Министерство образования Республики Беларусь
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ № 5
Шаблоны функций и классов

по дисциплине «Программирование на языках высокого уровня»

Выполнил ст. гр. 450503 А.П. Красько

Проверил асс. каф. ЭВМ И.Г. Скиба

Минск 2025

1 ПОСТАНОВКА ЗАДАЧИ

Создать параметризованный массив с конструкторами, деструктором и перегруженными операторами + , =.

2 ДИАГРАММА КЛАССОВ

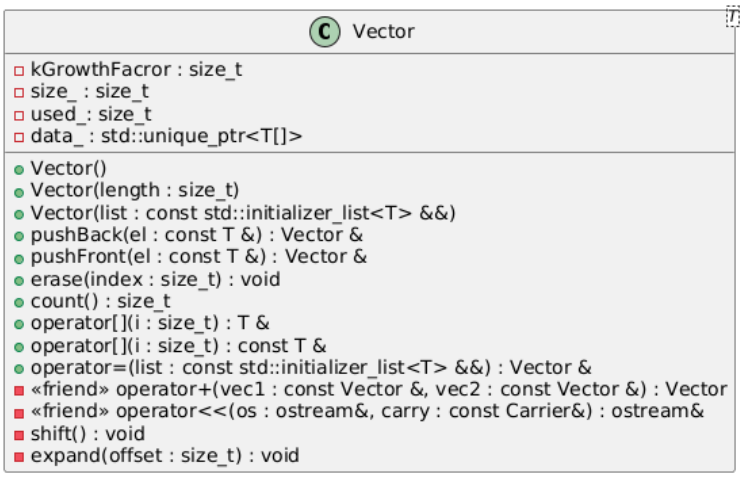

Рисунок 2.1 – Диаграмма классов

3 ЛИСТИНГ КОДА

Файл screens.hh:

```
#pragma once
#include <memory>
#include "vector.hh"
namespace screens {
void printMainScreen();
bool inputVector(vec::Vector<double> &vec);
bool addVectors(const vec::Vector<double> &vec1,const vec::Vector<double>
&vec2);
bool printVectors(const vec::Vector<double> &vec1,const vec::Vector<double>
&vec2);
}  // namespace screens
```

Файл vector.hh:

```
#pragma once
#include <exception>
#include <format>
#include <initializer_list>
#include <iostream>
#include <memory>
#include <ranges>
```

```
namespace vec {
template <class T>
class Vector {
   private:
    constexpr static const size_t kGrowthFacror = 2;
    size_t size_;
    size_t used_ = 0;
    std::unique_ptr<T[]> data_;

   private:
    friend std::ostream &operator<<(std::ostream &os, const Vector &vec) {
        os << '[';
        for (size_t i = 0; i < vec.used_; i++) {
            os << vec.data_[i] << ' ';
        }
        os << "]";
        return os;
    }
    friend Vector operator+(const Vector &vec1, const Vector &vec2) {
        if (vec1.used_ != vec2.used_) throw std::invalid_argument("Vector
lengths mismatch");
        Vector res(vec1.used_);
        for (size_t i = 0; i < vec1.used_; i++) {
            res[i] = vec1.data_[i] + vec2.data_[i];
        }
        return res;
    }

    void shift() {
        auto tmpList = std::make_unique_for_overwrite<T[]>(size_);
        std::copy(data_.get(), data_.get() + used_, tmpList.get() + 1);
        data_ = std::move(tmpList);
    }
    void expand(size_t offset = 0) {
        size_ = size_ * kGrowthFacror;
        if (size_ == 0) size_ = 1;
        auto tmpList = std::make_unique_for_overwrite<T[]>(size_);
        std::copy(data_.get(), data_.get() + used_, tmpList.get() + offset);
        data_ = std::move(tmpList);
    }

   public:
    Vector() : size_{0}, data_{nullptr} {};
    explicit Vector(size_t length) : size_{length}, used_{length},
data_{std::make_unique<T[]>(size_)} {}
    Vector(const std::initializer_list<T> &&list)
        : size_{list.size()}, used_{size_},
data_{std::make_unique_for_overwrite<T[]>(size_)} {
        if (size_ == 0) throw std::invalid_argument("empty
initializer_list");
        std::ranges::copy(list, data_.get());
    }

    Vector &operator=(const std::initializer_list<T> &&list) {
        if (list.size() == 0) throw std::invalid_argument("empty
```

```
initializer_list");
        size_ = list.size();
        data_ = std::move(std::make_unique_for_overwrite<T[]>(size_));
        std::ranges::copy(list, data_.get());
        return *this;
    };

    T &operator[](size_t i) { return data_[i]; }
    const T &operator[](size_t i) const { return data_[i]; }

    Vector &pushBack(const T &el) {
        if (used_ == size_) expand();
        data_[used_] = el;
        used_++;
        return *this;
    }
    Vector &pushFront(const T &el) {
        if (used_ == size_)
            expand(1);
        else
            shift();
        data_[0] = el;
        used_++;
        return *this;
    }
    void erase(size_t index) {
        if (index >= used_) return;
        T *data = data_.get();
        std::copy(data + index + 1, data + used_, data + index);
        used_--;
    }
    size_t count() const { return used_; }
};

}  // namespace vec
```

