|  |  |
| --- | --- |
| **Изображение выглядит как текст, эмблема, герб, нашивка  Автоматически созданное описание** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация (РК)

КАФЕДРА Системы автоматизированного проектирования (РК-6)

**ОТЧЕТ**

**по ознакомительной практике**

**модуль 1**

**по курсу «Объектно-ориентированное программирование»**

Выполнила:

Долженко Анастасия Тимофеевна

Группа:

РК6-32Б

Дата защиты:

«11» декабря 2023 г.

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Долженко А.Т.**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2023 г**.*

**Вариант 4**

Разработать шаблон класса-контейнера, в рамках которого реализовать итератор произвольного доступа в соответствии с вариантом задания (змейка, начиная с правого верхнего угла влево). Внутри контейнера элементы хранятся в виде прямоугольной матрицы с динамическим выделением памяти (число строк и столбцов передаются как параметры конструктора). Вывод контейнера на экран предполагает его матричное представление. Тестирование производить на контейнере элементов типа int, инициализировав их случайным образом в диапазоне [0;100). Проиллюстрировать возможность применения алгоритма sort из стандартной библиотеки шаблонов к разработанному контейнеру.

**Описание программы**

***Входные данные:*** -

***Выходные данные:*** вывод в стандартный поток вывода исходной и отсортированной матрицы

***Класс Container:***

*Информационные поля:*

* T \*\* values\_ - матрица типа, который мы зададим
* unsigned nrows\_, ncols\_ - кол-во строк, столбцов в матрице

*Методы:*

* Container<T>& operator=(const Container<T> &matrix) - перегрузка оператора присваивания
* T\* operator[](const int &i) - перегрузка оператора получения доступа
* const T\* operator[](const int &i)const) - перегрузка оператора получения доступа
* template <typename T1>
* friend std::ostream& operator<<(std::ostream &out, const Container<T1> &matrix) - перегрузка оператора побитового сдвига для вывода

*Внутренние классы:* class Iterator

***Класс Iterator:***

*Информационные поля:*

* T \*\* begin\_ - указатель на начало контейнера
* int curRow - индекс текущей строки в матрице, на которую указывает итератор
* int curColumn - индекс текущего столбца в матрице, на которую указывает итератор
* unsigned nRow – размер матрицы (количество строк)
* unsigned nCol – размер матрицы (количество столбцов)
* int len – текущая длина всей змейки
* int direction = -1 - флаг для обозначения направления движения итератора (изначально влево)

*Методы:*

* bool operator==(const Iterator &it) const – перегрузка оператора сравнения ==
* bool operator!=(const Iterator &it) const – перегрузка оператора сравнения !=
* bool operator<(const Iterator &it) const – перегрузка оператора сравнения <
* bool operator>(const Iterator &it) const – перегрузка оператора сравнения >
* bool operator<=(const Iterator &it) const – перегрузка оператора сравнения <=
* bool operator>=(const Iterator &it) const – перегрузка оператора сравнения >=
* T& operator \*() - перегрузка оператора вызова функции + разыменования
* const T& operator \*() const - перегрузка оператора вызова функции + разыменования
* Iterator& operator++() - перегрузка оператора префиксного инкремента
* Iterator& operator++(int) - перегрузка оператора постфиксного инкремента
* Iterator& operator--() - перегрузка оператора префиксного декремента
* Iterator& operator--(int) - перегрузка оператора постфиксного декремента
* Iterator operator+(difference\_type n) const - перегрузка оператора сложения
* Iterator operator+=(difference\_type n) const - перегрузка оператора +=
* Iterator operator-(difference\_type n) const - перегрузка оператора вычитание
* Iterator operator-=(difference\_type n) const - перегрузка оператора -=
* template <typename T1>

friend typename Container<T1>::Iterator& operator+(difference\_type n, const typename Container<T1>::Iterator &it) - перегрузка оператора сложения ( 5 + it)

