Правительство Российской Федерации

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

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине

«Языки программирования»

Работу выполнил

студент группы СКБ-203 А.Р. Фахретдинов

подпись, дата

Работу проверил С.А. Булгаков

подпись, дата

Москва 2021

Содержание

Постановка задачи

Разработать программу на языке Си++ (ISO/IEC 14882:2014), демонстрирующую решение поставленной задачи.

Общая часть

Разработать класс ADT и унаследовать от него классы, разработанные в рамках лабораторной работы 1.

Разработать набор классов, объекты которых реализуют типы данных, указанные ниже. Для этих классов разработать необходимые конструкторы, деструктор, конструктор копирования. Разработать операции: добавления/удаления элемента (уточнено в задаче); получения количества элементов; доступа к элементу (перегрузить оператор []). При ошибках запускать исключение.

В главной функции разместить тесты, разработанные с использованием библиотеки GoogleTest.

Задачи

1. Разработать класс ADT и унаследовать от него классы, разработанные в рамках лабораторной работы 1.
2. Динамический массив указателей на объекты ADT. Размерность массива указателей увеличивается в момент его переполнения. Начальная размерность задается как параметр конструктора, значение по умолчанию 0. Добавление/удаление элемента в произвольное место.
3. Стек, представленный динамическим массивом указателей на хранимые объекты ADT. Размерность стека увеличивается в момент его переполнения. Начальная размерность задается как параметр конструктора, значение по умолчанию 0.Добавление/удаление элемента в начало и в конец.
4. Односвязный список, содержащий указатели на объекты ADT. Добавление/удаление элемента в произвольное место.
5. Циклическая очередь, представленная динамическим массивом указателей на хранимые объекты ADT. Добавление/удаление элемента в произольное место.
6. Двоичное дерево, содержащее указатели на объекты ADT. Добавление/удаление элемента в произвольное место.

1 Алгоритм решения задачи

1.1 Задача №1
Для решения задачи был разработан класс ADT, содержащий виртуальный деструктор и метод print, для их перегрузки в классах наследниках.

1.2 Задача №2
Для решения задачи был разработан класс типа «контейнер», хранящий в себе динамический массив указателей на объекты типа ADT. Был разработан конструктор по умолчанию, задающий размерность по умолчанию, равной 0. Так же были созданы методы для работы с объектами класса, в частности метод insert для добавления элемента в произвольное место массива и метод erase для удаления элемента из произвольного места массива. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок.

1.3 Задача №3
Для решения задачи был разработан класс типа «контейнер», хранящий в себе динамический массив указателей на объекты типа ADT. Был разработан конструктор по умолчанию, задающий размерность по умолчанию, равной 0. По-мимо этого были созданы методы для работы с объектами класса, в частности методы push_back, push_front, pop_back, pop_front для добавления элемента в конец/начало массива и удаления элемента в конец/начало массива соответственно. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок.

1.4 Задача №4
Для решения задачи было разработано два класса: класс Node — узел, хранящий адреса следующего узла и указатель на объект типа ADT и класс типа «контейнер», хранящий в себе указатели на начальный и конечные узлы. Так же были созданы методы для работы с объектами класса, в частности метод insert для добавления элемента в произвольное место массива и метод erase для удаления элемента из произвольного места массива. Для доступа к указателям на объекты типа Node был перегружен оператор квадратных скобок.

1.5 Задача №5
Для решения задачи был разработан класс типа «контейнер», хранящий в себе динамический массив указателей на объекты типа ADT и при переполнении, циклично перезаписывающий данные. Таким образом в массиве выполняется принцип FIFO — first in first out. Так же были созданы методы для работы с объектами класса — метод insert для добавления элемента в произвольное место массива и метод erase для удаления элемента из произвольного места массива. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок.

1.6 Задача №6
Для решения задачи было разработано два класса: класс Tnode — лист, хранящий указатели на левый и правый лист дерева, родитель и объект типа ADT и класс типа «контейнер», хранящий указатель tree на корень дерева типа Tnode. Так же были созданы методы для работы с объектами класса, в частности метод insert для добавления элемента в произвольное место дерева и метод erase для удаления элемента из произвольного места дерева. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок.

2 Выполнение задания

2.1 Задача №1

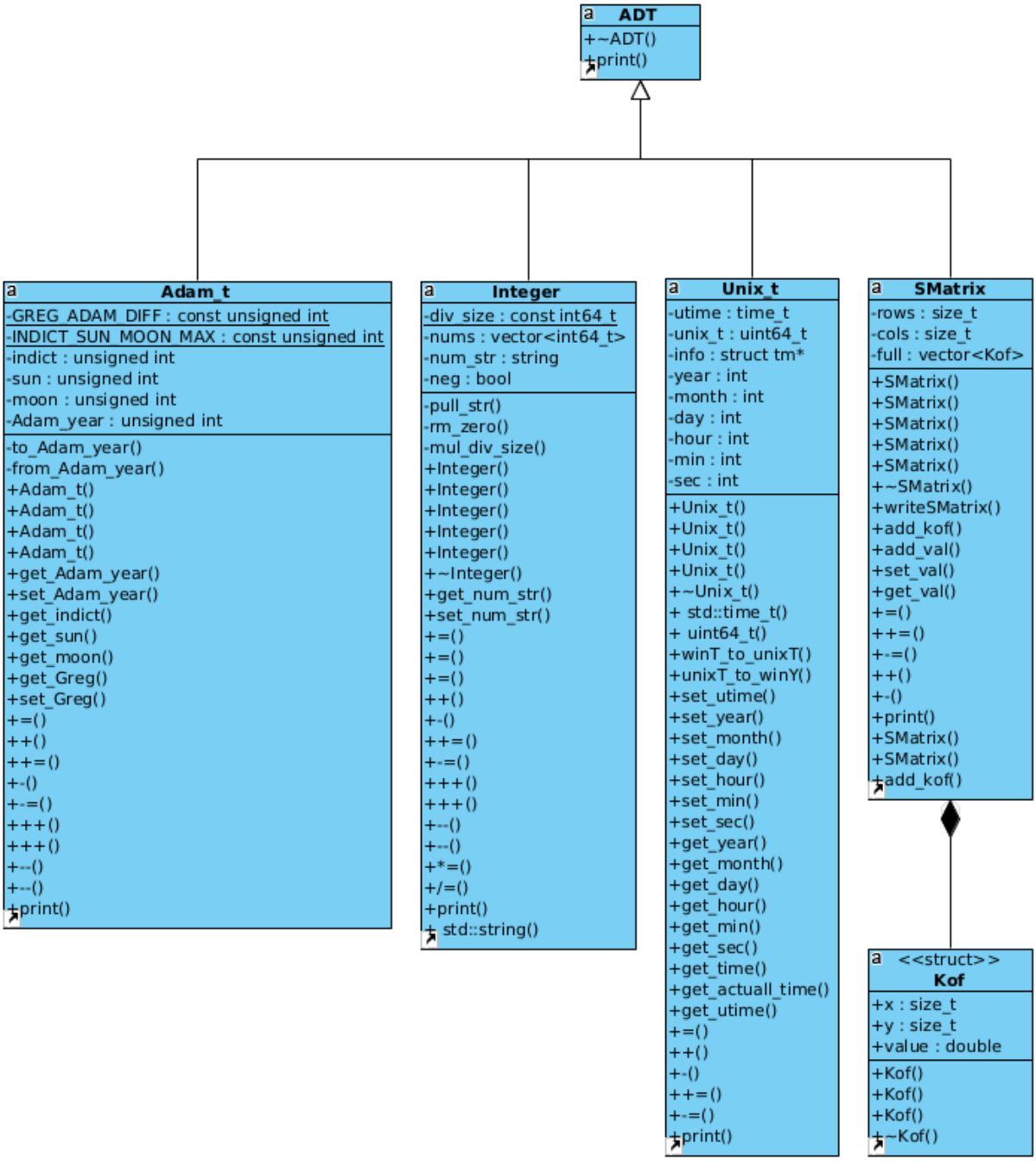

Рис.1. UML диаграмма класса ADT

Класс написан на языке C++. Код класса размещается в заголовочном файле ADT.h, заголовочный файл содержит защиту от повторного включения. Реализован виртуальный деструктор и метод print, для их перегрузки в классах наследниках. Так же перегружен оператор вывода в поток дружественным методом.

2.2 Задача №2

Класс написан на языке C++. Код класса размещается в одной единицах трансляции – код в файле dycont.cpp, прототип класса в заголовочном файле dycont.h. Тесты в виде проверки результатов и постусловий с помощью утверждений выполнены в единице трансляции main_dycont.cpp с функцией main, при этом, заголовочный файл содержит защиту от повторного включения.

Реализация класса включает в себя закрытые поля **_size**, **_capacity** типа size_t и **arr** типа указатель на указатель на объект типа ADT, где **_size** – количество элементов в массиве, а **_capacity** — реальная выделенная память под массив, для оптимизации количества выделений памяти при работе с массивом. Конструктор по умолчанию задаёт значения **_size** и **_capacity** равными 0. Конструктор общего вида принимает число типа size_t — задающее, размер массива _size и выделяющее количество памяти равное (**_size** * 3) / 2 + 1 — **_capacity**.

Реализованы методы для взаимодействия с массивом — **push_back** и **pop_back**, для вставки и извлечения последнего элемента, **insert** — для добавления элемента в произвольное место массива, метод **erase** для удаления элемента из произвольного места массива, метод **empalce** для замены произвольного элмента. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок. Для доступа к закрытым полям **_size** и **_capacity** были реализованы акцессоры. Методы **begin** и **end** возвращают указатель на начало и конец массива соответственно. Метод empty возвращает логическое значение соответствующие пустоте массива.

