**Министерство науки и высшего образования Российской Федерации**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский университет ИТМО**

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫЙ ТЕХНОЛОГИЙ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

**ЛАБОРАТОРНАЯ РАБОТА №5**

**По дисциплине «Программирование»**

**Лабораторная работа. STL. Контейнеры.**

Выполнил Кудашев И.Э

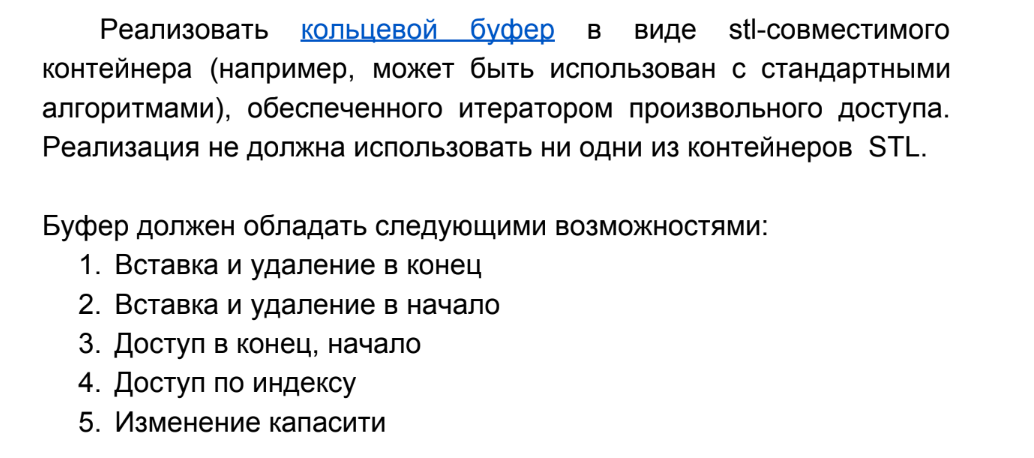
(Фамилия Имя Отчество)

Проверил Повышев В.В

(Фамилия Имя Отчество)

Санкт-Петербург, 2021г

условие лабораторной



код

#include <iostream>  
#include <algorithm>  
  
using namespace std;  
  
template<class T>  
class CircularBuffer {  
public:  
 class Iterator : public iterator<random\_access\_iterator\_tag, T> {  
 private:  
 T \*currentValue;  
 public:  
 explicit Iterator(T \*currentValue) {  
 this->currentValue = currentValue;  
 }  
  
 [[nodiscard]] T \*getCurent() const {  
 return currentValue;  
 }  
  
 //OPERATORS  
 Iterator operator+(int value) {  
 currentValue += value;  
 return \*this;  
 }  
  
 Iterator operator-(int value) {  
 currentValue -= value;  
 return \*this;  
 }  
  
 T &operator\*() const {  
 return \*currentValue;  
 }  
  
 T \*operator->() const {  
 return currentValue;  
 }  
  
 Iterator &operator++() {  
 ++currentValue;  
 return \*this;  
 }  
  
 Iterator operator--() {  
 currentValue--;  
 return \*this;  
 }  
  
 Iterator &operator=(T \*other) {  
 currentValue = other;  
 return \*this;  
 }  
  
 bool operator==(const Iterator &other) {  
 return this->currentValue = other.currentValue;  
 }  
  
 bool operator!=(const Iterator &other) {  
 return this->currentValue != other.currentValue;  
 }  
  
 bool operator>(const Iterator &other) {  
 return this->currentValue > other.currentValue;  
 }  
  
 bool operator>=(const Iterator &other) {  
 return this->currentValue >= other.currentValue;  
 }  
  
 bool operator<(const Iterator &other) {  
 return this->currentValue < other.currentValue;  
 }  
  
 bool operator<=(const Iterator &other) {  
 return this->currentValue <= other.currentValue;  
 }  
 };  
  
