ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Департамент прикладной математики

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ 1
по дисциплине «Алгоритмизация и программирование»

Работу выполнила
студентка группы БПМ 173 _____ М.В. Самоделкина
дата, подпись

Работу проверил _____ С.А. Булгаков
дата, подпись

Москва 2019

Содержание

Постановка задачи **3**

1 Основная часть **4**

1.1 Общая идея решения задачи 4

1.2 Структура и принципы действия 4

1.3 Процедура получения исполняемых программных модулей 5

1.4 Результаты тестирования 6

Приложение А **7**

List.h 7

Queue.h 9

Node.h 11

Iterator.h 11

ContainerIterator.h 13

Постановка задачи

Используя механизм шаблонных классов (*template*) написать классы для динамических структур данных и итераторов для их обхода в соответствии с вариантом 13:

1. Односвязный список - *ForwardIterator*
2. Очередь - *ForwardIterator*

1 Основная часть

1.1 Общая идея решения задачи

Для решения задачи были созданы шаблонные классы *List* и *Queue* с помощью агрегирования классов итераторов (*Iterator* и *ConstIterator*), классы итераторов в свою очередь содержат структуру узла *Node*.

1.2 Структура и принципы действия

Классы *List* и *Queue*, *Iterator* и *ConstIterator* содержат в себе структуру узла (*Node*), который состоит из элемента односвязного списка или очереди и указателя на следующий элемент.

Класс *List* содержит в себе указатель на начало списка (*Node**) и размер, определены конструктор умолчания, конструктор копирования, деструктор, а также такие стандартные методы, как *begin* (возвращает указатель на первый элемент), *len* (возвращает размер списка), *push* (добавляет элемент в начало списка), *pop* (удаляет элемент из начала списка и возвращает его).

Класс *Queue* содержит в себе указатель на начало очереди и конец очереди (*Node**) и размер, определены конструктор умолчания, конструктор копирования, деструктор, а также такие стандартные методы, как *begin* (возвращает указатель на первый элемент),*end* (возвращает указатель на последний элемент), *len* (возвращает размер очереди), *push* (добавляет элемент в конец очереди), *pop* (удаляет элемент из начала очереди и возвращает его).

Каждый класс содержит классы *Iterator* и *ConstIterator*, которые содержат указатель на элемент структуры *Node*, конструктор с параметром, для которых определены операторы ++, *, ==, *!=*.