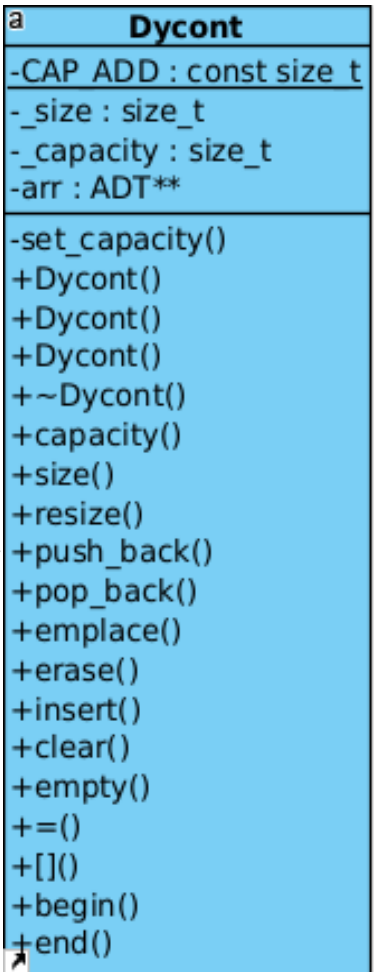

Рис.2 UML диаграмма класса Dycont

2.3 Задача №3

Класс написан на языке C++. Код класса размещается в одной единицах трансляции – код в файле stack.cpp, прототип класса в заголовочном файле stack.h. Тесты в виде проверки результатов и постусловий с помощью утверждений выполнены в единице трансляции main_stack.cpp с функцией main, при этом, заголовочный файл содержит защиту от повторного включения.

Реализация класса включает в себя закрытые поля **_size**, **_capacity** типа size_t и **arr** типа указатель на указатель на объект типа ADT, где **_size** – количество элементов в массиве, а **_capacity** — реальная выделенная память под массив, для оптимизации количества выделений памяти при работе с массивом. Конструктор по умолчанию задаёт значения **_size** и **_capacity** равными 0. Конструктор общего вида принимает число типа size_t — задающее, размер массива **_size** и выделяющее количество памяти равное (**_size** * 3) / 2 + 1 — **_capacity**.

Реализованы методы для взаимодействия с массивом — **push_back, push_front, pop_back, pop_front** для добавления элемента в конец/начало массива и удаления элемента в конец/начало массива соответственно., **insert** — для добавления элемента в произвольное место массива, метод **erase** для удаления элемента из произвольного места массива, метод **empalce** для замены произвольного элмента. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок. Для доступа к закрытым полям **_size** и **_capacity** были реализованы акцессоры. Методы **begin** и **end** возвращают указатель на начало и конец массива соответственно. Метод empty возвращает логическое значение соответствующие пустоте массива.

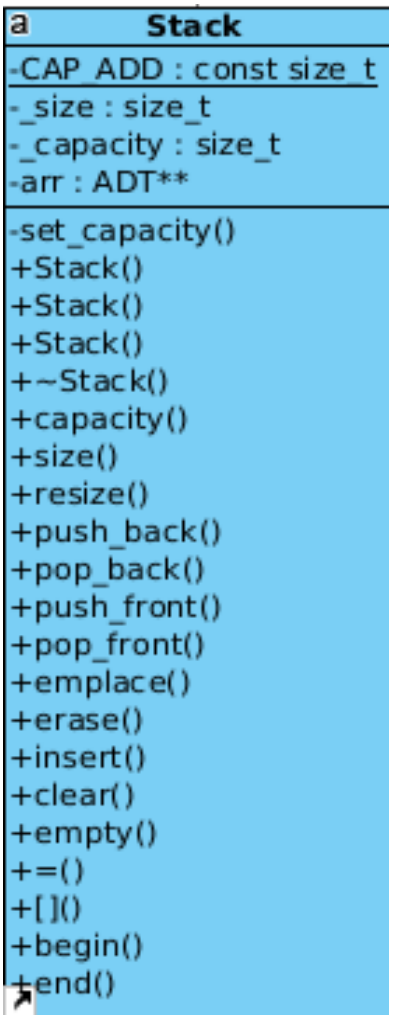

Рис.3 UML диаграмма класса Stack

2.4 Задача №4

Класс написан на языке C++. Код класса размещается в одной единицах трансляции – код в файле list.cpp, прототип класса в заголовочном файле list.h. Тесты в виде проверки результатов и постусловий с помощью утверждений выполнены в единице трансляции main_list.cpp с функцией main, при этом, заголовочный файл содержит защиту от повторного включения.

Реализация класса включает в себя закрытый класс Node — узел, хранящий адреса следующего узла — **next** и указатель на объект типа ADT — **data**. Класс List содержит закрытые поля **_size** типа size_t, **first** типа указатель на объект типа Tnode и last — типа указатель на объект типа Tnode, где **_size** – количество узлов в массиве, **fisrt** — указатель на начальный узел и **last** — указатель на конечный узел.

Присутствует конструктор по умолчанию задающий поля класса нулевыми значениями. Конструктор общего вида принимает число типа size_t — задающее, размер массива **_size** (количество узлов).

Реализованы методы для взаимодействия с массивом — **push_back, push_front, pop_back, pop_front** для добавления элемента в конец/начало массива и удаления элемента в конец/начало массива соответственно., **insert** — для добавления узла в произвольное место массива, метод **erase** для удаления узла из произвольного места массива, метод **empalce** для замены значения поля данных произвольного узла. Для доступа к закрытому полю **_size** был реализован соответствующий акцессор. Для доступа к указателям на объекты типа Node был перегружен оператор квадратных скобок. Метод empty возвращает логическое значение соответствующие пустоте массива.

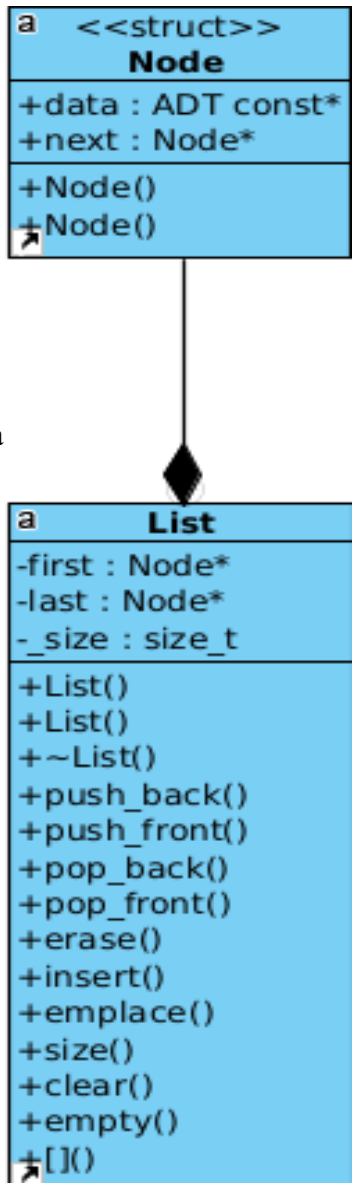

Рис.4 UML диаграмма класса List

2.5 Задача №5

Класс написан на языке C++. Код класса размещается в одной единицах трансляции – код в файле queue.cpp, прототип класса в заголовочном файле queue.h. Тесты в виде проверки результатов и постусловий с помощью утверждений выполнены в единице трансляции main_queue.cpp с функцией main, при этом, заголовочный файл содержит защиту от повторного включения.

Реализация класса включает в себя закрытые поля **_size**, **_quantity** типа size_t и **arr** типа указатель на указатель на объект типа ADT, где **_size** – количество выделенной памяти под массив, а **_quantity** — количество элементов в массиве. Конструктор по умолчанию отсутствует. Конструктор общего вида принимает число типа size_t — задающее, размер массива _size.

Реализованы методы для взаимодействия с массивом — **push_back** и **pop_back**, для вставки и извлечения последнего элемента, **insert** — для добавления элемента в произвольное место массива, метод **erase** для удаления элемента из произвольного места массива, метод **empalce** для замены произвольного элмента. Реализовано циклическое поведение массиво согласно принципу FIFO — first-in-first-out, при сдвиге элементов, последний элемент становится первым и продолжает сдвиг, элемент, стоящий на позиции до вставляемого уничтожается. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок. Для доступа к закрытому полю **_size** был реализован акцессор. Методы **begin** и **end** возвращают указатель на начало и конец массива соответственно. Метод empty возвращает логическое значение соответствующие пустоте массива.

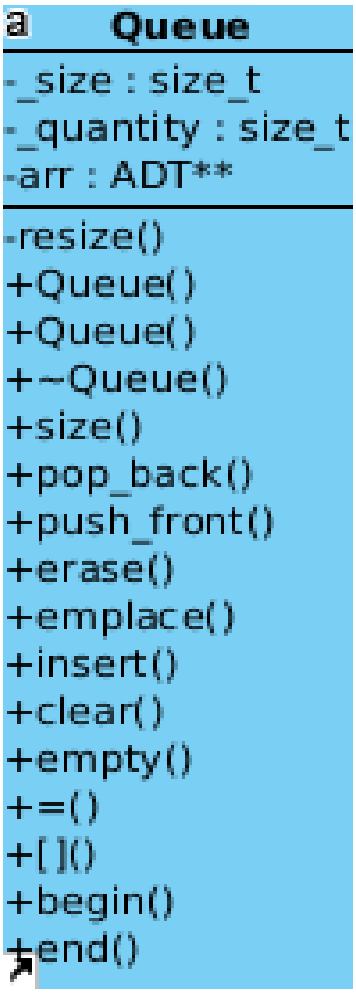

Рис.5 UML диаграмма класса Queue

2.6 Задача №6

Класс написан на языке C++. Код класса размещается в одной единицах трансляции – код в файле tree.cpp, прототип класса в заголовочном файле tree.h. Тесты в виде проверки результатов и постусловий с помощью утверждений выполнены в единице трансляции main_tree.cpp с функцией main, при этом, заголовочный файл содержит защиту от повторного включения.

