**Федеральное государственное автономное образовательное учреждение высшего образования   
"Национальный исследовательский университет   
"Высшая школа экономики"**

Факультет: Московский институт электроники и математики

Департамент компьютерной инженерии

**Методические указания по выполнению компьютерного практикума «Инструментальные средства и технологии программирования»**

**по теме**

**Элементы стандартной библиотеки шаблонов языка C++**

Выполнила: Опекунова А. А.

Группа БИВ193

|  |
| --- |
|  |

**Москва 2022**

**Оглавление**

[Цели 3](#_Toc98159169)

[Выполнение 4](#_Toc98159170)

[Задание 1 4](#_Toc98159171)

[Задание 2 8](#_Toc98159172)

[Задание 3 9](#_Toc98159173)

[Задание 4 14](#_Toc98159174)

[Задание 5 15](#_Toc98159175)

[Задание 6 17](#_Toc98159176)

[Задание 7 18](#_Toc98159177)

[Задание 8 20](#_Toc98159178)

[Задание 9 23](#_Toc98159179)

# Цели

**Цель выполнения заданий практикума**: освоить работу с контейнерами, итераторами, алгоритмами и функциональными объектами.

**Правила выполнения заданий:** все задания практикума выполняются в фреймворке Qt (IDE Qt Creator, библиотека классов Qt). Все задания практикума подразумевают GUI. Сдать требуется отчет, включающий краткое описание программы, листинг и скриншоты работающей программы.

# Выполнение

Полный код проекта: <https://github.com/AlinaOpekunova/Qt_pr_1.git>

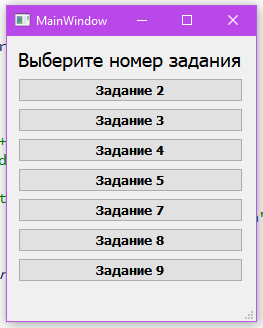


Рисунок 1 - Главное окно приложения

После запуска проекта появляется главное окно приложения (Рисунок 1), где можно увидеть кнопки с номерами заданий и выбрать нужную для тестирования работы.

## **Задание 1**

**Задание**: разработать собственные классы vector(массив), list(список), itor(итератор) в соответствие с предложенными объявлениями публичных свойств и методов (необходимые приватные поля и методы разработать самостоятельно):

Реализация класса vector:

#ifndef MYVECTOR\_H

#define MYVECTOR\_H

#include <cstddef>

#include<iostream>

using namespace std;

template <class T> class **Vector**{

public:

explicit **Vector**(size\_t n)

{// конструктор создает вектор указанного размера

sizeVector = n;

data = new T[n];

};

T& operator[](size\_t index)

{// получение элемента по индексу

return data[index];

};

size\_t **size**()

{// получение размера вектора

return sizeVector;

};

void **push**(int value, int index)

{// добавление элемента по индексу

data[index] = value;

}

void **print**()

{// вывод вектора на экран

for (size\_t i = 0; i < sizeVector; i++) {

cout << data[i] << " ";

}

cout << endl;

}

void **remove**(size\_t index)

{// удаление элемента по индексу

for (size\_t i = index + 1; i < sizeVector; ++i) {

data[i - 1] = data[i];

}

--sizeVector;

}

~**Vector**()

{// деструктор

delete [] data;

};

private:

T\* data;

size\_t sizeVector;

};

#endif // MYVECTOR\_H

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 - Демонстрация работы вектора

Реализация класса list:

#ifndef MYLIST\_H

#define MYLIST\_H

template <class T> class **List** {

public:

struct **Link** {

Link\* pre; // ссылка на предыдущий элемент

Link\* suc; // ссылка на следующий элемент

T data;

**Link**():suc(nullptr),pre(nullptr){};

**Link**(T d):data(d),suc(nullptr),pre(nullptr){};

};

Link\* head = nullptr;

Link\* tail = nullptr;

Link\* **getHead**(){

return head;

};

Link\* **getTail**(){

return tail;

};

void **put**(T value)

{// Вставить перед текущим элементом

Link\* current = new Link(value);

current->pre = tail->pre;

tail -> pre -> suc = current;

tail -> pre = current;

current -> suc = tail;

};

T **get**(Link \*node)

{ // Получить текущий элемент

return node->data;

};

void **push\_back**(T value)

{

Link\* current = new Link(value);

if (head == nullptr)

