МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Наследование в С++»**

*Вариант № 9*

Выполнил:

Студент группы

Ктбо2-8

Жалнин Д. И.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# Оглавление

[1 ФОРМУЛИРОВКА ЗАДАНИЯ 3](#_Toc58276144)

[2 Спецификация классов 4](#_Toc58276145)

[2.1 Класс Triad 4](#_Toc58276146)

[2.2 Класс Date 6](#_Toc58276147)

[2.3 Класс Time 7](#_Toc58276148)

[2.4 Класс Pare 8](#_Toc58276149)

[2.5 Класс Memories 8](#_Toc58276150)

[2.6 Класс Interaction 9](#_Toc58276151)

[2.7 Класс Exceptions 10](#_Toc58276152)

[2.8 Класс TriadNumber 11](#_Toc58276153)

[2.9 Класс SpecialRoundType 11](#_Toc58276154)

[3 Диаграмма классов 12](#_Toc58276155)

[4 Листинг программы 13](#_Toc58276156)

[4.1 Triad.h 13](#_Toc58276157)

[4.2 Triad.cpp 13](#_Toc58276158)

[4.3 Time.h 15](#_Toc58276159)

[4.4 Date.h 15](#_Toc58276160)

[4.5 Date.cpp 16](#_Toc58276161)

[4.6 Pare.h 17](#_Toc58276162)

[4.7 Pare.cpp 17](#_Toc58276163)

[4.8 Memories.h 18](#_Toc58276164)

[4.9 Memories.cpp 18](#_Toc58276165)

[4.10 Interaction.h 19](#_Toc58276166)

[4.11 Interaction.cpp 20](#_Toc58276167)

[4.12 Exceptions.h 23](#_Toc58276168)

[4.13 SpecialRoundType.h 23](#_Toc58276169)

[4.14 TriadNumber.h 23](#_Toc58276170)

[4.15 lab2.cpp 23](#_Toc58276171)

# 1 ФОРМУЛИРОВКА ЗАДАНИЯ

Согласно варианту задания, требуется создать класс Triad (тройка) и на его основе классы Date (дата) и Time (время) с методами увеличения и уменьшения на 1 каждого из значений тройки: год/час, месяц/минуты, день/секунды. В дополнительном классе Memories создать массив пар (дата-время) объектов этих классов: 01.09.2020 – 8.00.00, …. Количество пар задается конструктором класса Memories.

При выполнении работы необходимо: − разработать соответствующие классы, конструкторы, поля и методы − поля класса сделать закрытыми; для чтения и изменения их значений определить открытые методы; – предусмотреть во всех вариантах консольный ввод данных для создания объектов и консольный вывод результатов. Запрещено пользоваться контейнерами STL, за исключением string.

# 2 Спецификация классов

## 2.1 Класс Triad

Публичные поля:

**Triad(int maxA, int maxB, int maxC, int a, int b, int c)**

// Конструктор, принимающий на вход максимальные значения «триады» и

// значения, которые необходимо установить при создании класса

// В следующие 5 методов передается **enum TriadNumber**, который указывает – // какое число (1, 2 или 3) в «триаде» необходимо изменить

**int getAndIncrement(TriadNumber triadNumber)**

// Метод, который сперва возвращает значение, а затем инкрементирует

**int getAndDecrement(TriadNumber triadNumber)**

// Метод, который сперва возвращает значение, а затем декрементирует

**int incrementAndGet(TriadNumber triadNumber)**

// Метод, который инкрементирует значение, а затем возвращает

**int decrementAndGet(TriadNumber triadNumber)**

// Метод, который декрементирует значение, а затем возвращает

**int get(TriadNumber triadNumber)**

// Метод, возвращающий значение, находящееся в позиции «**triadNumber**»

**void clear()**

// Метод обнуления текущих значений «триады»

**bool isOverloaded()**

// Метод, проверяющий переполнение значений «триады»

// (значения больше максимальных)

Приватные поля:

**bool overloadedFlag**

// Флаг достижения «триадой» максимального значения

**int maxA**

// Максимальное значение первого элемента «триады»

**int maxB**

// Максимальное значение второго элемента «триады»

**int maxC**

// Максимальное значение третьего элемента «триады»

**int a**

// Текущее значение первого элемента «триады»

**int b**

// Текущее значение второго элемента «триады»

**int c**

// Текущее значение третьего элемента «триады»

**int\* getInt(TriadNumber triadNumber)**

// Метод, возвращающий указатель на поле **a**, **b** или **c**, в зависимости от

// переданного в него значение **triadNumber**

**bool check(int a, int b, int c)**

// Метод, сравнивающий максимальные и текущие значения «триады»

// Если текущие значение больше, чем максимальные, возникает исключение

**void round(int \*a, int \*b, int \*c)**

// Метод, изменяющий текущие значения «триады» при достижении одной из них // максимума или значения «0»

«Protected» поля:

**virtual void specialRound(int\* a, int\* b, int\* c, SpecialRoundType flag)**

/\* Виртуальный метод содержащий «специальный» алгоритм действий, особый для каждого реализуемого типа триад, который несколько раз может вызваться из метода **round**:

1. При достижении одного из чисел максимального значения с помощью операции декремента
2. При достижении одного из чисел 0 с мощью операции декремента
3. При достижении одного из чисел максимального значения с помощью операции инкремента
4. При достижении одного из чисел 0 с мощью операции инкремента

Тип операции указывается с помощью **enum SpecialRoundType**, который передается в метод 4м параметром \*/

## 2.2 Класс Date

Класс **Date** наследуется от класса **Triad**. Для реализации наследования в коде, при объявлении класса была прописана следующая конструкция:

**class Date :**

**public Triad**

Публичные поля:

**static const int MAX\_A = 31**

// Константное максимальное значение первой переменной

// (в данной реализации отвечает за день)

**static const int MAX\_B = 12**

// Константное максимальное значение второй переменной

// (в данной реализации отвечает за месяц)

**static const int MAX\_C = 10000**

// Константное максимальное значение третьей переменной

// (в данной реализации отвечает за год)

**Date(int a, int b, int c) : Triad(MAX\_A, MAX\_B, MAX\_C, a, b, c) {**

**}**

/\* При создании класса **Date** будет вызван конструктор класса **Triad**. В этот конструктор мы передаем переменные **a**, **b**, **c** из конструктора **Date** и максимальные значения для них \*/

**void specialRound(int\* a, int\* b, int\* c, SpecialRoundType flag) override**

/\* В данном методе происходит сложное взаимодействие с переменными класса с особыми условиями «округления» для отдельных месяцев и високосных годов. Рассмотреть подробнее можно в листинге программы\*/

## 2.3 Класс Time

Класс **Time** наследуется от класса **Triad**. Для реализации наследования в коде, при объявлении класса была прописана следующая конструкция:

**class Time :**

**public Triad**

Публичные поля:

**static const int MAX\_A = 59**

// Константное максимальное значение первой переменной

// (в данной реализации отвечает за секунды)

**static const int MAX\_B = 59**

// Константное максимальное значение второй переменной

// (в данной реализации отвечает за минуты)

**static const int MAX\_C = 23**

// Константное максимальное значение третьей переменной

// (в данной реализации отвечает за часы)

**Time(int a, int b, int c) : Triad(MAX\_A, MAX\_B, MAX\_C, a, b, c) {**

**}**

/\* При создании класса **Time** будет вызван конструктор класса **Triad**. В этот конструктор мы передаем переменные **a**, **b**, **c** из конструктора **Time** и максимальные значения для них \*/

## 2.4 Класс Pare

Класс представляет собой объект с полями «ключ-значение» с типом **Triad**.