Реализация класса включает в себя закрытый класс Tnode — лист, хранящий указатели на левый — **left** и правый — **right** лист дерева, родитель — **parent** , объект типа ADT — **data** и **pos** — позиция элемента в дереве(ключ). По-мимо этого в класс входит поле tree — указатель на объект типа Tnode.

Контруктор по умолчанию задаёт поля класса нулевыми значениями. Присутствуют закрытые служебные методы, обеспечивающие работоспособность класса.

Для корректного освобождения памяти был создан закрытый метод **delete_tree**, рекурсивно вызывающий сам себя, для прохода по всем листам дерева. Аналогично работает метод **print** для отображения значений, хранящихся в листах дерева.

Реализованы методы для взаимодействия с массивом — **insert** для добавления листа в произвольное место дерева и метод **erase** для удаления листа из произвольного места дерева. Для доступа к указателям на объекты типа ADT был перегружен оператор квадратных скобок. Для доступа к закрытому полю **_size** был реализован соответствующий акцессор. Для доступа к указателям на объекты типа Node был перегружен оператор квадратных скобок. Метод empty возвращает логическое значение соответствующие пустоте массива.

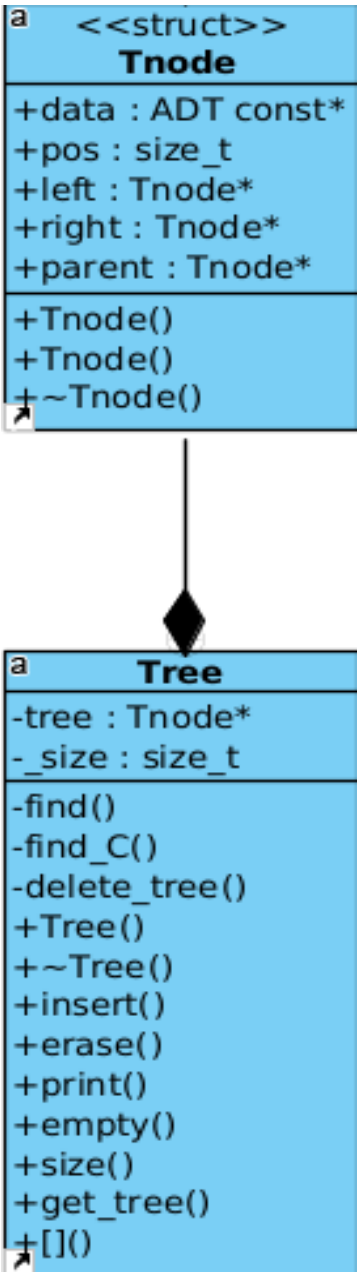

Рис.6 UML диаграмма класса Tree

3 Получение исполняемых модулей

Получение исполняемых модулей происходит с помощью системы сборки cmake. Задан стандарт языка C++14 и ключи компиляции -lgtest, для статического подключения библиотеки googletest, минимальная версия cmake 3.14.

Для подключения библеотеки для тестирования — googletest, в файле CmakeList.txt производится загрузка из оффициального источника распространения библиотеки.

Для уопрощения тестирования всех исполняемых модулей, был разработан файл main_lab2, содержащий тестирования всех разработанных классов.

Листинг 1 – Файл CmakeList.txt

```
cmake_minimum_required(VERSION 3.14 FATAL_ERROR)
project(lab_02)
add_definitions(-lgtest)

set(CMAKE_CXX_STANDARD 14)

include(FetchContent)

FetchContent_Declare(
googletest
GIT_REPOSITORY https://github.com/google/googletest.git
GIT_TAG release-1.8.0
)

add_executable(${PROJECT_NAME}  "data_time.cpp" "data_time.h"
                                "new_integer.cpp" "new_integer.h"
                                "Adam_time.cpp" "Adam_time.h"
                                "spare_matrix.cpp" "spare_matrix.h"
                                "dycont.cpp" "dycont.h"
                                "stack.cpp" "stack.h"
                                "list.cpp" "list.h"
                                "queue.cpp" "queue.h"
                                "tree.cpp" "tree.h"
                                "main_lab2.cpp"
                                )
```

4 Тестирование

Тестирование классов проводилось с использованием макросов библиотеки googletest. Были задействованы макросы EXPECT_STREQ — для сравнения строк, EXPECT_EQ — для сравнения арифметических значений, EXPECT_TRUE и EXPECT_FALSE — для логических сравнений.

4.1 Задача №1

Проведён базовый тест на функциональность класса: был создан объект класса и переданы ему указаетли на динамически выделенные объекты класса Integer и Adam_t.

4.2 Задача №2

Проведён базовый тест на функциональность класса: был создан объект класса и переданы ему указаетли на объекты класса ADT. Были проверены на работоспособность методы push_back, pop_back, insert, erase, emplace, size, empty и clear.

4.3 Задача №3

Проведён базовый тест на функциональность класса: был создан объект класса и переданы ему указаетли на объекты класса ADT. Были проверены на работоспособность методы push_back, pop_back, push_front, pop_front, insert, erase, emplace, size, empty и clear.

4.4 Задача №4

Проведён базовый тест на функциональность класса: был создан объект класса и переданы ему указаетли на объекты класса ADT. Были проверены на работоспособность методы push_back, pop_back, push_front, pop_front, insert, erase, emplace, size, empty и clear.

4.5 Задача №5

Проведён базовый тест на функциональность класса: был создан объект класса и переданы ему указаетли на объекты класса ADT. Были проверены на работоспособность методы push_front, pop_back, insert, erase, size, empty и clear.

4.6 Задача №6

Проведён базовый тест на функциональность класса: был создан объект класса и переданы ему указаетли на объекты класса ADT. Были проверены на работоспособность методы insert, erase, size и empty.

Приложение А

А – Файл ADT.h

```
#ifndef ADT_h
#define ADT_h

#include <iostream>

class ADT
{
    //
    public:
        virtual ~ADT() = default;
        virtual std::ostream &print(std::ostream &out) const { return out; }
        friend std::ostream &operator<<(std::ostream& out, const ADT& right)
        {
            return right.print(out);
        }

};

#endif // ADT_h
```