{

head = current;

}

else

{

tail -> suc = current;

current -> pre = tail;

}

tail = current;

}

void **push\_front**(T value)

{

Link\* current = new Link(value);

if (head == nullptr)

{

head = current;

tail = current;

}

else

{

head -> pre = current;

current -> suc = head;

head = current;

}

};

};

#endif // MYLIST\_H

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 - Демонстрация работы контейнера list

Реализация класса :

template <class T> class Itor {

public:

// Указатель на первый элемент

virtual T\* first() = 0;

// Указатель на следующий элемент

virtual T\* next() = 0;

};

template <class T> class Vector\_itor: public Itor<T> {

Vector<T>& v;

size\_t index;

public:

Vector\_itor (Vector <T>& vv);

T\* first();

T\* next();

}

template <class T> class List\_itor: public Itor<T> {

Public:

List\_itor(ListT>&);

T\* first();

T\* next();

}

## **Задание 2**

**Задание**: создать контейнер vector<char>, содержащий латинские буквы в алфавитном порядке. Выведите элементы вектора в прямом и обратном порядке в окно редактора по нажатию кнопок «Fwd» и «Rew» соответственно.

На Рисунок 4 показан листинг программы, а на Рисунок 5 и Рисунок 6 – результаты работы алгоритма.

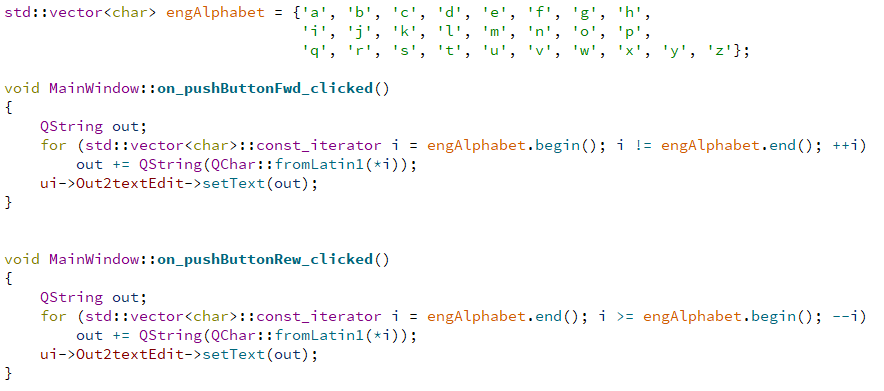


Рисунок 4 - Листинг программы задания 2

Сначала создаем контейнер vector<char> и инициализируем его буквами английского алфавита.

В окне приложения есть две кнопки, которые отвечают за вывод вектора в прямом или обратном порядке. При нажатии на кнопку «Fwd» вызывается функция void MainWindow::on\_pushButtonFwd\_clicked(), где в цикле for с использованием итераторов элементы вектора перебираются в прямом порядке и преобразуются из типа std::string в QString для вывода в textEdit на экран. Аналогично, при нажатии на кнопку «Rew» вызывается функция void MainWindow::on\_pushButtonRew\_clicked(), где в цикле for с использованием итераторов элементы вектора перебираются в обратном порядке и преобразуются из типа std::string в QString для вывода в textEdit на экран.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 - Вывод вектора в прямом порядке

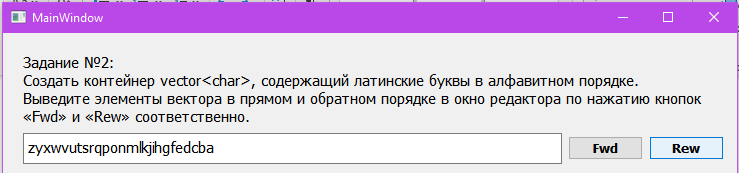


Рисунок 6 - Вывод вектора в обратном порядке

## **Задание 3**

**Задание**: создать контейнер vector<string>. Считать в него названия фруктов из окна редактора или заранее подготовленного файла. Названия фруктов вводить на английском языке. Отсортировать содержимое вектора и вывести отсортированный вектор в окно результатов. Выведите имена всех фруктов, начинающихся на «а». Используя разработанный вектор, разработайте цикл для удаления всех цитрусовых.

Интерфейс программы состоит из 4 кнопок: Отсортировать, Вывести фрукты, начинающиеся на «а», Удалить цитрусовые, Очистить.

