МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №8

по дисциплине: ООП
тема: «Создание шаблонов классов в C++.»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
пр. Черников Сергей Викторович

Белгород 2024 г.

Лабораторная работа №8
«Создание шаблонов классов в C++.»
Вариант 10

Цель работы: Получение теоретических знаний о шаблонах классов в C++. Получение практических навыков по созданию классов-шаблонов C++.

Реализовать шаблон класса в соответствии с указанным вариантом. Предусмотреть необходимые методы для работы со структурой данных, указанной в варианте. Предусмотреть исключительные ситуации, которые могут возникнуть в процессе работы. динамическая матрица (двумерный массив)

main.cpp

```
#include <iostream>

#include "../../libs/alg/matrix/matrix.tpp"

int main() {
    Matrix t({1, 2, 3,
              4, 5, 6,
              7, 8, 9}, 3, 3);

    t.push_back({10,11,12});

    std::cout << t(3, 2) << std::endl;
}
```

matrix.tpp

```
#ifndef MATRIX_TPP
#define MATRIX_TPP

#include <initializer_list>
#include <stdexcept>
#include <malloc.h>
#include <iostream>

template <typename T>
class Matrix;

template <typename T>
class MatrixRow {
    Matrix<T>* matrix;
    int index;

public:
    MatrixRow(Matrix<T>* matrix, int index) : matrix(matrix), index(index) {};

    void push_back(T element);
    T& operator()(int index_column);

    friend class Matrix<T>;
};
```

```cpp
template <typename T>
class Matrix {
    T **data = nullptr;
    int width = 0;
    int height = 0;
    int width_capacity = 0;
    int height_capacity = 0;

public:
    Matrix() {};
    Matrix(std::initializer_list<T> init, int width, int height);
    ~Matrix();

    void push_back(std::initializer_list<T> row);
    void resize_width(int new_width);
    void resize_height(int new_height);

    int get_width();
    int get_height();

    T& operator()(int index_row, int index_column);
    MatrixRow<T> operator()(int index_row);

    friend class MatrixRow<T>;
};

template <typename T>
void MatrixRow<T>::push_back(T element) {
    this->matrix->resize_width(this->matrix->width + 1);
    this->matrix->data[this->index][this->matrix->width - 1] = element;
}

template <typename T>
T& MatrixRow<T>::operator()(int index_column) {
    if (this->index < 0 || this->index >= this->matrix->height || index_column < 0 || index_column >=
    ↪  this->matrix->width)
        throw std::out_of_range("Out of bounds");

    auto& res = this->matrix->data[this->index][index_column];

    return res;
}

template <typename T>
Matrix<T>::Matrix(std::initializer_list<T> init, int width, int height) {
    if (init.size() != width * height)
        throw std::invalid_argument("Initializer list size conditions are not met");

    this->resize_width(width);
    this->resize_height(height);

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
```

```
            (*this)(i, j) = *(init.begin() + (i * width + j));
        }
    }
}

template <typename T>
void Matrix<T>::push_back(std::initializer_list<T> row) {
    if (row.size() != this->width)
        throw std::invalid_argument("Initializer list size conditions are not met");

    this->resize_height(this->height + 1);
    for (int i = 0; i < this->width; i++) {
        (*this)(this->height - 1, i) = *(row.begin() + i);
    }
}

template <typename T>
void Matrix<T>::resize_width(int new_width) {
    this->width = new_width;

    if (this->width > this->width_capacity) {
        this->width_capacity = std::max(this->width * 2 + 1, this->width_capacity * 2 + 1);

        for (int i = 0; i < this->height_capacity; i++) {
            this->data[i] = (T*) realloc(this->data[i], this->width_capacity * sizeof(T));

            if (!this->data[i])
                throw std::length_error("Out of memory");
        }
    }
}

template <typename T>
void Matrix<T>::resize_height(int new_height) {
    this->height = new_height;

    if (this->height > this->height_capacity) {
        int old_height_capacity = this->height_capacity;

        this->height_capacity = std::max(this->height * 2 + 1, this->height_capacity * 2 + 1);
        this->data = (T**) realloc(this->data, this->height_capacity * sizeof(T*));

        if (!this->data)
            throw std::length_error("Out of memory");

        for (int i = old_height_capacity; i < this->height_capacity; i++) {
            this->data[i] = (T*) malloc(this->width_capacity * sizeof(T));

            if (!this->data[i])
                throw std::length_error("Out of memory");
        }
    }
}
```

```
template <typename T>
T& Matrix<T>::operator()(int index_row, int index_column) {
    if (index_row < 0 || index_row >= this->height || index_column < 0 || index_column >= this->width)
        throw std::out_of_range("Out of bounds");

    auto& res = this->data[index_row][index_column];

    return res;
}

template <typename T>
MatrixRow<T> Matrix<T>::operator()(int index_row) {
    if (index_row < 0 || index_row >= this->height)
        throw std::out_of_range("Out of bounds");

    return MatrixRow(this, index_row);
}

template <typename T>
Matrix<T>::~Matrix() {
    for (int i = 0; i < this->height_capacity; i++)
        free(this->data[i]);

    free(this->data);
}

template <typename T>
int Matrix<T>::get_width() {
    return this->width;
}

template <typename T>
int Matrix<T>::get_height() {
    return this->height;
}

#endif // MATRIX_TPP
```