* difference\_type operator-(const Iterator& it) const - перегрузка оператора - для вычитания итераторов
* T\* operator[](difference\_type n) - перегрузка оператора получения доступа
* const T\* operator[](difference\_type n) const - перегрузка оператора получения доступа
* template <typename T>

typename Container<T>::Iterator Container<T>::begin() T\* operator[](const int &i) – метод для установки итератора в начало контейнера

* template <typename T>

typename Container<T>::Iterator Container<T>::end() T\* operator[](const int &i) – метод для установки итератора в конец контейнера

**Описание алгоритма**

Создается класс контейнера элементов некоторого типа, имеющий матричное представление для пользователя. Для контейнера предусмотрен свой итератор, являющийся частью класса контейнера и реализующий проход по нему в соответствии с условием - змейка, начиная с правого верхнего угла влево.

Первым элементом контейнера считается правый верхний элемент. В начале обхода в соответствующие поля класса контейнера (curRow, curCol) передаются значения, соответствующие первой строке и последнему столбцу. Далее при проходе итератора по контейнеру поле curCol изменяется в соответствии с установленным на данный момент значением в поле direction класса контейнера. Если в поле direction содержится значение -1, то значение поля curCol уменьшается на единицу, что соответствует движению итератора влево. И наоборот, если в поле direction содержится значение 1, то значение поля curCol увеличивается на единицу, что соответствует движению итератора вправо.

При каждом сдвиге итератора в какую-либо сторону делается проверка на то, не вышел ли итератор за границы контейнера или за границы матрицы. Если нет выхода за границы, то увеличивается (уменьшается) значение поля len, в котором хранится текущая длина «змейки», и изменяются значения полей curCol или curRow.

Для недопущения выхода итератора за «левую» и «правую» границы контейнера используется поле len. Значение, содержащееся в этом поле, сравнивается с нулем или общим числом элементов в контейнере для проверки выхода итератора за «левую» и «правую» границы контейнера соответственно. Если итератор достиг начала или конца контейнера, то дальше он двигаться не будет.

Для недопущения выхода итератора за «правую» и «левую» границы матрицы используются поля curRow и curCol, показывающие текущую позицию итератора в матрице, и поля nRow и nCol, содержащие информацию о количестве строк и столбцов в матрице. После изменения значения, содержащегося в curCol в соответсвии с direction, оно сравнивается с числом столбцов в матрице. Если в curCol оказалось число меньше 0, то это говорит о том, что итератор «вышел» за левую границу матрицы, что означает, что на этой итерации змейка должна не двигаться в сторону, оставаясь на той же строке матрицы, что и прежде, а змейка должна остаться в том же столбце матрицы, что и прежде, и перейти на другую строку матрицы (строку ниже или выше, зависит от того, в какую сторону двигается итератор – к концу контейнера или к началу). Для реализации этого возвращается прежнее значение curCol и изменятся значение curRow в соответствии с направлением обхода.

Таким образом, при в любом алгоритме обход контейнера будет производиться в соответствии с заданными условиями.

**Код файла container.h**

#pragma once

#include <iostream>

#include <iterator>

#include <iomanip>

template <typename T>

class Container{

private:

T \*\* values\_; //матрица типа который мы зададим

unsigned nrows\_, ncols\_; // кол-во строк, столбцов

public:

// Конструктор

Container(unsigned rows, unsigned cols): nrows\_(rows), ncols\_(cols){

if (rows == 0 || cols == 0) throw std::invalid\_argument("You can't create array with size of 0");

values\_ = new T\*[nrows\_];

for(unsigned i = 0; i < nrows\_; i++){

values\_[i] = new T[ncols\_];

//for(unsigned j = 0; j < ncols\_; j++) \_values[i][j] = 0;

}

}

//Деструктор

~Container(){

for(unsigned i = 0; i < nrows\_; i++) delete [] values\_[i];

delete [] values\_;

}

//Конструктор копирования

Container(const Container<T> &matrix)

{

nrows\_ = matrix.nrows\_;

ncols\_ = matrix.ncols\_;

values\_ = new T\*[nrows\_];

for(unsigned i = 0; i < nrows\_; i++){

values\_[i] = new T[ncols\_];

for(unsigned j = 0; j < ncols\_; j++) values\_[i][j] = matrix.values\_[i][j];