При нажатии на кнопку «Отсортировать» вызывается функция void ThirdTask::on\_sortPushButton\_clicked()(Рисунок 7), в которой из левого текстового поля (предназначен для ввода) считываются названия фруктов, потом в цикле for тип введенных данных преобразуется в string и записывается в ранее созданный контейнер inputVector. Далее с использованием функции sort вектор сортируется по алфавиту, в следующих строчках кода элементы вектора преобразуются из типа std::string в QString для вывода в правое поле на экране (Рисунок 8).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 - Листинг нажатия на кнопку "Отсортировать"

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 - Работа кнопки "Отсортировать"

При нажатии на кнопку «Вывести фрукты, начинающиеся на «а»» (Рисунок 9) вызывается функция void ThirdTask::on\_FruitPushButton\_clicked(), в которой из вектора с названием фруктов в цикле for перебираются все элементы контейнера и в строку out, созданную для вывода данных на экран, записывают только фрукты, начинающиеся на «а» (Рисунок 10).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 - Листинг нажатия на кнопку «Вывести фрукты, начинающиеся на «а»»

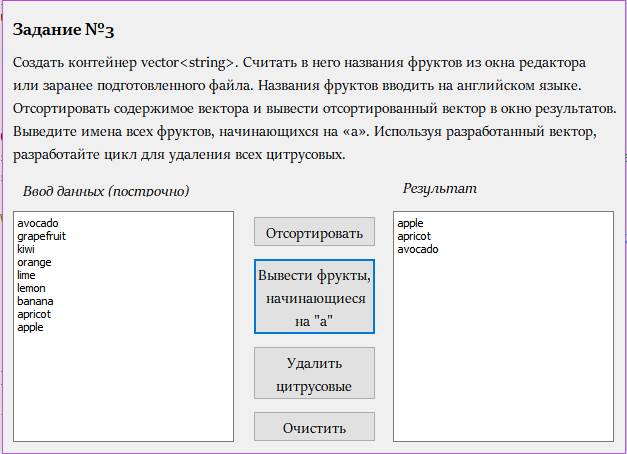


Рисунок 10 - Работа кнопки «Вывести фрукты, начинающиеся на «а»»

При нажатии на кнопку «Удалить цитрусовые» (Рисунок 11) вызывается функция void ThirdTask::on\_deletePushButton\_clicked(), в которой в цикле while с использованием итераторов перебираются элементы контейнера и сравниваются с элементами вектора citrus, если фрукт является цитрусовым, то он удаляется из вектора методом erase(). В следующих строчках кода элементы вектора преобразуются из типа std::string в QString для вывода в правое поле на экране (Рисунок 12).

При нажатии на кнопку «Очистить» удаляются все элементы вектора и очищаются поля для ввода и вывода на экране приложения.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 - Листинг нажатия на кнопку "Удалить цитрусовые"

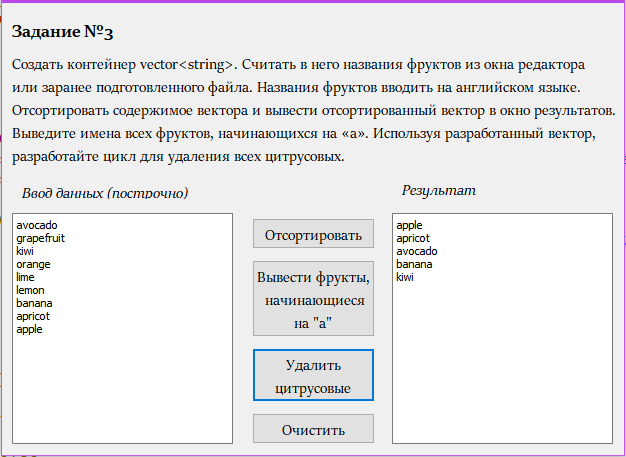


Рисунок 12 - Работа кнопки "Удалить цитрусовые"

## **Задание 4**

**Задание**: cчитать даты, вводимые из редактора, например Dec85, Dec50, Jan76, а потом вывести их так, чтобы более поздние даты шли первыми. Формат дат состоит из 3-х символов под месяц, после чего следуют 2 символа под год. Все даты относятся к одному веку.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 - Листинг работы с датами

По нажатию на кнопку «Вывести сортированные даты» (Рисунок 13) вызывается функция void FourthTask::on\_sortPushButton\_clicked(), в которой сначала считываются даты из поля ввода, далее в первом цикле for даты сортируются по годам, во втором – по месяцам. Метод QString QString::mid(int position, int n = -1) возвращает строку, содержащую n символов этой строки, начиная с указанного индекса позиции, используя этот метод мы можем получить часть строки с месяцем или годом (Рисунок 14).

