Министерство образования и науки РФ
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчёт по дисциплине «Алгоритмы и структуры данных»

Реализация структуры данных **Queue** (кольцевая очередь)

Работу выполнил:

студент группы 3530904/00006

Смирнов Е. А.

Работу принял:

преподаватель Павлов Е. А.

Санкт-Петербург
2021

Постановка задачи

1.Реализация структуры данных: *«ограниченная» кольцевая очередь*

2.*Тестирование* кольцевой очереди:

- добавление элементов
- обработка переполнения очереди
- исключение элементов
- обработка «опустошения»
- чередование добавления-исключения

3. *Итеративный* метод обхода двоичного дерева по уровням (в ширину). В реализации использовать класс очередь.

4. Написать метод, определяющий, являются ли два дерева похожими. *Похожими* будем называть *деревья* поиска, содержащие одинаковые наборы ключей.

Реализация:

Queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

template <class T>
class Queue
{
public:
	virtual ~Queue() = default;
	virtual void enQueue(const T& e) = 0; // Добавление элемента
	virtual const T& deQueue() = 0;// Удаление и возвращение верхнего элемента.
	virtual bool isEmpty() = 0; // Проверка на пустоту
	virtual void print() = 0;
};
#endif
```

QueueArray.cpp

```
#include "Queue.h"
#include "QueueOverflow.h"
#include "QueueUnderFlow.h"
#include "WrongQueueSize.h"

#include <iostream>

template <class T>
class QueueArray final : public Queue<T>
{
public:
	QueueArray(int size = 100);
	QueueArray(const QueueArray<T>& src);
	QueueArray(QueueArray<T>&& src) noexcept;
	virtual ~QueueArray();

	QueueArray<T>& operator= (const QueueArray<T>& src);
	QueueArray<T>& operator= (QueueArray<T>&& src) noexcept;

	void enQueue(const T& e);
	const T& deQueue();
	bool isEmpty();
	void print();
private:
	T* array_;
	int head_;
	int tail_;
	int size_;

	void swap(QueueArray<T>& right);
};

template <class T>
QueueArray<T>::QueueArray(int size) :
	head_(-1),
	tail_(-1),
	size_(size)
{
	try
	{
	   array_ = new T[size];
	}
	catch (...)
	{
```

```
                throw WrongQueueSize();
        }
}

template <class T>
QueueArray<T>::QueueArray(const QueueArray<T>& src) :
        head_(src.head_),
        tail_(src.tail_)
{
        try
        {
                array_ = new T[src.size_];
        }
        catch (...)
        {
                throw WrongQueueSize();
        }

        size_ = src.size_;
        for (int i = 0; i < size_; i++)
        {
                array_[i] = src.array_[i];
        }
}

template<class T>
QueueArray<T>::QueueArray(QueueArray<T>&& src) noexcept
{
        swap(src);
}

template<class T>
QueueArray<T>::~QueueArray()
{
        delete[] array_;
}

template<class T>
QueueArray<T>& QueueArray<T>::operator=(const QueueArray<T>& src)
{
        QueueArray<T> temp(src);
        swap(temp);
        return *this;
}

template<class T>
QueueArray<T>& QueueArray<T>::operator=(QueueArray<T>&& src) noexcept
{
        QueueArray<T> temp(std::move(src));
        swap(temp);
        return *this;
}

template <typename T>
void QueueArray<T>::swap(QueueArray<T>& right)
{
        std::swap(this->array_, right.array_);
        std::swap(this->head_, right.head_);
        std::swap(this->tail_, right.tail_);
        std::swap(this->size_, right.size_);
}

template<class T>
bool QueueArray<T>::isEmpty()
{
        return (head_ == tail_&& head_  == -1);
}

template <class T>
void QueueArray<T>::enQueue(const T& q)
{
```

```
	if ((head_ == 0 && tail_ == size_ - 1) || // очередь просто заполнена  - самый простой случай
		(tail_ == (head_ - 1) % (size_ - 1))) // хвост стоит перед головой
	{
		throw QueueOverflow();
	}
	else if (head_ == -1) // добавление в пустую очередь
	{
		head_ = tail_ = 0;
		array_[tail_] = q;
	}
	else if (tail_ == size_ - 1 && head_ != 0) // если мы что-то удалили и хвост подходит к 0 "индексу"
	{
		tail_ = 0;
		array_[tail_] = q;
	}
	else
	{
		tail_++;
		array_[tail_] = q;
	}
}

template <class T>
const T& QueueArray<T>::deQueue()
{
	if (head_ == -1) throw QueueUnderFlow();

	T& data = array_[head_];
	array_[head_] = -1;
	if (head_ == tail_) // 1 элемент добавлен и сразу его удаляем
	{
		head_ = -1;
		tail_ = -1;
	}
	else if (head_ == size_ - 1) head_ = 0;
	else head_++;

	return data;
}

template <class T>
void QueueArray<T>::print()
{
	if (head_ == -1)
	{
		std::cout << "Queue is Empty!";
	}
	else if (tail_ >= head_)
	{
		for (int i = head_; i <= tail_; i++)
		std::cout << array_[i] << " ";
	}
	else
	{
		for (int i = head_; i < size_; i++)
			std::cout << array_[i] << " ";

		for (int i = 0; i <= tail_; i++)
			std::cout << array_[i] << " ";
	}

	std::cout << std::endl;
}
```

Для исключительных ситуаций при работе с очередью реализованы следующие классы:

QueueOverFlow.h