}

}

//перегрузка оператора присваивания

Container<T>& operator=(const Container<T> &matrix){

if(this == &matrix) return \*this;

// если матрицы не совпадают, то очищаем имеющуюся и выделяем место под матрицу с новыми размерами

if(nrows\_ != matrix.nrows\_ || ncols\_ != matrix.ncols\_){

for(unsigned i = 0; i < nrows\_; i++) delete [] values\_[i];

delete [] values\_;

nrows\_ = matrix.nrows\_;

ncols\_ = matrix.ncols\_;

values\_ = new T\*[nrows\_];

for(unsigned i = 0; i < nrows\_; i++) values\_[i] = new T[ncols\_];

}

//копируем в старую матрицу новые значения

for(unsigned i = 0; i < nrows\_; i++){

for(unsigned j = 0; j < ncols\_; j++) values\_[i][j] = matrix.values\_[i][j];

}

return \*this;

}

//перегрузка оператора получения доступа

T\* operator[](const int &i){

if (i >= nrows\_) throw std::out\_of\_range("Error: index bigger than array size");

return values\_[i];

}

//перегрузка оператора получения доступа

const T\* operator[](const int &i)const {

if (i >= nrows\_) throw std::out\_of\_range("Error: index bigger than array size");

return values\_[i];

}

// перегрузка оператора побитового сдвига для вывода

template <typename T1>

friend std::ostream& operator<<(std::ostream &out, const Container<T1> &matrix){

for (unsigned i = 0; i < matrix.nrows\_; ++i){

out << "[ ";

for (unsigned j = 0; j < matrix.ncols\_; ++j){

out << std::setw(3) << matrix.values\_[i][j] << ' ';

}

out << ']' << '\n';

}

return out;

}

//объявляем итератор как часть классса контейнера

class Iterator;

Iterator begin();

Iterator end();

};

template<typename T>

class Container<T>::Iterator{

private:

//указатель на начало контейнера

T \*\* begin\_;

//Индекс текущей строки в матрице

int curRow;

//Индекс текущего столбца в матрице

int curColumn;

//Размеры матрицы: число строк и столбцов

unsigned nRow, nCol;

//Флаг для направления движения итератора (изначально влево)

int direction;

//Длина змейки

int len;

public:

using iterator\_category = std::random\_access\_iterator\_tag; // объявили тип итератора

using value\_type = T; // тип значения которое хранится и возвращается операторами \* и ->

using difference\_type = std::ptrdiff\_t; // тип который описывает растояние между итераторами

using pointer = value\_type\*; // тип указателя на значение

using reference = value\_type&; // тип ссылки на значение

//Конструктор

Iterator(T \*\*begin, int x, int y, unsigned nColl, unsigned nRoww, int \_len, int dir): \

begin\_(begin), curColumn(x), curRow(y), nCol(nColl), nRow(nRoww), len(\_len), direction(dir){}

//перегрузка операторов сравнения

bool operator==(const Iterator &it) const { return len == it.len;}

bool operator!=(const Iterator &it) const { return len != it.len;}

bool operator<(const Iterator &it) const { return len < it.len;}

bool operator>(const Iterator &it) const { return len > it.len;}

bool operator<=(const Iterator &it) const { return len <= it.len;}

bool operator>=(const Iterator &it) const { return len >= it.len;}

//перегрузка оператора вызова функции + разыменования

T& operator \*(){

std:: cout << std:: endl << curRow << ' ' << curColumn << std:: endl;

return \*( \*(begin\_ + curRow) + curColumn);

}

//перегрузка оператора вызова функции + разыменования

const T& operator \*() const{

std:: cout << std:: endl << curRow << ' ' << curColumn << std:: endl;

return \*( \*(begin\_ + curRow) + curColumn);

}

//перегрузка оператора префиксного инкремента ++it (перемещение по контейнеру)