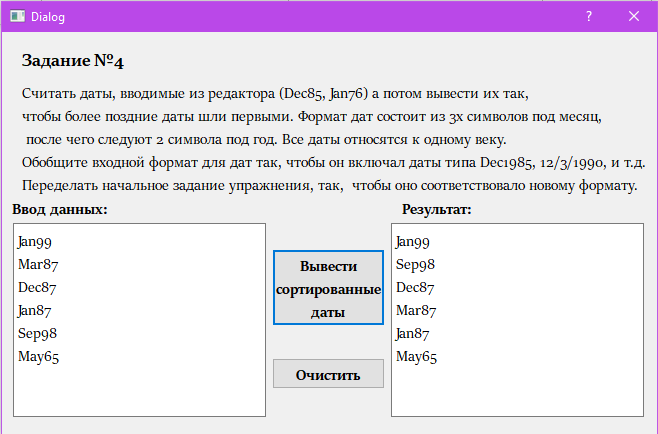


Рисунок 14 - Демонстрация работы программы

## **Задание 5**

**Задание**: используя контейнер bitset сохранять информацию о присутствии студентов на занятиях. По результатам чтения за 12 дней, определить, кто присутствует всегда, кто присутствует менее 8 дней.

Введя количество студентов в соответствующее поле и нажимая кнопку «Result»(Рисунок 15), запускается функция для подсчета присутствующих, в которой сначала инициализируются необходимые переменные и создается контейнер указанного размера. Далее в цикле for этот контейнер заполняется значениями. В следующем цикле for подсчитывается посещаемость с использование метода count() контейнера bitset (этот метод подсчитывает количество единичных битов в контейнере). Строка out предназначена для вывода в текстовое поле на экране(Рисунок 16).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 - Листинг подсчета посещаемости студентов

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16 - Работа приложения по подсчету посещаемости

## **Задание 6**

**Задание**: реализовать расширенный контейнер list, поддерживающий индексацию (возможность доступа к элементу по индексу).

В данном задании реализуем шаблонный класс NewList (Рисунок 17), используя паттерн заместителя, для которого перегрузим оператор [] и проверим работу на примере контейнера list типа int (Рисунок 18).

Для начала получим ссылку на начало контейнера, далее с помощью функции advance() передвигаем итератор на указанное число позиций и возвращаем элемент, стоящий на этой позиции.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 - Реализация индексации

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 18 - Демонстрация работы индексации

## **Задание 7**

**Задание**: создать вектор квадратов целых чисел (от 1до 100). С помощью алгоритма for\_each и разработанного функтора вывести в окно редактора, значения всех элементов вектора. Получить новый вектор, значения элементов которого равны корню значений элементов исходного вектора. Вывести в окно редактора значения элементов нового вектора.

Окно приложения состоит из двух кнопок, нажимая на которые выводится один или другой вектор в предназначенные для этого поля (**Ошибка! Источник ссылки не найден.**).

В начале каждой функции инициализируются созданные контейнеры, далее с помощью for\_each и функции Printer выводятся в текстовое поле вектор из чисел (Рисунок 19).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 19 - Листинг программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 20 - Работа программы

## **Задание 8**

**Задание**: разработать набор функциональных объектов, выполняющих логические операции над своими операндами. Проверить объекты на векторах типов char, int, bitset<67> (Рисунок 21)

Для выполнения задания созданы классы-шаблоны, соответствующие логическим операциям.