```
#ifndef OVERFLOW_H
#define OVERFLOW_H

#include <exception>
#include <string>

class QueueOverflow : public std::exception
{
public:
	QueueOverflow() : reason_("QueueOverflow") {}
	const char* what() const { return reason_; }
private:
	const char* reason_;
};
#endif
```

QueueUnderFlow.h

```
#ifndef UNDERFLOW_H
#define UNDERFLOW_H

#include<exception>
#include <string>

class QueueUnderFlow : public std::exception
{
public:
	QueueUnderFlow() : reason_("QueueUnderFlow") {}
	const char* what() const { return reason_; }
private:
	const char* reason_;
};
#endif
```

WrongQueueSize.h

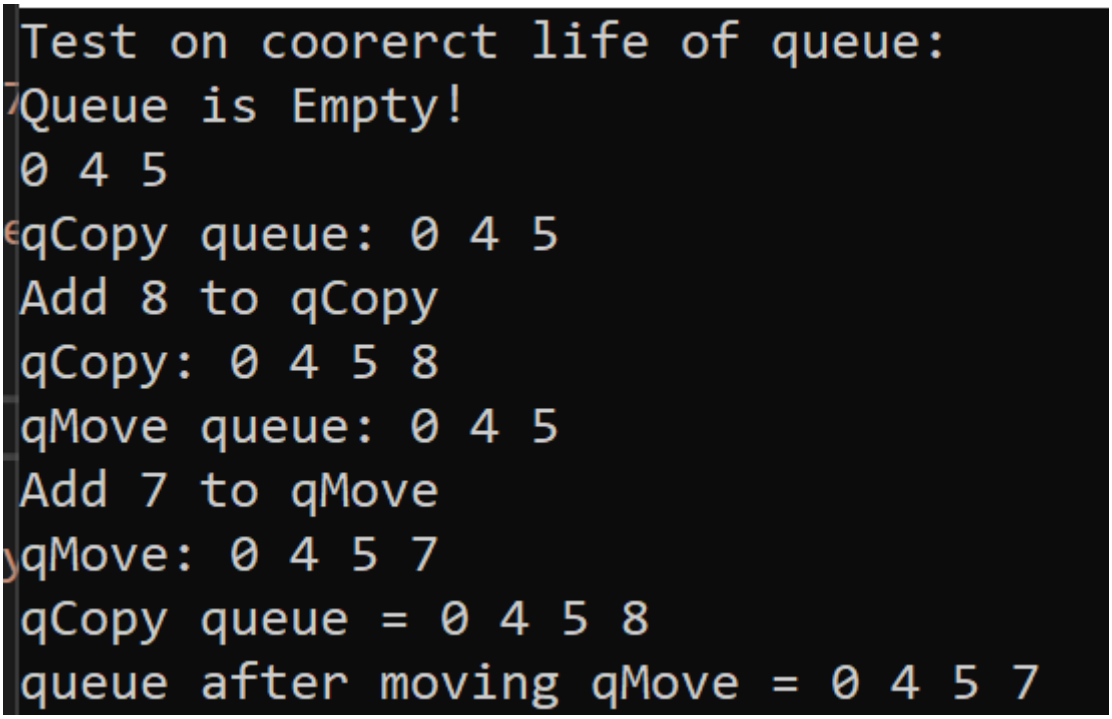
```
#ifndef WRONG_SIZE_H
#define WRONG_SIZE_H

#include <exception>
#include <string>

class WrongQueueSize : public std::exception
{
public:
	WrongQueueSize() : reason_("WrongQueueSize") {}
	const char* what() const { return reason_; }
private:
	const char* reason_;
};
#endif
```

Тестовая функция
Main.cpp

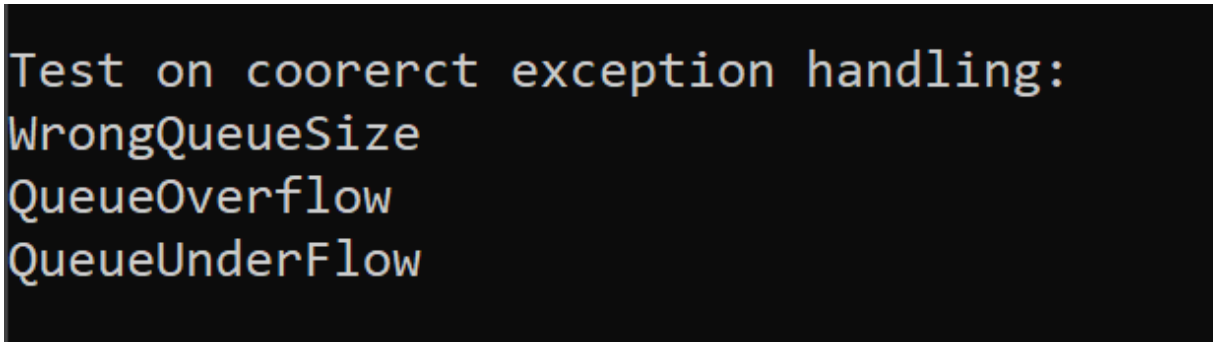
```
#include "QueueArray.h"
#include <iostream>