Iterator& operator++(){

if(len == nRow \* nCol) return \*this; // если мы уже в конце, то не двигаемся

len++;

curColumn += direction; // двигаемся на клетку вперёд

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева и подвинули клетку обратно (вправо)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление движения (с левого на правое)

return \*this;

}

if(curColumn > nCol - 1){ // проверка на границу справа

curColumn--; // вышли за границу справа подвинули клетку в обратно (влево)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление движения (с правого на левое)

return \*this;

}

return \*this;

}

//перегрузка оператора постфиксного инкремента it++ (перемещение по контейнеру)

Iterator operator++(int){

Iterator temp = \*this;

if(len == nRow \* nCol) return temp; // если мы уже в конце, то не двигаемся

len++;

curColumn += direction; // двигаемся на клетку вперёд

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева и подвинули клетку обратно (вправо)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление (с левого на правое)

return temp;

}

if(curColumn > nCol - 1){ // проверка на границу справа

curColumn--; // вышли за границу справа подвинули клетку в обратно (влево)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление движения (с правого на левое)

return temp;

}

return temp;

}

//перегрузка оператора префиксного декремента --it (перемещение по контейнеру)

Iterator& operator--(){

if(len == 0) return \*this;

len--;

curColumn += (-1) \* direction; // двигаемся в обратном направлении

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева подвинули клетку обратно (вправо)

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с левого на правое)

return \*this;

}

if(curColumn > nCol - 1) // проверка на границу справа

{

curColumn--; // вышли за границу справа подвинули клетку в обратно

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с правого на левое)

return \*this;

}

return \*this;

}

//перегрузка оператора постфиксного декремента it-- (перемещение по контейнеру)

Iterator& operator--(int){

Iterator& temp = \*this;

if(len == 0) return temp;

len--;

curColumn += (-1) \* direction; // двигаемся в обратном направлении

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева подвинули клетку обратно (впрво)

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с левого на правое)

return temp;

}

if(curColumn > nCol - 1) // проверка на границу справо

{

curColumn--; // вышли за границу справа подвинули клетку в обратно (влево)

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с правого на левое)

return temp;

}

return temp;

}

// перегрузка оператора сложения (it + 5)

Iterator operator+(difference\_type n) const{

Iterator shiftedIt = \*this; // копируем итератор

for(difference\_type i = 0; i < n; i++){

if(shiftedIt.len == shiftedIt.nRow \* shiftedIt.nCol) return shiftedIt;

shiftedIt++;

}

return shiftedIt;

}

//перегрузка оператора += (it += 5)

Iterator& operator+=(difference\_type n){

for(difference\_type i = 0; i < n; i++){

if(len == nRow \* nCol) return \*this;

(\*this)++;

}

return \*this;

}

// перегрузка оператора вычитания (it - 5)

Iterator operator-(difference\_type n) const{

Iterator shiftedIt = \*this; // копируем итератор

for(difference\_type i = 0; i < n; i++){

if(shiftedIt.len == 0) return shiftedIt;

shiftedIt--;

}

return shiftedIt;

}

//перегрузка операции -= (it -= 5)

Iterator& operator-=(difference\_type n){

for(difference\_type i = 0; i < n; i++){

if(len == 0) return \*this;

(\*this)--;

}

return \*this;

}

// перегрузка оператора сложения ( 5 + it)

friend typename Container<T>::Iterator& operator+(difference\_type n, const typename Container<T>::Iterator &it){

typename Container<T>::Iterator shiftedId = it;

for(difference\_type i = 0; i < n; i++){

if(shiftedId.len == shiftedId.nRow \* shiftedId.nCol) return shiftedId;

shiftedId++;

}

return shiftedId;

}

//перегрузка оператора - для вычитания итераторов

difference\_type operator-(const Iterator& it) const{

//вычисляем длину между итераторами

return len - it.len;

}

//перегрузка оператора получения доступа (для строки, на которой находится итератор)

T\* operator[](difference\_type n){

if( (unsigned) n < nRow) return \*(begin\_ + n);