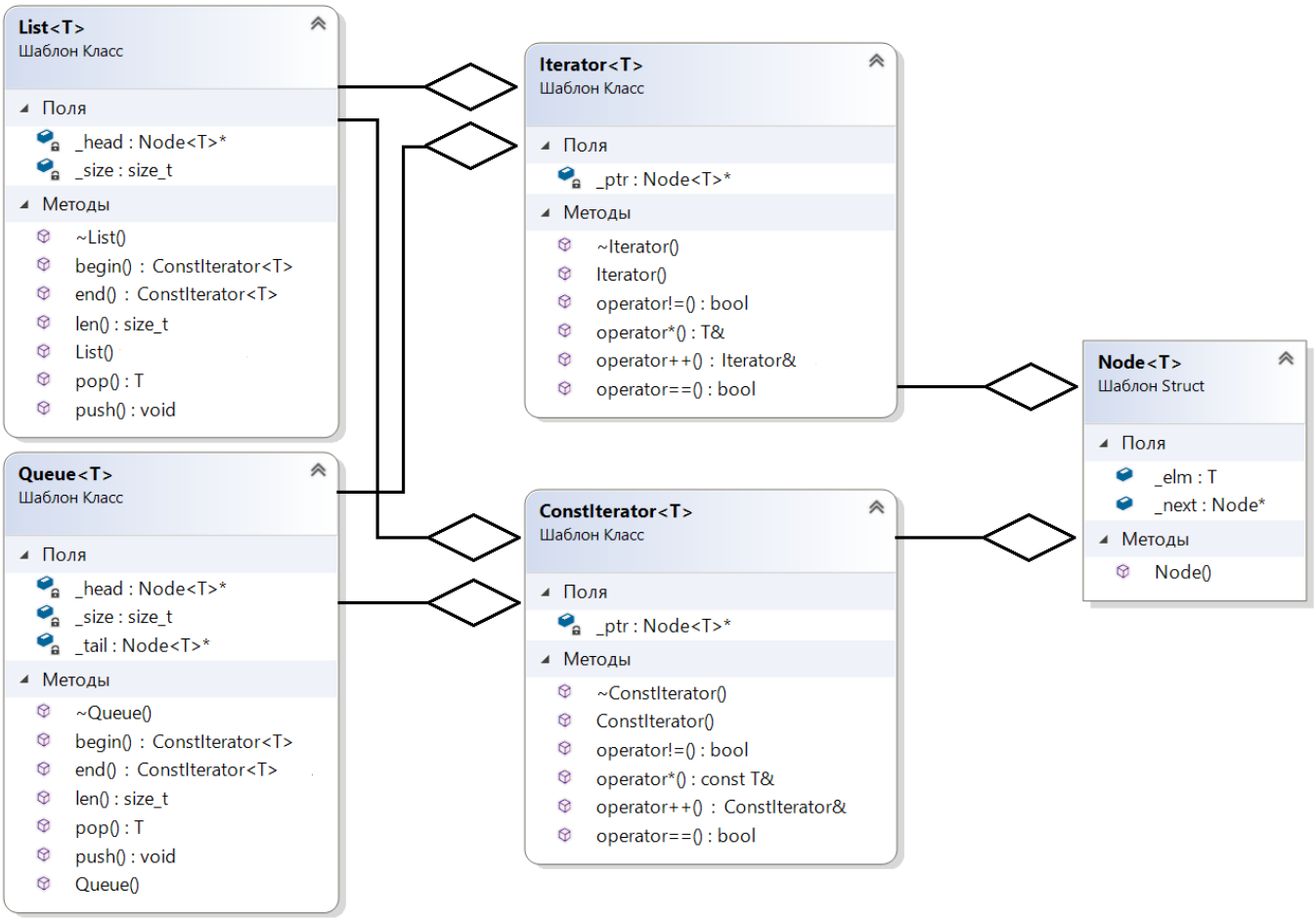

Рис. 1: Диграма классов *List* и *Queue*

Также были реализованы шаблонные функции, перегружающие оператор вывода в поток для отображения коллекций *List* и *Queue* на экране.

1.3 Процедура получения исполняемых программных модулей

Программный код был скомпилирован с среде *Visual Studio 2017*. Компиляция раздельная: исходный код программы разделён на несколько файлов. Для ускоренной компиляции программы используются предварительно откомпилированные заголовки *"pch.h"*. Помимо этого никаких дополнительных ключей не добавлялось, использовались ключи, которые добавляются по умолчанию. Параметры командной строки: /c /ZI /JMC /nologo /W3 /WX- /diagnostics:classic /sdl /Od /Oy- /D WIN32 /D _DEBUG /D _CONSOLE /D _UNICODE /D UNICODE /Gm- /EHsc /RTC1 /MDd /GS /fp:precise /permissive- /Zc:wchar_t /Zc:forScope /Zc:inline /Yc"pch.h"/Fp"DEBUG \ CONTAINERITERATOR.PCH"/Fo"DEBUG \ "/Fd"DEBUG \ VC141.PDB"/Gd /TP /analyze- /FC C: \ USERS \ PRO18 \ SOURCE \ REPOS \ CONTAINERITERATOR \ CONTAINERITERATOR \ PCH.CPP

1.4 Результаты тестирования

Тестирование программы представлено в файле *”ContainerIterator.cpp”* в функции *Main()*. Ожидаемый вывод функции:

20 19 1
1
20 19 1
1
1.7 0.333
0.333
1

Приложение А

полный код программы

A.1 - List.h

```
#pragma once
#include <iostream>
#include "Iterator.h"

template <typename T>
class List
{
private:
        Node<T>* _head;
        size_t _size;
public:
        List() {
                _head = nullptr;
                _size = 0;
        }
        List(const List&list) {
                _head = nullptr;
                _size = 0;
                Node<T>* next(nullptr);
                for (auto i = list._head; i != nullptr;
                        i = i->_next) {
                        if (!_size) {
                                _head = new
                                        Node<T>(T(i->_elm));
                                next = _head;
                                _size++;
                        }
                        else{
                                Node<T>* node = new
                                        Node<T>(T(i->_elm));
```

```
                        next->_next = node;
                        next = node;
                        _size++;
                    }
                }
            }
            ~List() {
                while (_head != nullptr) {
                    Node<T>* old_head = _head;
                    _head = _head->_next;
                    delete old_head;
                }
                _size = 0;
            }
            void push(const T & elm) {
                Node<T>* node = new Node<T>(elm);
                node->_next = _head;
                _head = node;
                _size++;
            }
            T pop() {
                if (_size) {
                    Node<T>* node = _head->_next;
                    T elm = _head->_elm;
                    delete _head;
                    _head = node;
                    _size--;
                    return elm;
                }
            }
            Iterator<T> begin() {
                return Iterator<T>(_head);
            }
            Iterator<T> end() {
```

```
                return Iterator<T>(nullptr);
        }
        ConstIterator<T> begin() const {
                return ConstIterator<T>(_head);
        }
        ConstIterator<T> end() const {
                return ConstIterator<T>(nullptr);
        }
        size_t len() {
                return _size;
        }
};
```