void DeafaultFunction();
void ErrorsCatch();

int main()
{
 DeafaultFunction();
 std::cout << "\n";
 ErrorsCatch();
 return 0;
}

void ErrorsCatch()
{
 std::cout << "Test on coorerct exception handling:\n";
 try
 {
  QueueArray<int> q(-5);
 }
 catch (const std::exception& e)
 {
  std::cout << e.what() << std::endl;
 }

 QueueArray<int> q(5);

 q.enQueue(1);
 q.enQueue(2);
 q.enQueue(3);
 q.enQueue(4);
 q.enQueue(5);

 try
 {
  q.enQueue(6);
 }
 catch (const std::exception& e)
 {
  std::cout << e.what() << std::endl;
 }
 q.deQueue();
 q.deQueue();
 q.deQueue();
 q.deQueue();
 q.deQueue();
 try
 {
  q.deQueue();
 }
 catch (const std::exception& e)
 {
  std::cout << e.what() << std::endl;
 }
}
```

```
void DeafaultFunction()
{
 std::cout << "Test on coorerct life of queue:\n";
 QueueArray<int> queue(4);
 queue.print();
 queue.enQueue(1004);
 queue.enQueue(1005);
 queue.enQueue(9);
 queue.enQueue(0);
 queue.deQueue();
 queue.deQueue();

 queue.enQueue(4);
 queue.enQueue(5);
 queue.deQueue();

 queue.print();

 // копирование
 QueueArray<int> qCopy(queue);
 std::cout << "qCopy queue: ";
 qCopy.print();
 qCopy.enQueue(8);
 std::cout << "Add 8 to qCopy" << std::endl;
 std::cout << "qCopy: "; qCopy.print();

 // перемещение
 QueueArray<int> qMove(std::move(queue));
 std::cout << "qMove queue: ";
 qMove.print();
 std::cout << "Add 7 to qMove" << std::endl;
 qMove.enQueue(7);
 std::cout << "qMove: ";
 qMove.print();

 // = копирования
 queue = qCopy;
 std::cout << "qCopy queue = ";
 qCopy.print();

 // = перемещения
 queue = std::move(qMove);
 std::cout << "queue after moving qMove = ";
 queue.print();
}
```

DeafaultFunction результаты



ErrorsCatch результаты



К классу BinarySearchTree добавлены **2 метода**

1. метод обхода дерева в ширину с помощью queue

```
template<class T>
void BinarySearchTree<T>::acrossWideWalk() const
{
	std::queue<Node<T>*> q;
	q.push(root_);
	while (!(q.empty()))
	{
		Node<T>* temp = q.front();
		q.pop();
		if (temp->left_ != nullptr)
		{
			q.push(temp->left_);
		}
		if (temp->right_!= nullptr)
		{
			q.push(temp->right_);
		}
		std::cout << temp->key_ << " ";
	}
}
```

2. . метод определяющий похожи ли деревья?

Деревья – похожи, если они имеют одинаковый набор ключей

```
template<class T>
bool BinarySearchTree<T>::areSimilar(const BinarySearchTree& rhs) const
{
	if (this->getHeight() == -1 && rhs.getHeight() == -1) return true;
	else if (this->getHeight() == -1) return false;

	std::queue<Node<T>*> q;
	q.push(root_);
	while (!(q.empty()))
	{
		Node<T>* top = q.front();
		q.pop();
		if (top->left_ != nullptr)
		{
			q.push(top->left_);
		}
		if (top->right_ != nullptr)
		{
			q.push(top->right_);
		}
		if (!(rhs.iterativeSearch(top->key_)))
		{
			return false;
		}
	}
	return true;
}
```

Тестирование:

```
BinarySearchTree<int> tree;

tree.insert(6);
tree.insert(1);
tree.insert(2);
tree.insert(10);
tree.insert(8);
tree.insert(7);
tree.insert(14);
tree.insert(16);
tree.insert(15);
tree.deleteKey(14);

std::cout << "В ширину обход дерева ";
tree.acrossWideWalk();


BinarySearchTree<int> tree1;
tree1.insert(1);
tree1.insert(2);
tree1.insert(3);

BinarySearchTree<int> tree2;
tree2.insert(3);
tree2.insert(2);
tree2.insert(1);

std::cout << "\n" << "Похожи ли деревья ?: ";
std::cout << tree1.areSimilar(tree2) << "\n";


BinarySearchTree<int> tree3;

BinarySearchTree<int> tree4;
tree4.insert(3);
tree4.insert(2);
tree4.insert(1);

std::cout << "Похожи ли деревья ?: ";
std::cout << tree3.areSimilar(tree4) << "\n";
```