}

const T\* operator[](difference\_type n) const {

if( (unsigned) n < nrows\_) return \*(begin\_ + n);

}

};

template <typename T>

typename Container<T>::Iterator Container<T>::begin() {

return Container<T>::Iterator(values\_, ncols\_ - 1, 0, ncols\_, nrows\_, 0, -1);

}

template <typename T>

typename Container<T>::Iterator Container<T>::end(){

int col = nrows\_ % 2 == 0 ? ncols\_ - 1 : 0;

int dir = nrows\_ % 2 == 0 ? 1 : -1;

return Container<T>::Iterator(values\_, col, nrows\_-1, ncols\_, nrows\_, nrows\_ \* ncols\_, dir);

}#pragma once

#include <iostream>

#include <iterator>

#include <iomanip>

template <typename T>

class Container{

private:

T \*\* values\_; //матрица типа который мы зададим

unsigned nrows\_, ncols\_; // кол-во строк, столбцов

public:

// Конструктор

Container(unsigned rows, unsigned cols): nrows\_(rows), ncols\_(cols){

if (rows == 0 || cols == 0) throw std::invalid\_argument("You can't create array with size of 0");

values\_ = new T\*[nrows\_];

for(unsigned i = 0; i < nrows\_; i++){

values\_[i] = new T[ncols\_];

//for(unsigned j = 0; j < ncols\_; j++) \_values[i][j] = 0;

}

}

//Деструктор

~Container(){

for(unsigned i = 0; i < nrows\_; i++) delete [] values\_[i];

delete [] values\_;

}

//Конструктор копирования

Container(const Container<T> &matrix)

{

nrows\_ = matrix.nrows\_;

ncols\_ = matrix.ncols\_;

values\_ = new T\*[nrows\_];

for(unsigned i = 0; i < nrows\_; i++){

values\_[i] = new T[ncols\_];

for(unsigned j = 0; j < ncols\_; j++) values\_[i][j] = matrix.values\_[i][j];

}

}

//перегрузка оператора присваивания

Container<T>& operator=(const Container<T> &matrix){

if(this == &matrix) return \*this;

// если матрицы не совпадают, то очищаем имеющуюся и выделяем место под матрицу с новыми размерами

if(nrows\_ != matrix.nrows\_ || ncols\_ != matrix.ncols\_){

for(unsigned i = 0; i < nrows\_; i++) delete [] values\_[i];

delete [] values\_;

nrows\_ = matrix.nrows\_;

ncols\_ = matrix.ncols\_;

values\_ = new T\*[nrows\_];

for(unsigned i = 0; i < nrows\_; i++) values\_[i] = new T[ncols\_];

}

//копируем в старую матрицу новые значения

for(unsigned i = 0; i < nrows\_; i++){

for(unsigned j = 0; j < ncols\_; j++) values\_[i][j] = matrix.values\_[i][j];

}

return \*this;

}

//перегрузка оператора получения доступа

T\* operator[](const int &i){

if (i >= nrows\_) throw std::out\_of\_range("Error: index bigger than array size");

return values\_[i];

}

//перегрузка оператора получения доступа

const T\* operator[](const int &i)const {

if (i >= nrows\_) throw std::out\_of\_range("Error: index bigger than array size");

return values\_[i];

}

// перегрузка оператора побитового сдвига для вывода

template <typename T1>

friend std::ostream& operator<<(std::ostream &out, const Container<T1> &matrix){

for (unsigned i = 0; i < matrix.nrows\_; ++i){

out << "[ ";

for (unsigned j = 0; j < matrix.ncols\_; ++j){

out << std::setw(3) << matrix.values\_[i][j] << ' ';

}

out << ']' << '\n';

}

return out;

}

//объявляем итератор как часть классса контейнера

class Iterator;

Iterator begin();

Iterator end();

};

template<typename T>

class Container<T>::Iterator{

private:

//указатель на начало контейнера

T \*\* begin\_;

//Индекс текущей строки в матрице

int curRow;

//Индекс текущего столбца в матрице

int curColumn;