А – Файл main_lab2.cpp

```
#include <iostream>
#include <string>

#include "dycont.h"
#include "stack.h"
#include "list.h"
#include "queue.h"
#include "tree.h"

#include "data_time.h"
#include "new_integer.h"
#include "Adam_time.h"
#include "spare_matrix.h"

#include <gtest/gtest.h>


TEST(TestGroupMane, TestName)
{
    ASSERT_TRUE(true);
}
```

```
int main()
{
    ::testing::InitGoogleTest();
    std::cout << "Dycont:\n";
    Integer var("32832183912938");
    ADT *test1 = &var;
    Integer *test2 = new Integer(21212312);
    Dycont qwa_dycont(2);
    qwa_dycont[0] = test1;
    qwa_dycont[1] = test2;
    std::cout <<  *(qwa_dycont[0]) << std::endl;
    //test на результат сложения
    std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_dycont[0])).get_num_str()).c_st
r());
    EXPECT_FALSE(*dynamic_cast<Integer*>(qwa_dycont[0]) ==
*dynamic_cast<Integer*>(qwa_dycont[1]) );
    std::cout <<"########################################\n";

    ADT *test3 = new Adam_t(5,20,15);
    qwa_dycont.push_back(test3);
    // тест на хранимые данные в qwa_dycont[2]
    EXPECT_EQ(2960, (*dynamic_cast<Adam_t*>(qwa_dycont[2])).get_Adam_year() );
    std::cout << "test: push_back "<< *dynamic_cast<Adam_t*>(qwa_dycont[2]) <<
std::endl;
    for(size_t i = 0; i< qwa_dycont.size(); ++i)
        std::cout <<  (*qwa_dycont[i]) << std::endl;

    std::cout <<"########################################\n";

    qwa_dycont.pop_back();
    // тест на хранимые данные в qwa_dycont[2]
    std::cout << "test: pop_back "<< std::endl;
    for(size_t i = 0; i< qwa_dycont.size(); ++i)
        std::cout <<  (*qwa_dycont[i]) << std::endl;

    std::cout <<"########################################\n";

    Adam_t qwer(9,12,11);
    ADT *test5 = &qwer;
    qwa_dycont.insert(1,test5);
    // тест на хранимые данные в qwa_dycont[1]

    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(qwa_dycont[1])).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa_dycont.size(); ++i)
        std::cout <<  (*qwa_dycont[i]) << std::endl;

    std::cout << "size = " << qwa_dycont.size() << "\t capacity = " <<
qwa_dycont.capacity() << std::endl;
    // тест на size и capacity

    std::cout <<"##################\n";
```

```
    qwa_dycont.erase(1);
    // тест на хранимые данные в qwa[1]

    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_dycont[0])).get_num_str()).c_st
r());
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa_dycont.size(); ++i)
        std::cout <<  (*qwa_dycont[i]) << std::endl;

    std::cout << "new size = " << qwa_dycont.size() << "\t new capacity = " <<
qwa_dycont.capacity() << std::endl;
    // тест на size и capacity

    std::cout <<"########################################\n";

    ADT *test6 = new Adam_t(5,14,10);
    qwa_dycont.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(qwa_dycont[0])).get_Adam_year() );

    std::cout << "test: emaplce(0)\n";
    // тест на хранимые данные в qwa_dycont[0]
    for(size_t i = 0; i< qwa_dycont.size(); ++i)
        std::cout <<  (*qwa_dycont[i]) << std::endl;


    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_dycont.empty() << '\n';
    EXPECT_FALSE(qwa_dycont.empty());
    qwa_dycont.clear();
    EXPECT_TRUE(qwa_dycont.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_dycont.empty() << '\n';

    std::cout <<"\n\n########################################\n\n";
    std::cout << "Stack:\n";


    //Integer *test2 = new Integer(21212312);
    Stack qwa_stack(2);
    qwa_stack[0] = test1;
    qwa_stack[1] = test2;
    //std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_stack[0])).get_num_str()).c_str
());
    EXPECT_FALSE(*dynamic_cast<Integer*>(qwa_stack[0]) ==
*dynamic_cast<Integer*>(qwa_stack[1]) );
    std::cout <<  *(qwa_stack[0]) << std::endl;
    std::cout <<"########################################\n";

    //ADT *test3 = new Adam_t(5,20,15);
    qwa_stack.push_back(test3);
    // тест на хранимые данные в qwa_stack[2]
```

```
    EXPECT_EQ(2960, (*dynamic_cast<Adam_t*>(qwa_stack[2])).get_Adam_year() );
    std::cout << "test: push_back "<< *dynamic_cast<Adam_t*>(qwa_stack[2]) <<
std::endl;
    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;

    std::cout <<"########################################\n";

    ADT *test4 = new Integer(12345);
    qwa_stack.push_front(test4);
    // тест на хранимые данные в qwa_stack[2]
    std::string test_int = "12345";
    EXPECT_STREQ(test_int.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_stack[0])).get_num_str()).c_str
());
    std::cout << "test: push_front "<< *(qwa_stack[0]) << std::endl;
    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;

    std::cout <<"########################################\n";

    qwa_stack.pop_back();
    // тест на хранимые данные в qwa_stack[2]
    std::cout << "test: pop_back "<< std::endl;
    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;

    std::cout <<"########################################\n";
    qwa_stack.pop_front();
    // тест на хранимые данные в qwa_stack[2]
    std::cout << "test: pop_front "<< std::endl;
    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;

    std::cout <<"########################################\n";

    //Adam_t qwer(9,12,11);
    //ADT *test5 = &qwer;
    qwa_stack.insert(1,test5);
    // тест на хранимые данные в qwa_stack[1]

    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(qwa_stack[1])).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;

    std::cout << "size = " << qwa_stack.size() << "\t capacity = " <<
qwa_stack.capacity() << std::endl;
    // тест на size и capacity

    std::cout <<"##################\n";
    qwa_stack.erase(1);
    // тест на хранимые данные в qwa_stack[1]
```

```
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_stack[0])).get_num_str()).c_str
());
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;

    std::cout << "new size = " << qwa_stack.size() << "\t new capacity = " <<
qwa_stack.capacity() << std::endl;
    // тест на size и capacity

    std::cout <<"########################################\n";

    //ADT *test6 = new Adam_t(5,14,10);
    qwa_stack.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(qwa_stack[0])).get_Adam_year() );

    std::cout << "test: emaplce(0)\n";
    // тест на хранимые данные в qwa_stack[0]
    for(size_t i = 0; i< qwa_stack.size(); ++i)
        std::cout <<  (*qwa_stack[i]) << std::endl;


    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_stack.empty() << '\n';
    EXPECT_FALSE(qwa_stack.empty());
    qwa_stack.clear();
    EXPECT_TRUE(qwa_stack.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_stack.empty() << '\n';


    std::cout <<"\n\n########################################\n\n";
    std::cout << "List:\n";

    List qwa_list;
    qwa_list.push_back(test1);
    qwa_list.push_back(test2);
    //std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(const_cast<ADT*>(qwa_list[0]-
>data))).get_num_str()).c_str());

    EXPECT_FALSE(*dynamic_cast<Integer*>(const_cast<ADT*>(qwa_list[0]->data)) ==
*dynamic_cast<Integer*>(const_cast<ADT*>(qwa_list[1]->data)) );
    // тест на сравнение строк
    std::cout <<"########################################\n";

    //ADT *test3 = new Adam_t(5,20,15);
    qwa_list.push_back(test3);
    // тест на хранимые данные в qwa_list[2]
    std::cout << "test: push_back "<<  std::endl;
    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout <<  (*qwa_list[i]->data) << std::endl;
```

```
    std::cout <<"########################################\n";

    //ADT *test4 = new Integer(12345);
    qwa_list.push_front(test4);
    // тест на хранимые данные в qwa_list[2]
    std::cout << "test: push_front "<< *(qwa_list[0]->data) << std::endl;
    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout <<  (*qwa_list[i]->data) << std::endl;

    std::cout <<"########################################\n";

    qwa_list.pop_back();
    // тест на хранимые данные в qwa_list[2]
    std::cout << "test: pop_back "<< std::endl;
    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout <<  *(qwa_list[i]->data) << std::endl;

    std::cout <<"########################################\n";
    qwa_list.pop_front();
    // тест pop_front
    std::cout << "test: pop_front "<< std::endl;
    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout <<  (*qwa_list[i]->data) << std::endl;

    std::cout <<"########################################\n";

    // тест insert
    //Adam_t qwer(9,12,11);
    //ADT *test5 = &qwer;
    qwa_list.insert(1,test5);
    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(const_cast<ADT*>(qwa_list[1]-
>data))).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout <<  (*qwa_list[i]->data) << std::endl;

    // тест size
    std::cout << "size = " << qwa_list.size() << std::endl;

    // тест erase
    std::cout <<"##################\n";
    qwa_list.erase(1);
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout << (*qwa_list[i]->data) << std::endl;

    std::cout << "new size = " << qwa_list.size() << std::endl;

    std::cout <<"########################################\n";

    // тест emplace
    //ADT *test6 = new Adam_t(5,14,10);
```

```
    qwa_list.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(const_cast<ADT*>(qwa_list[0]-
>data))).get_Adam_year() );
    std::cout << "test: emaplce(0)\n";
    for(size_t i = 0; i< qwa_list.size(); ++i)
        std::cout << (*qwa_list[i]->data) << std::endl;

    // тест на empty и clear
    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_list.empty() << '\n';
    EXPECT_FALSE(qwa_list.empty());
    qwa_list.clear();
    EXPECT_TRUE(qwa_list.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_list.empty() << '\n';



    std::cout
<<"\n\n############################################################\n\n";
    std::cout << "Queue:\n";



    Queue qwa_queue(3);
    qwa_queue.push_front(test1);
    qwa_queue.push_front(test2);
    //std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_queue[1])).get_num_str()).c_str
());
    std::cout <<"########################################\n";

    //ADT *test4 = new Integer(12345);
    qwa_queue.push_front(test4);
    // тест на хранимые данные в qwa_queue
    std::string sad= "12345";
    EXPECT_STREQ(sad.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_queue[0])).get_num_str()).c_str
());
    std::cout << "test: push_front "<< *(qwa_queue[0]) << std::endl;
    for(size_t i = 0; i< qwa_queue.size(); ++i)
        std::cout <<  (*qwa_queue[i]) << std::endl;

    std::cout <<"########################################\n";



    // тест на хранимые данные в qwa_queue
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa_queue.pop_back())).get_num_str(
)).c_str());
    std::cout << "test: pop_back " << *(qwa_queue.pop_back()) << std::endl;
    for(size_t i = 0; i< qwa_queue.size(); ++i)
        std::cout <<  (*qwa_queue[i]) << std::endl;
```

```
    std::cout <<"########################################\n";


    //Adam_t qwer(9,12,11);
    //ADT *test5 = &qwer;
    qwa_queue.insert(1,test5);
    // тест на хранимые данные в qwa_queue
    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(qwa_queue[1])).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa_queue.size(); ++i)
        std::cout <<  (*qwa_queue[i]) << std::endl;

    std::cout <<"########################################\n";

    ADT *test8 = new Integer(5432);
    qwa_queue.push_front(test8);
    // тест на хранимые данные в qwa_queue[2]
    std::cout << "test: push_front "<< *(qwa_queue[0]) << std::endl;
    for(size_t i = 0; i< qwa_queue.size(); ++i)
        std::cout <<  (*qwa_queue[i]) << std::endl;

    std::cout <<"########################################\n";
    std::cout << "size = " << qwa_queue.size() << std::endl;
    // тест на size и capacity

    std::cout <<"##################\n";
    qwa_queue.erase(1);
    // тест на хранимые данные в qwa_queue[1]
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa_queue.size(); ++i)
        std::cout <<  (*qwa_queue[i]) << std::endl;

    std::cout << "new size = " << qwa_queue.size() << std::endl;
    // тест на size и capacity

    std::cout <<"########################################\n";

    //ADT *test6 = new Adam_t(5,14,10);
    qwa_queue.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(qwa_queue[0])).get_Adam_year() ); // тест
на хранимые данные в qwa_queue[0]
    std::cout << "test: emaplce(0)\n";
    for(size_t i = 0; i< qwa_queue.size(); ++i)
        std::cout <<  (*qwa_queue[i]) << std::endl;


    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_queue.empty() << '\n';
    EXPECT_FALSE(qwa_queue.empty());
    qwa_queue.clear();
    EXPECT_TRUE(qwa_queue.empty());
    std::cout << "test: clear\n";
```

```
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_queue.empty() << '\n';


    std::cout <<"\n\n#########################################\n\n";
    std::cout << "Tree:\n";



    Tree qwa_tree;
    ADT *test11 = new Integer("123456");
    ADT *test22 = new Integer("8998");
    ADT *test33 = new Integer("5555");
    ADT *test44 = new Integer(444);

    qwa_tree.insert(4,test11);
    qwa_tree.insert(1,test22);
    qwa_tree.insert(12,test33);
    qwa_tree.insert(6,test44);

    std::string qwaqwa_tree= "123456";
    EXPECT_STREQ(qwaqwa_tree.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(const_cast<ADT*>(qwa_tree[4])))).get
_num_str()).c_str());

    // тест insert
    //Adam_t qwer(9,12,11);
    //ADT *test5 = &qwer;
    qwa_tree.insert(2,test5);
    EXPECT_EQ(6984,
(*dynamic_cast<Adam_t*>(const_cast<ADT*>(qwa_tree[2])))).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;
    qwa_tree.print(qwa_tree.get_tree(), std::cout);
    // тест erase
    std::cout <<"###################\n";
    qwa_tree.erase(6);
    std::cout << "test: erase(1)\n";

    qwa_tree.print(qwa_tree.get_tree(), std::cout);

    std::cout <<"#########################################\n";

    // тест на empty
    std::cout <<"#########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa_tree.empty() << '\n';
    EXPECT_FALSE(qwa_tree.empty());
    return RUN_ALL_TESTS();
}
```