Публичные поля:

**Pare()**

/\* Стандартный конструктор класса на случай, если нужно сперва создать объект Pare, а затем внести в него значения\*/

**Pare(Triad\* key, Triad\* value)**

/\* Конструктор, принимающий на вход два указателя на объекты типа **Triad** и присваивающий приватным полям **key** и **value** эти значения \*/

**Triad\* getKey()**

// «Геттер» приватной переменной **key**

**Triad\* getValue()**

// «Геттер» приватной переменной **value**

**void setKey(Triad\* triad)**

// «Сеттер» приватной переменной **key**

**void setValue(Triad\* triad)**

// «Сеттер» приватной переменной **value**

**~Pare();**

// Деструктор, удаляющий объекты **key** и **value** с помощью **delete**

Приватные поля:

**Triad\* key = NULL**

**Triad\* value = NULL**

/\* Указатели (ключ и значение пары) на объекты класса Triad \*/

## 2.5 Класс Memories

**Memories(int size)**

/\* Конструктор, принимающий на вход размерность массива **pareArray** и инициализирующий его. Так же значение **size** присваивается переменной **maxSize**\*/

**Pare\* getPare(int index)**

/\* Метод, возвращающий указатель на объект **Pare**, хранимый по индексу **index** \*/

**void setPare(int index, Triad\* t1, Triad\* t2)**

/\* Метод, который создает объект **Pare** и записывает указатель на него в массив по указанному индексу \*/

**void setPare(int index, Pare\* pare)**

/\* Метод, который записывает указатель на **Pare** в массив по указанному индексу \*/

**void removePare(int index)**

/\* Метод, удаляющий объект **Pare**, хранимый по переданному индексу, и заменяющий указатель объекта в массиве на **NULL** \*/

**int getMaxSize()**

// «Геттер» приватной переменной **maxSize**

**~Memories()**

// Деструктор, циклично удаляющий все элементы массива через **delete**

Приватные поля:

**Pare\*\* pareArray**

// Массив указателей на объекты класса **Pare**

**int maxSize**

// Размер массива **pareArray**

## 2.6 Класс Interaction

Данный класс является реализацией класса-обертки, с помощью которого происходит взаимодействие пользователя с интерфейсом программы. Данный класс циклически вводит сообщения с консоли пользователя, а также оповещает его о текущем состоянии работы программы. В конце своей работы класс очищает созданный в нем объект класса **memories** с помощью **delete**. Демонстрация пользовательского интерфейса расположена на рисунке 1.



*Рисунок 1 – интерфейс пользователя в командной строке*

## 2.7 Класс Exceptions

Класс представляет из себя контейнер наследованных от exception классов:

class TriadDataException : public std::exception {

public:

const char\* what() const noexcept override {

return "Bad data";

}

};

class MemoriesException : public std::exception {

private:

const char\* reason;

public:

MemoriesException(const char\* reason) {

this->reason = reason;

}

const char\* what() const noexcept override {

return reason;

}

};

**TriadDataException** используется в классе **Triad**.

**MemoriesException** используется в классе **Memories**.

Выброшенные исключения ловятся в классе Interaction, оповещая об этом пользователя, но не прерывая работу программы.

## 2.8 Класс TriadNumber

Класс-перечисление, используемый в работе программы для указания номера переменной в классе **Triad**.

enum TriadNumber

{

FIRST,

SECOND,

THIRD

};