//Размеры матрицы: число строк и столбцов

unsigned nRow, nCol;

//Флаг для направления движения итератора (изначально влево)

int direction;

//Длина змейки

int len;

public:

using iterator\_category = std::random\_access\_iterator\_tag; // объявили тип итератора

using value\_type = T; // тип значения которое хранится и возвращается операторами \* и ->

using difference\_type = std::ptrdiff\_t; // тип который описывает растояние между итераторами

using pointer = value\_type\*; // тип указателя на значение

using reference = value\_type&; // тип ссылки на значение

//Конструктор

Iterator(T \*\*begin, int x, int y, unsigned nColl, unsigned nRoww, int \_len, int dir): \

begin\_(begin), curColumn(x), curRow(y), nCol(nColl), nRow(nRoww), len(\_len), direction(dir){}

//перегрузка операторов сравнения

bool operator==(const Iterator &it) const { return len == it.len;}

bool operator!=(const Iterator &it) const { return len != it.len;}

bool operator<(const Iterator &it) const { return len < it.len;}

bool operator>(const Iterator &it) const { return len > it.len;}

bool operator<=(const Iterator &it) const { return len <= it.len;}

bool operator>=(const Iterator &it) const { return len >= it.len;}

//перегрузка оператора вызова функции + разыменования

T& operator \*(){

std:: cout << std:: endl << curRow << ' ' << curColumn << std:: endl;

return \*( \*(begin\_ + curRow) + curColumn);

}

//перегрузка оператора вызова функции + разыменования

const T& operator \*() const{

std:: cout << std:: endl << curRow << ' ' << curColumn << std:: endl;

return \*( \*(begin\_ + curRow) + curColumn);

}

//перегрузка оператора префиксного инкремента ++it (перемещение по контейнеру)

Iterator& operator++(){

if(len == nRow \* nCol) return \*this; // если мы уже в конце, то не двигаемся

len++;

curColumn += direction; // двигаемся на клетку вперёд

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева и подвинули клетку обратно (вправо)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление движения (с левого на правое)

return \*this;

}

if(curColumn > nCol - 1){ // проверка на границу справа

curColumn--; // вышли за границу справа подвинули клетку в обратно (влево)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление движения (с правого на левое)

return \*this;

}

return \*this;

}

//перегрузка оператора постфиксного инкремента it++ (перемещение по контейнеру)

Iterator operator++(int){

Iterator temp = \*this;

if(len == nRow \* nCol) return temp; // если мы уже в конце, то не двигаемся

len++;

curColumn += direction; // двигаемся на клетку вперёд

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева и подвинули клетку обратно (вправо)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление (с левого на правое)

return temp;

}

if(curColumn > nCol - 1){ // проверка на границу справа

curColumn--; // вышли за границу справа подвинули клетку в обратно (влево)

curRow++; // спустились вниз

direction \*= -1; // поменяли направление движения (с правого на левое)

return temp;

}

return temp;

}

//перегрузка оператора префиксного декремента --it (перемещение по контейнеру)

Iterator& operator--(){

if(len == 0) return \*this;

len--;

curColumn += (-1) \* direction; // двигаемся в обратном направлении

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева подвинули клетку обратно (вправо)

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с левого на правое)

return \*this;

}

if(curColumn > nCol - 1) // проверка на границу справа

{

curColumn--; // вышли за границу справа подвинули клетку в обратно

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с правого на левое)

return \*this;

}

return \*this;

}

//перегрузка оператора постфиксного декремента it-- (перемещение по контейнеру)

Iterator& operator--(int){

Iterator& temp = \*this;

if(len == 0) return temp;

len--;

curColumn += (-1) \* direction; // двигаемся в обратном направлении

if(curColumn < 0){ // проверка на границу слева

curColumn++; // вышли за границу слева подвинули клетку обратно (впрво)

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с левого на правое)

return temp;