Приложение Б

Б – Файл dycont.h

```
#ifndef dy_mass_h
#define dy_mass_h

#include <iostream>

#include "ADT.h"

class Dycont
{
    private:

        const static size_t CAP_ADD = 5;

        size_t _size;
        size_t _capacity;
        ADT **arr;

        void set_capacity(size_t new_size);

    public:

        Dycont();
        Dycont(int size);
        Dycont(const Dycont &right);
        ~Dycont();

        size_t capacity() const;
        size_t size() const;
        void resize(size_t new_size);
        void push_back(ADT *right);
        void pop_back();
        void emplace(size_t pos, ADT *value);
        void erase(size_t pos);
        void insert(size_t pos, ADT *value);
        void clear();
        bool empty() const;

        const Dycont &operator= (const Dycont &right);
        ADT* &operator[](size_t pos);
        ADT* &begin();
        ADT* &end();


};

#endif // dy_mass_h
```

Б – Файл dycont.cpp

```
#include <iostream>
#include <algorithm>

#include "dycont.h"

void Dycont::set_capacity(size_t new_size = 0)
{
    if (new_size) this->_capacity = (new_size * 3) / 2 + 1;
    else this->_capacity = (_size * 3) / 2 + 1;
}

Dycont::Dycont() : _size(0), _capacity(0)  { arr = nullptr; }

Dycont::Dycont(int size) : _size(size)
{
    if (_size < 0) std::runtime_error( "Error: length < 0");
    set_capacity();
    arr = new ADT*[_capacity];
    for (size_t i = 0; i < _size; ++i)
    {
        arr[i] = new ADT;
    }
}

Dycont::Dycont(const Dycont &right)
: _size(right._size)
{
    this->_capacity = right._capacity;
    try
    {
        this->arr = new ADT* [this->_capacity];
        for (size_t i = 0; i < _size; ++i)
        {
            arr[i] = new ADT;
            arr[i] = right.arr[i];
        }
    }
    catch (...)
    {
        std::runtime_error( "Error memory allocated" );
        exit(1);
    }
}

Dycont::~Dycont()
{
    if (arr != nullptr)
    {
```

```
        delete [] arr;
        arr = nullptr;
    }

}

size_t Dycont::capacity() const { return this->_capacity; }
size_t Dycont::size() const { return this->_size; }


void Dycont::resize(size_t new_size)
{
    this->_size = new_size;
    size_t old_capacity = this->_capacity;
    set_capacity(new_size);
    ADT **new_arr;
    try
    {
        new_arr = new ADT*[this->_capacity];
        for (size_t i = 0; i < this->_size; ++i )
        {
            new_arr[i] = new ADT;
            new_arr[i] = arr[i];
        }
    }
    catch (...)
    {
        std::runtime_error( "Error memory allocated" );
        exit(1);
    }
    for (size_t i = 0; i < old_capacity; ++i) delete [] arr[i];
    delete [] arr;
    arr = new_arr;

}

void Dycont::push_back(ADT *value)
{
    if (_capacity == _size) resize(_size + 1);
    arr[_size++] = value;
}
void Dycont::pop_back()
{
    if (arr != nullptr && _size > 0) --_size;
}
void Dycont::emplace(size_t pos, ADT *value)
{
    arr[pos] = value;
}

void Dycont::insert(size_t pos, ADT *value)
{
    ++_size;
    if (_capacity == _size)
```

```
    {
        set_capacity();
        ADT **new_arr;
        try
        {
            new_arr = new ADT*[this->_capacity];
            for (size_t i = 0; i < this->_size; ++i)
            {
                new_arr[i] = new ADT;
            }
        }
        catch (...)
        {
            std::runtime_error( "Error memory allocated" );
            exit(1);
        }
        for (size_t i = 0; i < pos; ++i ) new_arr[i] = arr[i];
        for (size_t i = pos+1; i < _size; ++i ) new_arr[i] = arr[i-1];
        new_arr[pos] = value;
        delete [] arr;
        arr = new_arr;
    }
    else
    {
        ADT *var = arr[pos];
        arr[pos] = value;
        ADT *qwa;
        for (size_t i = pos+1; i < _size; ++i )
        {
            if (i < _size) qwa = arr[i];
            arr[i] = var;
            var = qwa;
        }
    }
}

void Dycont::erase(size_t pos)
{
    --_size;
    for (size_t i = pos; i < _size; ++i) arr[i] = arr[i+1];
}

void Dycont::clear()
{
    delete [] arr;
    arr = nullptr;
    _size = 0;
    _capacity = 0;
}
bool Dycont::empty() const { return _size == 0; }

//const Dycont &Dycont::operator= (const ADT &right)

const Dycont &Dycont::operator= (const Dycont& right)
```

```
{
    if (_capacity != right._capacity)
    {
        this->_size = right._size;
        if (arr != nullptr)
        {
            for (size_t i = 0; i < this->_capacity; ++i) delete [] arr[i];
            delete [] arr;
            arr = nullptr;
        }
        this->_capacity = right._capacity;
        try
        {
            this->arr = new ADT* [right._capacity];
            for (size_t i = 0; i < right._size; ++i)
            {
                arr[i] = new ADT;
            }
        }
        catch (...)
        {
            std::runtime_error( "Error memory allocated" );
            exit(1);
        }
    }
    for ( size_t i = 0; i < right._size; ++i )
        arr[i] = right.arr[i];
    return *this;
}
ADT* &Dycont::operator[](size_t pos) { return arr[pos]; }
ADT* &Dycont::begin() { return *arr; }
ADT* &Dycont::end() { return *(arr + _size); }
```

Б – Файл main_dycont.cpp

```
#include <iostream>
#include <string>

#include "dycont.h"
#include "new_integer.h"
#include "Adam_time.h"
#include <gtest/gtest.h>


TEST(TestGroupMane, TestName)
{
    ASSERT_TRUE(true);
}


int main()
{
```

```
    ::testing::InitGoogleTest();
    Integer var("32832183912938");
    ADT *test1 = &var;
    Integer *test2 = new Integer(21212312);
    Dycont qwa(2);
    qwa[0] = test1;
    qwa[1] = test2;
    std::cout <<  *(qwa[0]) << std::endl;
    //test на результат сложения
    std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[0])).get_num_str()).c_str());
    EXPECT_FALSE(*dynamic_cast<Integer*>(qwa[0]) ==
*dynamic_cast<Integer*>(qwa[1]) );
    std::cout <<"########################################\n";

    ADT *test3 = new Adam_t(5,20,15);
    qwa.push_back(test3);
    // тест на хранимые данные в qwa[2]
    EXPECT_EQ(2960, (*dynamic_cast<Adam_t*>(qwa[2])).get_Adam_year() );
    std::cout << "test: push_back "<< *dynamic_cast<Adam_t*>(qwa[2]) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";

    qwa.pop_back();
    // тест на хранимые данные в qwa[2]
    std::cout << "test: pop_back "<< std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";

    Adam_t qwer(9,12,11);
    ADT *test5 = &qwer;
    qwa.insert(1,test5);
    // тест на хранимые данные в qwa[1]

    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(qwa[1])).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout << "size = " << qwa.size() << "\t capacity = " << qwa.capacity() <<
std::endl;
    // тест на size и capacity

    std::cout <<"##################\n";
    qwa.erase(1);
    // тест на хранимые данные в qwa[1]

    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[0])).get_num_str()).c_str());
```

```
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout << "new size = " << qwa.size() << "\t new capacity = " <<
qwa.capacity() << std::endl;
    // тест на size и capacity

    std::cout <<"########################################\n";

    ADT *test6 = new Adam_t(5,14,10);
    qwa.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(qwa[0])).get_Adam_year() );

    std::cout << "test: emaplce(0)\n";
    // тест на хранимые данные в qwa[0]
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;


    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';
    EXPECT_FALSE(qwa.empty());
    qwa.clear();
    EXPECT_TRUE(qwa.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';


    return RUN_ALL_TESTS();
}
```