 CircularBuffer() = default;  
  
 explicit CircularBuffer(int size\_) {  
 this->capacity = 1.2 \* size\_;  
 this->size = size\_;  
 this->bufferData = new T[size\_];  
 this->bufferFirst = &bufferData[0];  
 this->bufferLast = &bufferData[size\_ - 1];  
 this->currentStart = this->currentEnd = &bufferData[0];  
 }  
  
 CircularBuffer(const CircularBuffer &other) {  
 this->size = other.size;  
 this->capacity = other.capacity;  
 this->bufferData = other.bufferData;  
 this->bufferFirst = other.bufferFirst;  
 this->bufferLast = other.bufferLast;  
 }  
  
  
 //NEEDED FUNCS  
 [[nodiscard]] int getSize() const {  
 return this->size;  
 }  
  
 [[nodiscard]] int getCapacity() const {  
 return this->capacity;  
 }  
  
  
 [[nodiscard]] Iterator start() {  
 return Iterator(bufferData);  
 }  
  
 [[nodiscard]] Iterator end() {  
 return Iterator(bufferData + size - 1);  
 }  
  
 void pushFront(const T &value) {  
 \*currentStart = value;  
 if (bufferFirst == currentStart) {  
 currentStart = bufferLast;  
 } else {  
 --currentStart;  
 }  
 }  
  
 void pushBack(const T &value) {  
 \*currentEnd = value;  
 if (bufferLast == currentEnd) {  
 currentEnd = bufferFirst;  
 } else {  
 currentEnd++;  
 }  
 }  
  
 void popFront() {  
 \*currentStart = 0;  
 if (currentStart == bufferLast) {  
 currentStart = bufferFirst;  
 } else {  
 ++currentStart;  
 }  
 }  
  
 void popBack() {  
 \*currentEnd = 0;  
 if (currentEnd == bufferFirst) {  
 currentEnd = bufferLast;  
 } else {  
 --currentEnd;  
 }  
 }  
  
 void setCapacity(int value) {  
 if (value > capacity) {  
 this->capacity = value;  
 } else {  
 throw invalid\_argument("Error");  
 }  
 }  
  
 T operator[](int index) const {  
 return bufferData[index % capacity];  
 }  
  
 ~CircularBuffer() {  
 delete[] bufferData;  
 }  
  
 void getInfo() {  
 for (auto i = &bufferData[0]; i != &bufferData[size]; i++) {  
 cout << \*i << endl;  
 }  
 }  
  
  
private:  
 int size = 0;  
 int capacity = 0;  
 T \*bufferData;  
 T \*bufferFirst; // указывает на 1-й элемент  
 T \*bufferLast; // указывает на последний эдемент  
 T \*currentStart; // указывает на текующий первый  
 T \*currentEnd; // указзывает на текущий последний  
  
};  
  
  
int main() {  
 CircularBuffer<int> Buffer(8);  
  
 auto iter = CircularBuffer<int>::Iterator(Buffer.start());  
  
  
 Buffer.pushFront(1);  
 Buffer.pushFront(2);  
 Buffer.pushFront(3);  
 Buffer.pushFront(4);  
  
 Buffer.pushBack(1.0);  
 Buffer.pushBack(2.0);  
 Buffer.pushBack(3.0);  
 Buffer.pushBack(4.0);  
 Buffer.pushBack(5.0);  
 Buffer.pushBack(6.0);  
 Buffer.pushBack(7.0);  
 Buffer.pushBack(8.0);  
 Buffer.pushBack(9.0);  
 Buffer.getInfo();  
 cout << endl;  
 cout << Buffer.getSize() << endl;  
 cout << Buffer.getCapacity() << endl;  
 Buffer.setCapacity(10);  
 cout << Buffer.getCapacity() << endl;  
  
  
 auto print = [](const int &n) { std::cout << " " << n; };  
 for\_each(Buffer.start(), Buffer.end()+1, print);  
  
}

Кудашев И.Э 