}

if(curColumn > nCol - 1) // проверка на границу справо

{

curColumn--; // вышли за границу справа подвинули клетку в обратно (влево)

curRow--; // поднялись вверх

direction \*= -1; // поменяли направление движения (с правого на левое)

return temp;

}

return temp;

}

// перегрузка оператора сложения (it + 5)

Iterator operator+(difference\_type n) const{

Iterator shiftedIt = \*this; // копируем итератор

for(difference\_type i = 0; i < n; i++){

if(shiftedIt.len == shiftedIt.nRow \* shiftedIt.nCol) return shiftedIt;

shiftedIt++;

}

return shiftedIt;

}

//перегрузка оператора += (it += 5)

Iterator& operator+=(difference\_type n){

for(difference\_type i = 0; i < n; i++){

if(len == nRow \* nCol) return \*this;

(\*this)++;

}

return \*this;

}

// перегрузка оператора вычитания (it - 5)

Iterator operator-(difference\_type n) const{

Iterator shiftedIt = \*this; // копируем итератор

for(difference\_type i = 0; i < n; i++){

if(shiftedIt.len == 0) return shiftedIt;

shiftedIt--;

}

return shiftedIt;

}

//перегрузка операции -= (it -= 5)

Iterator& operator-=(difference\_type n){

for(difference\_type i = 0; i < n; i++){

if(len == 0) return \*this;

(\*this)--;

}

return \*this;

}

// перегрузка оператора сложения ( 5 + it)

friend typename Container<T>::Iterator& operator+(difference\_type n, const typename Container<T>::Iterator &it){

typename Container<T>::Iterator shiftedId = it;

for(difference\_type i = 0; i < n; i++){

if(shiftedId.len == shiftedId.nRow \* shiftedId.nCol) return shiftedId;

shiftedId++;

}

return shiftedId;

}

//перегрузка оператора - для вычитания итераторов

difference\_type operator-(const Iterator& it) const{

//вычисляем длину между итераторами

return len - it.len;

}

//перегрузка оператора получения доступа (для строки, на которой находится итератор)

T\* operator[](difference\_type n){

if( (unsigned) n < nRow) return \*(begin\_ + n);

}

const T\* operator[](difference\_type n) const {

if( (unsigned) n < nrows\_) return \*(begin\_ + n);

}

};

template <typename T>

typename Container<T>::Iterator Container<T>::begin() {

return Container<T>::Iterator(values\_, ncols\_ - 1, 0, ncols\_, nrows\_, 0, -1);

}

template <typename T>

typename Container<T>::Iterator Container<T>::end(){

int col = nrows\_ % 2 == 0 ? ncols\_ - 1 : 0;

int dir = nrows\_ % 2 == 0 ? 1 : -1;

return Container<T>::Iterator(values\_, col, nrows\_-1, ncols\_, nrows\_, nrows\_ \* ncols\_, dir);

}

**Код файла main.cpp**

#include <algorithm>

#include <iostream>

#include <cstdlib>

#include "container.h"

#include <cstdlib>

#include <ctime>

// #include <fstream>

int main(){

std::srand(std::time(nullptr));

const unsigned nRows = 4, nCols = 3;

Container<int> matrix(nRows, nCols);

for (unsigned i = 0; i < nRows; ++i) {

for (unsigned j = 0; j < nCols; ++j)

matrix[i][j] = std::rand() % 100;

}

std::cout << "Исходная матрица: " << std::endl;

std::cout << matrix << std::endl;

// std::cout << \*(matrix.begin()) << std::endl;

// std::cout << \*(--matrix.end()) << std::endl;

std::sort(matrix.begin(), matrix.end());

std::cout << "Матрица после сортировки: " << std::endl;

std::cout << matrix << std::endl;

return 0;

}

**Результаты работы программы**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

**Список литературы**

1. Волосатова Т.М., Родионов С.В. Объектно-ориентированное программирование на С++. Режим доступа: <http://bigor.bmstu.ru/?cnt/?doc=VU/base.cou>
2. <https://en.cppreference.com/w/cpp/iterator>
3. https://en.cppreference.com/w/cpp/named\_req/RandomAccessIterator