Приложение В

В – Файл stack.h

```
#ifndef stack_h
#define stack_h

#include <iostream>

#include "ADT.h"

class Stack
{
    private:
        const static size_t CAP_ADD = 5;

        size_t _size;
```

```
        size_t _capacity;
        ADT** arr;

        void set_capacity(size_t new_size);

    public:

        Stack();
        Stack(size_t size);
        Stack(const Stack &right);
        ~Stack();

        size_t capacity() const;
        size_t size() const;
        void resize(size_t new_size);
        void push_back(ADT *right);
        void pop_back();
        void push_front(ADT *value);
        void pop_front();
        void emplace(size_t pos, ADT *value);
        void erase(size_t pos);
        void insert(size_t pos, ADT *value);
        void clear();
        bool empty() const;

        const Stack &operator= (const Stack &right);
        ADT* &operator[](size_t pos);
        ADT* &begin();
        ADT* &end();
};

#endif // stack_h
```

В – Файл stack.cpp

```
#include "stack.h"

void Stack::set_capacity(size_t new_size = 0)
{
    if (new_size) this->_capacity = (new_size * 3) / 2 + 1;
    else this->_capacity = (_size * 3) / 2 + 1;
}

Stack::Stack() : _size(0), _capacity(0)  { *arr = nullptr;}

Stack::Stack(size_t size) : _size(size)
{
    set_capacity();
    arr = new ADT*[_capacity];
    for (size_t i = 0; i < _size; ++i)
    {
        arr[i] = new ADT;
```

```
    }
}

Stack::Stack(const Stack &right)
: _size(right._size)
{
    if (arr != nullptr)
    {
        delete [] arr;
        arr = nullptr;
    }
    this->_capacity = right._capacity;
    try
    {
        this->arr = new ADT* [this->_capacity];
        for (size_t i = 0; i < _size; ++i)
        {
            arr[i] = new ADT;
            arr[i] = right.arr[i];
        }
    }
    catch (...)
    {
        std::runtime_error( "Error memory allocated" );
        exit(1);
    }
}

Stack::~Stack()
{
    if (arr != nullptr)
    {
        for (size_t i = 0; i < this->_capacity; ++i) delete arr[i];
        delete [] arr;
        arr = nullptr;
    }

}

size_t Stack::capacity() const { return this->_capacity; }
size_t Stack::size() const { return this->_size; }

void Stack::resize(size_t new_size)
{
    this->_size = new_size;
    size_t old_capacity = this->_capacity;
    set_capacity(new_size);
    ADT **new_arr;
    try
    {
        new_arr = new ADT*[this->_capacity];
        for (size_t i = 0; i < this->_size; ++i )
        {
            new_arr[i] = new ADT;
```

```
            new_arr[i] = arr[i];
        }
    }
    catch (...)
    {
        std::runtime_error( "Error memory allocated" );
        exit(1);
    }
    for (size_t i = 0; i < old_capacity; ++i) delete [] arr[i];
    delete [] arr;
    arr = new_arr;
}

void Stack::push_back(ADT *value)
{
    if (_capacity == _size) resize(_size + 1);
    arr[_size++] = value;
}
void Stack::pop_back()
{
    if (!empty()) --_size;
}

void Stack::push_front(ADT *value)
{
    this->insert(0, value);
}
void Stack::pop_front()
{
    if (empty()) return;
    this->erase(0);
}

void Stack::emplace(size_t pos, ADT *value)
{
    arr[pos] = value;
}
void Stack::insert(size_t pos, ADT *value)
{
    ++_size;
    if (_capacity == _size)
    {
        set_capacity();
        ADT **new_arr;
        try
        {
            new_arr = new ADT*[this->_capacity];
            for (size_t i = 0; i < this->_size; ++i)
            {
                new_arr[i] = new ADT;
            }
        }
        catch (...)
        {
```

```
            std::runtime_error( "Error memory allocated" );
            exit(1);
        }
        for (size_t i = 0; i < pos; ++i ) new_arr[i] = arr[i];
        for (size_t i = pos+1; i < _size; ++i ) new_arr[i] = arr[i-1];
        new_arr[pos] = value;
        delete [] arr;
        arr = new_arr;
    }
    else
    {
        ADT *var = arr[pos];
        arr[pos] = value;
        ADT *qwa;
        for (size_t i = pos+1; i < _size; ++i )
        {
            if (i < _size) qwa = arr[i];
            arr[i] = var;
            var = qwa;
        }
    }
}
void Stack::erase(size_t pos)
{
    --_size;
    for (size_t i = pos; i < _size; ++i) arr[i] = arr[i+1];
}

void Stack::clear()
{
    delete [] arr;
    arr = nullptr;
    _size = 0;
    _capacity = 0;
}
bool Stack::empty() const { return _size == 0; }
const Stack &Stack::operator= (const Stack &right)
{
    if (_capacity != right._capacity)
    {
        this->_size = right._size;
        if (arr != nullptr)
        {
            for (size_t i = 0; i < this->_capacity; ++i) delete [] arr[i];
            delete [] arr;
            arr = nullptr;
        }
        this->_capacity = right._capacity;
        try
        {
            this->arr = new ADT* [right._capacity];
            for (size_t i = 0; i < right._size; ++i)
            {
                arr[i] = new ADT;
```

```
            }
        }
        catch (...)
        {
            std::runtime_error( "Error memory allocated" );
            exit(1);
        }
    }
    for ( size_t i = 0; i < right._size; ++i )
        arr[i] = right.arr[i];
    return *this;
}

ADT* &Stack::operator[](size_t pos) { return arr[pos]; }
ADT* &Stack::begin() { return *arr; }
ADT* &Stack::end() { return *(arr + _size); }
```

В – Файл main_stack.cpp

```
#include <iostream>

#include "stack.h"
#include "new_integer.h"
#include "Adam_time.h"
#include <gtest/gtest.h>

TEST(TestGroupMane, TestName)
{
    ASSERT_TRUE(true);
}

int main()
{
    ::testing::InitGoogleTest();
    Integer var("32832183912938");
    ADT *test1 = &var;
    Integer *test2 = new Integer(21212312);
    Stack qwa(2);
    qwa[0] = test1;
    qwa[1] = test2;
    std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[0])).get_num_str()).c_str());
    EXPECT_FALSE(*dynamic_cast<Integer*>(qwa[0]) ==
*dynamic_cast<Integer*>(qwa[1]) );
    std::cout <<  *(qwa[0]) << std::endl;
    std::cout <<"########################################\n";

    ADT *test3 = new Adam_t(5,20,15);
    qwa.push_back(test3);
    // тест на хранимые данные в qwa[2]
    EXPECT_EQ(2960, (*dynamic_cast<Adam_t*>(qwa[2])).get_Adam_year() );
    std::cout << "test: push_back "<< *dynamic_cast<Adam_t*>(qwa[2]) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
```

```
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";

    ADT *test4 = new Integer(12345);
    qwa.push_front(test4);
    // тест на хранимые данные в qwa[2]
    std::string test_int = "12345";
    EXPECT_STREQ(test_int.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[0])).get_num_str()).c_str());
    std::cout << "test: push_front "<< *(qwa[0]) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";

    qwa.pop_back();
    // тест на хранимые данные в qwa[2]
    std::cout << "test: pop_back "<< std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";
    qwa.pop_front();
    // тест на хранимые данные в qwa[2]
    std::cout << "test: pop_front "<< std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";

    Adam_t qwer(9,12,11);
    ADT *test5 = &qwer;
    qwa.insert(1,test5);
    // тест на хранимые данные в qwa[1]

    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(qwa[1])).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout << "size = " << qwa.size() << "\t capacity = " << qwa.capacity() <<
std::endl;
    // тест на size и capacity

    std::cout <<"##################\n";
    qwa.erase(1);
    // тест на хранимые данные в qwa[1]

    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[0])).get_num_str()).c_str());
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;
```

```
    std::cout << "new size = " << qwa.size() << "\t new capacity = " <<
qwa.capacity() << std::endl;
    // тест на size и capacity

    std::cout <<"########################################\n";

    ADT *test6 = new Adam_t(5,14,10);
    qwa.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(qwa[0])).get_Adam_year() );

    std::cout << "test: emaplce(0)\n";
    // тест на хранимые данные в qwa[0]
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;


    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';
    EXPECT_FALSE(qwa.empty());
    qwa.clear();
    EXPECT_TRUE(qwa.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';


    return RUN_ALL_TESTS();
}
```

Приложение Г

Г – Файл list.h

```
#ifndef list_h
#define list_h

#include <iostream>

#include "ADT.h"

class List
{

private:
    // Структура узла односвязного списка
    struct Node
    {
        // Значение узла
        const ADT *data;
```

```
        // Указатель на следующий узел
        Node *next;

        Node() : next(nullptr){}

        Node(const ADT *obj ) : data(obj), next(nullptr) {}

    };

    Node *first;
    Node *last;
    size_t _size;

public:

    List() : first(nullptr), last(nullptr), _size(0) {}
    List(size_t size); // наверное лучше убрать
    ~List();

    void push_back(const ADT *right);
    void push_front(const ADT *right);
    void pop_back();
    void pop_front();

    void erase(size_t pos);
    void insert(size_t pos, const ADT *value);
    void emplace(size_t pos, const ADT *obj);

    size_t size() const;
    void clear();
    bool empty() const;

    Node* &operator[](size_t pos);
    const Node* operator[](size_t pos) const;
};

#endif // list_h
```

