая строка для
завершения):
Создана идея:
Идея: ТестИдея
Автор: Автор
Год: 2020
Реализации: Импл1, Импл2,
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден

```
Тест Древняя идея
Ввод 1: 3
Ввод 2: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Древняя идея:
Идея: Пирамиды
Автор: Фараон
Год: -2500
Реализации: Каменные блоки,
До нашей эры: Да
Исторический период: До нашей эры
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден
```

```
Тест Коллективные авторы
Ввод 1: 4
Ввод 2: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Идея с коллективными авторами:
Идея: Кооператив
Авторы: Иванов, Петров,
Год: 2020
Реализации: Прототип,
Количество соавторов: 2
Есть автор Иванов: Да
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден
```

```
Тест Адаптер
Ввод 1: 5
Ввод 2: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Вектор:
Реализации в контейнере: Импл1, Импл2,
Список:
Реализации в контейнере: Импл1, Импл2,
Множество:
Реализации в контейнере: Импл1, Импл2,
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден
```

Тест Сортировка идей
Ввод 1: 6
Ввод 2: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Отсортированные идеи по году:
Идея: Идея2
Автор: Автор2
Год: 1990
Реализации: Импл2,
Идея: Идея1
Автор: Автор1
Год: 2000
Реализации: Импл1,
Идея: Идея3
Автор: Автор3
Год: 2010
Реализации: Импл3,
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден

Тест Поиск по автору
Ввод 1: 7
Ввод 2: Иванов
Ввод 3: 0
Вывод:
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие: Введите автора для поиска: Идея: Идея1
Автор: Иванов
Год: 2000
Реализации: Импл1,
Меню:
1. Создать идею по умолчанию
2. Создать идею с параметрами
3. Тестировать древнюю идею
4. Тестировать коллективных авторов
5. Тестировать адаптер
6. Сортировать идеи
7. Найти идею по автору
8. Фильтровать идеи по реализациям
0. Выход
Выберите действие:
Статус: Пройден