Листинг "logoperation.h":

#ifndef LOGOPERATION\_H

#define LOGOPERATION\_H

#include<QString>

#include <vector>

#include <bitset>

using namespace std;

template <class T> class **And** {

public:

vector<T> operator()(vector<T> x, vector<T> y) {

unsigned int size = 0;

if (x.size() < y.size()){

size = x.size();

} else {

size = y.size();

}

vector<T> result(size);

int index = 0;

for (unsigned int i=0; i < size; i++){

result[index] = (x[i] & y[i]);

index++;

}

return result;

}

};

template <class T> class **Or** {

public:

vector<T> operator()(vector<T> x, vector<T> y) {

unsigned int size = 0;

if (x.size() > y.size())

size = x.size();

else

size = y.size();

vector<T> result(size);

int index = 0;

for (unsigned int i=0; i < size; i++){

if (i < x.size() && i < y.size())

result[index++] = (x[i] | y[i]);

else if (i >= x.size() && i < y.size())

result[index++] = y[i];

else if (i < x.size() && i >= y.size())

result[index++] = x[i];

}

return result;

}

};

template <class T> class **Xor** {

public:

vector<T> operator()(vector<T> x, vector<T> y) {

unsigned int size = 0;

if (x.size() > y.size())

size = x.size();

else

size = y.size();

vector<T> result(size);

int index = 0;

for (unsigned int i=0; i < size; i++){

if (i < x.size() && i < y.size())

result[index++] = (x[i] ^ y[i]);

else if (i >= x.size() && i < y.size())

result[index++] = y[i];

else if (i < x.size() && i >= y.size())

result[index++] = x[i];

}

return result;

}

};

template <class T> class **Not** {

public:

vector<T> operator()(vector<T> x) {

vector<T> result(x.size());

unsigned int index = 0;

for (unsigned int i=0; i < x.size(); i++){

result[index++] = (~x[i]);

}

return result;

}

};

// Следующие классы предназначены для преобразования типов int, char, bitset в окна приложения

class **IntPrinter**

{

public:

QString operator()(std::vector<int> &numbers);

};

class **CharPrinter**

{

public:

QString operator()(std::vector<char> &chars);

};

class **BitsetPrinter**

{

public:

QString operator()(std::vector<bitset<67>> &chars);

};

#endif // LOGOPERATION\_H

Листинг: "logoperation.сpp"

#include "logoperation.h"

#include <vector>

#include <iostream>

#include <bitset>

QString IntPrinter::operator()(std::vector<int> &numbers){

QString outInt = "";

for (std::vector<int>::const\_iterator i = numbers.begin(); i != numbers.end(); i++)

outInt += QString::number(\*i)+" ";

return outInt;

}

QString CharPrinter::operator()(std::vector<char> &chars){

QString outChar = "";

for (std::vector<char>::const\_iterator i = chars.begin(); i != chars.end(); ++i)

outChar += QString(QChar::fromLatin1(\*i))+" ";

return outChar;

}

QString BitsetPrinter::operator()(std::vector<bitset<67>> &chars){

QString outBit = "";

for(unsigned int i=0;i<chars.size();i++)

outBit.append(QString::fromStdString(chars[i].to\_string()));

return outBit;

}

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 21 - Демонстрация работы приложения

## **Задание 9**

**Задание**: определите формат записи, содержащей ссылки на статьи и книги в файле. Разработайте программу, которая могла бы читать из файла записи по времени издания, имени автора, по ключевому слову в названии или по издательству.

Для выполнения задания создадим структуру книги, которая содержит поля для хранения названия, автора, издания и года издания. Создадим вектор таких структур (Рисунок 22).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22 - Структура Book

В окне приложения есть кнопка «Прочитать файл», нажимая которую вызывается функция чтения файла и записи структур в вектор (Рисунок 23). В самом файле на первой строке указывается число книг, далее на каждой строке записана информация об одной книге, которая разделяется запятыми.

В начале функции инициализируем необходимые переменные и открываем файл для чтения, далее проверяем удалось ли открыть файл и не пуст ли он, если все удачно, то считываем количество книг, а потом структуры в вектор. В конце закрываем файл и выводим прочитанные данные на экран пользователя (Рисунок 24).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 23 - Чтение данных из файла

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 24 - Чтение данных из файла

Также в окне приложения есть кнопки, соответствующие поиску книг по ключевому слову в названии, автору, времени издания и изданию, нажимая на которые получаем выборку данных по введенному условию (Рисунок 25, Рисунок 26, Рисунок 27, Рисунок 28, Рисунок 29, Рисунок 30).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25 - Поиск книг по дате или автору

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 26 - Работа поиска книг по времени издания

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 27 - Работа поиска книг по автору

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 28 - Поиск книг по ключевому слову в названии или издательству

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 29 - Работа поиска по ключевому слову в названии

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 30 - Работа поиска по издательству