Г – Файл list.cpp

```
#include "list.h"

List::List(size_t size)
{
    Node *var = new Node();
    first = var;
    last = var;
    for (size_t i = 1; i < size-1; ++i)
    {
        Node *next = new Node();
        last->next = next;
        last = next;
```

```
    }
    _size = size;
}

List::~List()
{
    if (empty()) return;
    Node *var = first;
    if (var) return;
    else {
        while (var->next != last)
        {
            delete var;
            var = var->next;
            if (!var) return;
        }
        delete var;
    }
}


void List::push_back(const ADT *obj)
{
    Node *var = new Node(obj);
    if (empty())
    {
        first = var;
        last = var;
        ++_size;
        return;
    }
    last->next = var;
    last = var;
    ++_size;
}
void List::push_front(const ADT *obj)
{
    Node *var = new Node(obj);
    var->next = first;
    first = var;
    ++_size;
}


void List::pop_front()
{
    if (empty()) return;
    --_size;
    Node* val = first;
    first = first->next;
    delete val;
}

void List::pop_back()
```

```
{
    if (empty()) return;
    if (first == last)
    {
        pop_front();
        return;
    }
    Node* val = first;
    while (val->next != last) val = val->next;
    val->next = nullptr;
    delete last;
    last = val;
    --_size;
}


void List::erase(size_t pos)
{
    if (pos == 1) return pop_front();
    if (empty()) return;
    --_size;
    Node *var = first;
    for (size_t i = 1; i < pos-1; ++i)
    {
        var = var->next;
        if (!var) return;
    }

    Node *temp = var->next;
    //delete var->next;
    var->next = temp->next;

}

void List::insert(size_t pos, const ADT *value)
{
    if (pos > _size) std::runtime_error("Position more size");
    if (empty()) return push_back(value);
    ++_size;
    Node *var = first;
    for (size_t i = 1; i < pos-1; ++i)
    {
        var = var->next;
        if (!var) return;
    }
    Node *temp = var->next;
    Node *qwa = new Node(value);
    var->next = qwa;
    qwa->next = temp;
}

void List::emplace(size_t pos, const ADT *obj)
{
    if (empty()) return;
```

```
    if (pos > _size) std::runtime_error("Position more size");
    Node *var = first;
    for (size_t i = 0; i < pos-1; ++i)
    {
        var = var->next;
        if (!var) return;
    }
    var->data = obj;
}


size_t List::size() const { return _size; }

void List::clear()
{
    if (empty()) return;
    Node *var = first;
    if (var) return;
    else {
        while (var->next != last)
        {
            delete var;
            var = var->next;
            if (!var) return;
        }
        delete var;
    }
    _size = 0;
}

bool List::empty() const { return _size == 0; }


List::Node* &List::operator[](size_t pos)
{
    Node* var = this->first;
    for (size_t i = 0; i < pos; ++i)
    {
        if (var->next != nullptr) var = var->next;
    }
    Node* &ref = var;
    return ref;
}

const List::Node* List::operator[](size_t pos) const
{
    Node* var = this->first;
    for (size_t i = 0; i < pos; ++i)
    {
        if (var->next != nullptr) var = var->next;
        else if (i < pos) std::runtime_error("Error of position");
    }
    return var;
}
```

Г – Файл main_list.cpp

```
#include <iostream>

#include "list.h"
#include "new_integer.h"
#include "Adam_time.h"

#include <gtest/gtest.h>


TEST(TestGroupMane, TestName)
{
    ASSERT_TRUE(true);
}


int main()
{
    ::testing::InitGoogleTest();
    Integer var("32832183912938");
    ADT *test1 = &var;
    Integer *test2 = new Integer(21212312);
    List qwa;
    qwa.push_back(test1);
    qwa.push_back(test2);
    std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(const_cast<ADT*>(qwa[0]-
>data))).get_num_str()).c_str());

    EXPECT_FALSE(*dynamic_cast<Integer*>(const_cast<ADT*>(qwa[0]->data)) ==
*dynamic_cast<Integer*>(const_cast<ADT*>(qwa[1]->data)) );
    // тест на сравнение строк
    std::cout <<"########################################\n";

    ADT *test3 = new Adam_t(5,20,15);
    qwa.push_back(test3);
    // тест на хранимые данные в qwa[2]
    std::cout << "test: push_back "<<  std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]->data) << std::endl;

    std::cout <<"########################################\n";

    ADT *test4 = new Integer(12345);
    qwa.push_front(test4);
    // тест на хранимые данные в qwa[2]
    std::cout << "test: push_front "<< *(qwa[0]->data) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
```

```
        std::cout <<  (*qwa[i]->data) << std::endl;

    std::cout <<"########################################\n";

    qwa.pop_back();
    // тест на хранимые данные в qwa[2]
    std::cout << "test: pop_back "<< std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  *(qwa[i]->data) << std::endl;

    std::cout <<"########################################\n";
    qwa.pop_front();
    // тест pop_front
    std::cout << "test: pop_front "<< std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]->data) << std::endl;

    std::cout <<"########################################\n";

    // тест insert
    Adam_t qwer(9,12,11);
    ADT *test5 = &qwer;
    qwa.insert(1,test5);
    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(const_cast<ADT*>(qwa[1]-
>data))).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]->data) << std::endl;

    // тест size
    std::cout << "size = " << qwa.size() << std::endl;

    // тест erase
    std::cout <<"##################\n";
    qwa.erase(1);
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout << (*qwa[i]->data) << std::endl;

    std::cout << "new size = " << qwa.size() << std::endl;

    std::cout <<"########################################\n";

    // тест emplace
    ADT *test6 = new Adam_t(5,14,10);
    qwa.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(const_cast<ADT*>(qwa[0]-
>data))).get_Adam_year() );
    std::cout << "test: emaplce(0)\n";
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout << (*qwa[i]->data) << std::endl;

    // тест на empty и clear
    std::cout <<"########################################\n";
```

```
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';
    EXPECT_FALSE(qwa.empty());
    qwa.clear();
    EXPECT_TRUE(qwa.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';

    return RUN_ALL_TESTS();
}
```

Приложение Д

Д – Файл queue.h

```
#ifndef queue_h
#define queue_h

#include "ADT.h"

class Queue
{
    private:

        size_t _size;    // размер очереди
        size_t _quantity;// количество элементов
        ADT** arr;

        void resize(size_t new_size);

    public:

        Queue() = delete;
        Queue(size_t size);
        Queue(const Queue &right);
        ~Queue();

        size_t size() const;
        ADT* pop_back();
        void push_front(ADT *value);
        void erase(size_t pos);
        void emplace(size_t, ADT*);
        void insert(size_t pos, ADT *value);
        void clear();
        bool empty() const;

        const Queue &operator= (const Queue &right);
```

```
        ADT* &operator[](size_t pos);
        ADT* &begin(); // first
        ADT* &end();   // last

};

#endif // queue_h
```

Д – Файл queue.cpp

```
#include <iostream>
#include <algorithm>

#include "queue.h"


Queue::Queue(size_t size) : _size(size)
{
    arr = new ADT*[_size];
    for (size_t i = 0; i < _size; ++i)
    {
        arr[i] = new ADT;
    }
    _quantity = 0;
}

Queue::Queue(const Queue &right)
{
    if (arr != nullptr)
    {
        for (size_t i = 0; i < this->_size; ++i) delete [] arr[i];
        delete [] arr;
        arr = nullptr;
    }
    try
    {
        this->_size = right._size;
        this->_quantity = right._quantity;
        this->arr = new ADT* [this->_size];
        for (size_t i = 0; i < _quantity; ++i)
        {
            arr[i] = new ADT;
            arr[i] = right.arr[i];
        }
    }
    catch (...)
    {
        std::runtime_error("Error memory allocated");
        exit(1);
    }
}
```

```
Queue::~Queue()
{
    if (arr != nullptr)
    {
        for (size_t i = 0; i < this->_size; ++i) delete arr[i];
        delete [] arr;
    }

}

size_t Queue::size() const { return this->_quantity; }


void Queue::resize(size_t new_size)
{
    ADT **new_arr;
    try
    {
        new_arr = new ADT*[new_size];
        for (size_t i = 0; i < this->_quantity; ++i )
        {
            new_arr[i] = new ADT;
            new_arr[i] = arr[i];
        }
    }
    catch (...)
    {
        std::runtime_error("Error memory allocated");
        exit(1);
    }
    for (size_t i = 0; i < _size; ++i) delete [] arr[i];
    delete [] arr;
    arr = new_arr;
    this->_size = new_size;

}


ADT* Queue::pop_back()
{
    if (arr != nullptr && _quantity > 0)
    {
        ADT *val = arr[_quantity-1];
        --_quantity;
        return val;
    }
    else return nullptr;
}

void Queue::push_front(ADT *value)
{
    this->insert(0, value);
}
```

```
void Queue::emplace(size_t pos, ADT *value)
{
    arr[pos] = value;
}

void Queue::insert(size_t pos, ADT *value)
{
    if (empty())
    {
        arr[0] = value;
        ++_quantity;
    }
    else if (_quantity < _size)
    {
        ++_quantity;
        ADT *var = arr[pos];
        arr[pos] = value;
        ADT *qwa;
        for (size_t i = pos+1; i < _quantity; ++i)
        {
            qwa = arr[i];
            arr[i] = var;
            var = qwa;
        }
    }
    else if (_quantity == _size)
    {

        for (size_t i = 0; i < pos; ++i)
            arr[i+1] = arr[i];
        arr[0] = arr[_quantity-1];
        ADT *var = arr[pos];
        arr[pos] = value;
        ADT *qwa;
        for (size_t i = pos+1; i < _quantity; ++i)
        {
            qwa = arr[i];
            arr[i] = var;
            var = qwa;
        }
    }
}


void Queue::erase(size_t pos)
{
    --_quantity;
    for (size_t i = pos; i < _quantity; ++i)
        arr[i] = arr[i+1];
}

void Queue::clear()
{
    delete [] arr;
```

```
    arr = nullptr;
    _size = 0;
    _quantity = 0;
}
bool Queue::empty() const { return _quantity == 0; }

//const Queue &Queue::operator= (const ADT &right)

const Queue &Queue::operator= (const Queue& right)
{
    if (_size != right._size)
    {
        this->_quantity = right._quantity;
        if (arr != nullptr)
        {
            for (size_t i = 0; i < this->_size; ++i) delete [] arr[i];
            delete [] arr;
            arr = nullptr;
        }
        this->_size = right._size;
        try
        {
            this->arr = new ADT* [right._size];
            for (size_t i = 0; i < right._quantity; ++i)
            {
                arr[i] = new ADT;
            }
        }
        catch (...)
        {
            std::runtime_error("Error memory allocated");
            exit(1);
        }
    }
    for ( size_t i = 0; i < right._quantity; ++i )
        arr[i] = right.arr[i];
    return *this;
}
ADT* &Queue::operator[](size_t pos) { return arr[pos]; }
ADT* &Queue::begin() { return *arr; }
ADT* &Queue::end() { return *(arr + _quantity); }
```