A.2 - Queue.h

```
#pragma once
#include <iostream>
#include "Iterator.h"

template <typename T>
class Queue
{
private:
        Node<T> *_head, *_tail;
        size_t _size;
public:
        Queue() {
                _head = _tail = nullptr;
                _size = 0;
        }
        Queue(const Queue&queue) {
                _head = _tail = nullptr;
                _size = 0;
                Node<T>* next(nullptr);
                for (auto i = queue._tail;
                        i != nullptr; i = i->_next) {
```

```
                push(T(i->_elm));
        }
    }
    ~Queue() {
        while (_head != nullptr) {
            Node<T>* old_head = _head;
            _head = _head->_next;
            delete old_head;
        }
        _size = 0;
    }
    void push(const T & elm) {
        if (_tail == nullptr) {
            _tail = _head = new Node<T>(elm);
            _size++;
        }
        else {
            Node<T>* node = new Node<T>(elm);
            _head->_next = node;
            _head = node;
            _size++;
        }
    }
    T pop() {
        if (_size) {
            Node<T>* node = _tail->_next;
            T elm = _tail->_elm;
            delete _tail;
            _tail = node;
            _size--;
            return elm;
        }
    }
    Iterator<T> begin() {
```

```
                return Iterator<T>(_tail);
        }
        Iterator<T> end() {
                return Iterator<T>(nullptr);
        }
        ConstIterator<T> begin() const {
                return ConstIterator<T>(_tail);
        }
        ConstIterator<T> end() const {
                return ConstIterator<T>(nullptr);
        }
        size_t len() {
                return _size;
        }
};
```

A.3 - Node.h

```
#pragma once
template <typename T>
struct Node {
        T _elm;
        Node* _next;
        Node(const T & elm) : _elm(elm), _next(nullptr) {}
};
```

A.4 - Iterator.h

```
#pragma once
#include "Node.h"

template <typename T>
class Iterator {
private:
        Node<T>* _ptr;
public:
        Iterator(Node<T>* ptr) : _ptr(ptr) {}
```

```
        ~Iterator() {}
        Iterator& operator++() {
                _ptr = _ptr->_next;
                return *this;
        }
        Iterator operator++(int) {
                Iterator i = *this;
                _ptr = _ptr->_next;
                return i;
        }
        T& operator*() const {
                return _ptr->_elm;
        }
        bool operator==(const Iterator& rhs) const {
                return _ptr == rhs._ptr;
        }
        bool operator!=(const Iterator& rhs) const {
                return !(_ptr == rhs._ptr);
        }
};

template <typename T>
class ConstIterator {
private:
        Node<T>* _ptr;
public:
        ConstIterator(Node<T>* ptr) : _ptr(ptr) {}
        ~ConstIterator() {}
        ConstIterator& operator++() {
                _ptr = _ptr->_next;
                return *this;
        }
        ConstIterator operator++(int) {
                ConstIterator i = *this;
```

```
            _ptr = _ptr->_next;
            return i;
        }
        const T& operator*() const {
            return _ptr->_elm;
        }
        bool operator==(const ConstIterator& rhs) const {
            return _ptr == rhs._ptr;
        }
        bool operator!=(const ConstIterator& rhs) const {
            return !(_ptr == rhs._ptr);
        }
};
```

A.5 - ContainerIterator.cpp

```
#include "pch.h"
#include "List.h"
#include "Queue.h"

template<typename T >
std::ostream & operator<<(std::ostream & os,
        const List <T> & list) {
        for (auto i = list.begin();
            i != list.end(); ++i)
        {
            os << *i;
            os << "␣";
        }
        os << std::endl;
        return os;
}

template<typename T >
std::ostream & operator<<(std::ostream & os,
        const Queue <T> & list) {
```

```
        for (auto i = list.begin();
                i != list.end(); ++i)
        {
                os << *i;
                os << "␣";
        }
        os << std::endl;
        return os;
}

int main()
{
        List<int> list;
        list.push(1);
        list.push(19);
        list.push(20);
        List<int> list2(list);
        std::cout << list;
        list.pop();
        list.pop();
        std::cout << list;
        std::cout << list2;
        std::cout << list.len() << std::endl;
        Queue<double> queue;
        queue.push(1.7);
        queue.push(0.333);
        Queue<double> queue2(queue);
        std::cout << queue;
        queue2.pop();
        std::cout << queue2;
        std::cout << queue2.len();
        return 0;
}
```