На основе разработанного шаблона решить прикладную задачу в соответствии с выбранным вариантом? Дан файл структур с тремя полями: фамилия, возраст, оценка. Реализовать структуру для упорядоченного поиска элемента из файла по возрасту, оценки, фамилии. В качестве метода сравнения двух элементов обязательно использовать template функцию.

main.cpp

```
#include "../../libs/alg/studentslist/studentslist.h"

#include <cstring>
#include <fstream>
#include <algorithm>
#include <iostream>

bool filter_by_name(Student& a, CharString & val) {
```

```
    return strcmp(a.name.val, val.val) == 0;
}

struct {
    bool operator()(const Student &a, const Student &b) const {
        int mark1 = std::atoi(a.mark.val);
        int mark2 = std::atoi(b.mark.val);

        return mark1 < mark2;
    }
} sort_by_mark;

int main() {
    StudentsList list;

    std::ifstream in("data.txt");
    while (!in.eof()) {
        std::string val;
        std::getline(in, val);
        auto space_pos = std::find(val.begin(), val.end(), ' ');
        std::string name = val.substr(0, space_pos - val.begin());
        CharString namec;
        strcpy(namec.val, name.c_str());

        auto old_space_pos = space_pos + 1;
        space_pos = std::find(old_space_pos, val.end(), ' ');
        std::string age = val.substr(old_space_pos - val.begin(), space_pos - old_space_pos);
        CharString agec;
        strcpy(agec.val, age.c_str());

        old_space_pos = space_pos + 1;
        std::string mark = (val.substr(old_space_pos - val.begin(), space_pos - old_space_pos));
        CharString markc;
        strcpy(markc.val, mark.c_str());

        list.add({namec, agec, markc});
    }
    in.close();
    CharString search_val = {"Christina"};

    auto search = list.find<CharString>(search_val, sort_by_mark, filter_by_name);

    for (auto& student : search)
        std::cout << student.name.val << " " << student.age.val << " " << student.mark.val << std::endl;
}
```

studentslist.h

```
#ifndef STUDENTSLIST_H
#define STUDENTSLIST_H

#include "../matrix/matrix.tpp"
```

```cpp
#include <string>
#include <vector>
#include <functional>

struct CharString {
    char val[50];
};

class Student {
public:
    CharString name;
    CharString age;
    CharString mark;

    Student(CharString name, CharString age, CharString mark) : name(name), age(age), mark(mark) {};
};

class StudentsList
{
    Matrix<CharString> students;
public:
    StudentsList() {
        students.resize_width(3);
    }

    void add(Student student);
    template <typename T>
    std::vector<Student> find(T val, std::function<bool (Student&, Student&)> sort_func, std::function<bool (Student&,
    ↪  T&)> filter);
};

void StudentsList::add(Student student) {
    this->students.push_back({student.name, student.age, student.mark});
}

template <typename T>
std::vector<Student> StudentsList::find(T val, std::function<bool (Student&, Student&)> sort_func, std::function<bool
↪  (Student&, T&)> filter) {
    std::vector<Student> result;
    for (int i = 0; i < this->students.get_height(); i++){
        Student student = {this->students(i, 0), this->students(i, 1), this->students(i, 2)};

        if (filter(student, val))
            result.push_back(student);
    }

    std::sort(result.begin(), result.end(), sort_func);

    return result;
}

#endif // STUDENTSLIST_H
```

Ссылка на репозиторий

Вывод: в ходе лабораторной работы получили теоретических знаний о шаблонах классов в C++. Получение практических навыков по созданию классов-шаблонов C++.