Д – Файл main_queue.cpp

```
#include <iostream>

#include "queue.h"

#include "new_integer.h"
#include "Adam_time.h"
```

```
#include <gtest/gtest.h>


TEST(TestGroupMane, TestName)
{
    ASSERT_TRUE(true);
}


int main()
{
    ::testing::InitGoogleTest();
    Integer var("32832183912938");
    ADT *test1 = &var;
    Integer *test2 = new Integer(21212312);
    Queue qwa(3);
    qwa.push_front(test1);
    qwa.push_front(test2);
    std::string qwaqwa= "32832183912938";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[1])).get_num_str()).c_str());
    std::cout <<"########################################\n";

    ADT *test4 = new Integer(12345);
    qwa.push_front(test4);
    // тест на хранимые данные в qwa
    std::string sad= "12345";
    EXPECT_STREQ(sad.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa[0])).get_num_str()).c_str());
    std::cout << "test: push_front "<< *(qwa[0]) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";



    // тест на хранимые данные в qwa
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(qwa.pop_back())).get_num_str()).c_s
tr());
    std::cout << "test: pop_back " << *(qwa.pop_back()) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";

    Adam_t qwer(9,12,11);
    ADT *test5 = &qwer;
    qwa.insert(1,test5);
    // тест на хранимые данные в qwa
    EXPECT_EQ(6984, (*dynamic_cast<Adam_t*>(qwa[1])).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;

    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;
```

```
    std::cout <<"########################################\n";

    ADT *test8 = new Integer(5432);
    qwa.push_front(test8);
    // тест на хранимые данные в qwa[2]
    std::cout << "test: push_front "<< *(qwa[0]) << std::endl;
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout <<"########################################\n";
    std::cout << "size = " << qwa.size() << std::endl;
    // тест на size и capacity

    std::cout <<"###################\n";
    qwa.erase(1);
    // тест на хранимые данные в qwa[1]
    std::cout << "test: erase(1)\n";
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;

    std::cout << "new size = " << qwa.size() << std::endl;
    // тест на size и capacity

    std::cout <<"########################################\n";

    ADT *test6 = new Adam_t(5,14,10);
    qwa.emplace(0, test6);
    EXPECT_EQ(770, (*dynamic_cast<Adam_t*>(qwa[0])).get_Adam_year() ); // тест на
хранимые данные в qwa[0]
    std::cout << "test: emaplce(0)\n";
    for(size_t i = 0; i< qwa.size(); ++i)
        std::cout <<  (*qwa[i]) << std::endl;


    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';
    EXPECT_FALSE(qwa.empty());
    qwa.clear();
    EXPECT_TRUE(qwa.empty());
    std::cout << "test: clear\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';
    return RUN_ALL_TESTS();
}
```

Приложение E

E – Файл tree.h

```
#ifndef tree_h
#define tree_h

#include "ADT.h"

class Tree
{
    private:
    //public:
        struct Tnode
        {
                const ADT *data;    // поле данных
                size_t pos;         // префиксная позиция в дереве (иначе говоря
ключ)
                Tnode *left;        // левый потомок
                Tnode *right;       // правый потомок
                Tnode *parent;      // родитель
                Tnode() : data(nullptr), pos(0), left(nullptr), right(nullptr) {}
                Tnode(const ADT *data, const size_t pos = 0)
                {
                    this->data = data;
                    this->pos = pos;
                    this->left = nullptr;
                    this->right = nullptr;
                }
                ~Tnode() = default;
        };

        Tnode *find(Tnode *tree, const size_t pos) const;
        // находит элемент, с заданной позицией
        Tnode *find_C(const size_t pos) const;
        // находит элемент с ключем, следующим за данным числом
        void delete_tree(Tnode* tree); // очистка памяти
        Tnode *tree;
        size_t _size;

    public:

        Tree();
        ~Tree();
        void insert(const size_t pos, const ADT *data);
        void erase(const size_t pos);
        void print(Tnode *tree, std::ostream &out) const;
        const bool empty() const;
        const size_t size() const;
        Tnode *get_tree();
        const ADT* operator[](const size_t pos) const;

};

#endif // tree_h
```

Е – Файл tree.cpp

```
#include "tree.h"

Tree::Tree() : tree(0), _size(0) {}
Tree::~Tree()
{
    if (tree != nullptr) delete_tree(this->tree);
}

void Tree::insert(const size_t pos, const ADT *data)
{
    Tnode *var = new Tnode(data, pos);
    Tnode *p1;
    Tnode *p2;
    p1 = tree;
    do
    {
        p2 = p1;
        if (p1 != nullptr && pos % 2 == 1) p1 = p1->left;
        else if (p1 != nullptr) p1 = p1->right;
    }while (p1 != nullptr);

    var->parent = p2;
    if (p2 == nullptr) tree = var;
    else
    {
        if ( pos %2==1  ) p2->left = var;
        else p2->right = var;
    }
    ++_size;
}
void Tree::erase(const size_t pos)
{
    Tnode *p1;
    Tnode *p2;
    Tnode *var = this->find(this->tree, pos);
    if (var->left == nullptr || var->right == nullptr) p1 = var;
    else p1 = find_C(var->pos);

    if (p1->left != nullptr) p2 = p1->left;
    else p2 = p1->right;

    if (p2 != nullptr) p2->parent = p1->parent;

    if (p1->parent == nullptr) tree = p2;
    else
    {
        if (p1 == (p1->parent)->left) (p1->parent)->left = p2;
        else (p1->parent)->right = p2;
    }
    --_size;
}
```

```
const bool Tree::empty() const
{
    return this->_size == 0;
}

const size_t Tree::size() const
{
    return _size;
}

Tree::Tnode *Tree::get_tree()
{
    return tree;
}

void Tree::print(Tnode *tree, std::ostream &out) const
{
    if (tree != nullptr)
    {
        print(tree->left, out);
        out << *(tree->data) << std::endl;
        print(tree->right, out);
    }
}

const ADT *Tree::operator[](const size_t pos) const
{
    Tnode *var = find(this->tree, pos);
    if (var == nullptr)
    {
        std::runtime_error("Not real");
        exit(1);
    }
    return var->data;
}

Tree::Tnode *Tree::find(Tnode *tree, const size_t pos) const
{
    if (tree == nullptr || pos == tree->pos) return tree;
    if (pos % 2 == 0) return find(tree->right, pos);
    else return find(tree->left, pos);
}


Tree::Tnode *Tree::find_C(const size_t pos) const
{
    Tnode *p1 = find(tree, pos);
    Tnode *p2;
    if ( p1 == nullptr ) return nullptr;
    if (p1->right != nullptr)
    {
        while (p1->left != nullptr)
        {
            p1 = p1->left;
```

```
        }
        return p1;
    }
    p2 = p1->parent;
    while (p2 != nullptr && p1 == p2->right)
    {
        p1 = p2;
        p2 = p2->parent;
    }
    return p2;

}

void Tree::delete_tree(Tnode* tree)
{
    if (tree != nullptr)
    {
        delete_tree(tree->left);
        delete tree;
        delete_tree(tree->right);
    }
    tree = nullptr;
}
```

Е – Файл main_tree.cpp

```
#include <iostream>

#include "tree.h"
#include "new_integer.h"
#include "Adam_time.h"

#include <gtest/gtest.h>

TEST(TestGroupMane, TestName)
{
    ASSERT_TRUE(true);
}

int main()
{
    ::testing::InitGoogleTest();
    Tree qwa;
    ADT *test1 = new Integer("123456");
    ADT *test2 = new Integer("8998");
    ADT *test3 = new Integer("5555");
    ADT *test4 = new Integer(444);

    qwa.insert(4,test1);
    qwa.insert(1,test2);
```

```
    qwa.insert(12,test3);
    qwa.insert(6,test4);

    std::string qwaqwa= "123456";
    EXPECT_STREQ(qwaqwa.c_str(),
static_cast<std::string>((*dynamic_cast<Integer*>(const_cast<ADT*>(qwa[4]))).get_num_
str()).c_str());

    // тест insert
    Adam_t qwer(9,12,11);
    ADT *test5 = &qwer;
    qwa.insert(2,test5);
    EXPECT_EQ(6984,
(*dynamic_cast<Adam_t*>(const_cast<ADT*>(qwa[2]))).get_Adam_year() );
    std::cout << "test: insert(1,6984) " << std::endl;
    qwa.print(qwa.get_tree(), std::cout);
    // тест erase
    std::cout <<"###################\n";
    qwa.erase(6);
    std::cout << "test: erase(1)\n";

    qwa.print(qwa.get_tree(), std::cout);

    std::cout <<"########################################\n";

    // тест на empty
    std::cout <<"########################################\n";
    std::cout << "test: empty: ";
    std::cout << std::boolalpha << qwa.empty() << '\n';
    EXPECT_FALSE(qwa.empty());
    return RUN_ALL_TESTS();
}
```