Файл screens.cc:

```
#include <consoleUtils.hh>
#include <l5/include/vector.hh>
#include <limits>
#include <memory>
#include <print>
#include <sstream>

using namespace std;
using namespace vec;
using namespace console_utils;

namespace screens {
void printMainScreen() {
    auto [cols, rows] = getConsoleDimensions();
    println("{:^{}}", "\x{1B}[48;5;35mLab 5\x{1B}[0m", cols);
```

```
    println("Please select action:\n");
    println("    1.Input first vector");
    println("    2.Input second vector");
    println("    3.Print vectors");
    println("    4.Add");
    println("    5.Exit");
}

bool inputVector(Vector<double> &vec) {
    string line;
    print("Please enter vector (numbers separated by spaces): ");
    getline(std::cin, line);
    istringstream iss(line);
    double num;
    vec::Vector<double> tmpVec;
    while (iss >> num) {
        tmpVec.pushBack(num);
    }
    cin.clear();
    cout << tmpVec;
    vec = std::move(tmpVec);
    return true;
}
bool addVectors(const Vector<double> &vec1, const Vector<double> &vec2) {
    print("addition result: ");
    try {
        cout << vec1 + vec2 << endl;
    } catch (const std::invalid_argument &e) {
        std::cerr << "Error: " << e.what() << '\n';
    }

    return true;
}
bool printVectors(const Vector<double> &vec1, const Vector<double> &vec2) {
    cout << vec1 << endl;
    cout << vec2 << endl;
    return true;
}
}  // namespace screens
```

Файл main.cc:

```
#include <consoleUtils.hh>
#include <functional>
#include <l5/include/vector.hh>
#include <l5/include/screens.hh>


#include <iostream>
#include <memory>


using namespace std;
using namespace console_utils;
using namespace screens;
```

```
int main(void) {
    vec::Vector<double> vec1;
    vec::Vector<double> vec2;
    static array<function<bool()>, 5> actions = {
        [&vec1]() { return inputVector(vec1); },
        [&vec2]() { return inputVector(vec2); },
        [&vec1, &vec2]() { return printVectors(vec1, vec2); },
        [&vec1, &vec2]() { return addVectors(vec1, vec2); },
        []() { return false; },

    };
    unsigned int response;
    do {
        printMainScreen();
        readT(response, ">", [](unsigned int num) { return num > 0 && num <=
5; });
        cout << "\x{1B}[2J\x{1B}[H\n";
    } while (actions[response - 1]());

    return 0;
}
```

Файл consoleUtils.hh:

```
#pragma once
#include <functional>
#include <iostream>
#include <limits>
#include <iomanip>
#include <chrono>
namespace console_utils {
std::pair<int, int> getConsoleDimensions();


template <typename T, typename CT>
void readT(T& data, const std::string& massage, CT validator) {
    std::cout << massage;
    while (((std::cin >> data).fail()) || !validator(data)) {
        std::cout << "Invalid input. Reread input requierments\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << massage;
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
template <typename T>
void readT(T& data, const std::string& massage) {
    std::cout << massage;
    while ((std::cin >> data).fail()) {
        std::cout << "Invalid input. Reread input requierments\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << massage;
    }
```

```
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
template <typename T, typename CT>
void readT(T& data, const std::string& message, CT validator, const
std::string& errmess) {
    std::cout << message;
    while (((std::cin >> data).fail()) || !validator(data)) {
        std::cout << errmess;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << message;
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
template <typename T = std::chrono::sys_seconds, typename CT = const char * >
void readT(std::chrono::sys_seconds &data,  const char * massage, const char
* format) {
    std::cout << massage;
    while ((std::cin >> std::chrono::parse(format, data)).fail()) {
        std::cout << "Invalid input. Reread input requierments\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << massage;
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
}  // namespace console_utils
```

Файл consoleUtils.cc:

```
#include <iostream>

#ifdef __linux__
#include <sys/ioctl.h>
#include <unistd.h>
#endif

#ifdef _WIN32
#include <Windows.h>
#endif

namespace console_utils {
std::pair<int, int> getConsoleDimensions() {
#ifdef _WIN32
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    return std::make_pair(csbi.srWindow.Right - csbi.srWindow.Left + 1,
csbi.srWindow.Bottom - csbi.srWindow.Top + 1);
#endif
#ifdef __linux__
```

```
    struct winsize w;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
    return std::make_pair(w.ws_col, w.ws_row);

#endif
}
}  // namespace console_utils
```

4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Рисунок 4.1 – Главное меню

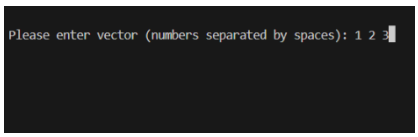

Рисунок 4.2 – Меню ввода массива

Рисунок 4.3 – Вывод массивов

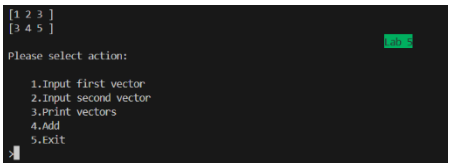

Рисунок 4.4 – Результат сложения

5 ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы был разработан параметризованный класс Vector, реализующий динамический массив с поддержкой конструкторов, деструктора и перегруженных операторов + и =.