## 2.9 Класс SpecialRoundType

enum SpecialRoundType {

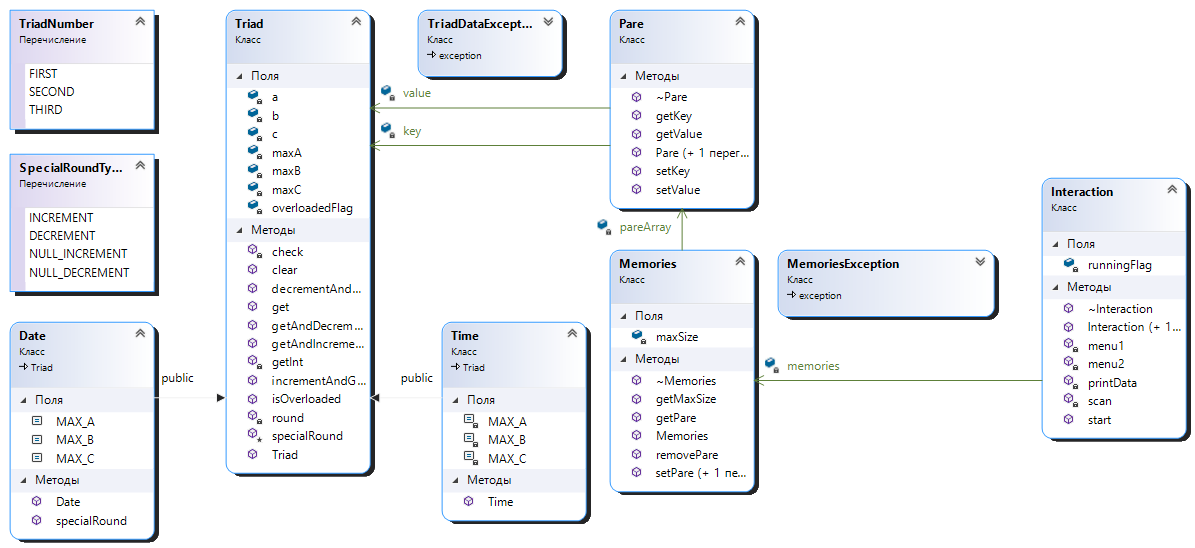
INCREMENT, DECREMENT, NULL\_INCREMENT, NULL\_DECREMENT

};

Класс-перечисление, элементы которого передаются в метод **specialRound:**

1. **DECREMENT** – При достижении одного из чисел максимального значения с помощью операции декремента
2. **NULL\_DECREMENT** При достижении одного из чисел 0 с мощью операции декремента
3. **INCREMENT** При достижении одного из чисел максимального значения с помощью операции инкремента
4. **NULL\_INCREMENT** При достижении одного из чисел 0 с мощью операции инкремента

# 3 Диаграмма классов



*Рисунок 1 – UML диаграмма классов.*

# 4 Листинг программы

## 4.1 Triad.h

#pragma once

#include "TriadNumber.h"

#include "SpecialRoundType.h"

class Triad

{

public:

Triad(int maxA, int maxB, int maxC, int a, int b, int c);

int getAndIncrement(TriadNumber triadNumber);

int getAndDecrement(TriadNumber triadNumber);

int incrementAndGet(TriadNumber triadNumber);

int decrementAndGet(TriadNumber triadNumber);

int get(TriadNumber triadNumber);

void clear();

bool isOverloaded();

private:

bool overloadedFlag;

int maxA;

int maxB;

int maxC;

int a;

int b;

int c;

int\* getInt(TriadNumber triadNumber) {

switch (triadNumber)

{

case FIRST:

return &a;

case SECOND:

return &b;

case THIRD:

return &c;

}

}

bool check(int a, int b, int c);

void round(int \*a, int \*b, int \*c);

protected:

virtual void specialRound(int\* a, int\* b, int\* c, SpecialRoundType flag) {};

};

## 4.2 Triad.cpp

#include "Triad.h"

#include "Exceptions.h"

Triad::Triad(int maxA, int maxB, int maxC, int a, int b, int c) {

this->maxA = maxA;

this->maxB = maxB;

this->maxC = maxC;

if (check(a, b, c)) {

this->a = a;

this->b = b;

this->c = c;

}

else {

throw TriadDataException();

}

}

bool Triad::isOverloaded() {

return overloadedFlag;

}

int Triad::getAndIncrement(TriadNumber triadNumber) {

int\* n = getInt(triadNumber);

int nV = \*n;

(\*n)++;

round(&a, &b, &c);

return (nV);

}

int Triad::getAndDecrement(TriadNumber triadNumber) {

int\* n = getInt(triadNumber);

int nV = \*n;

(\*n)--;

round(&a, &b, &c);

return (nV);

}

int Triad::incrementAndGet(TriadNumber triadNumber) {

int\* n = getInt(triadNumber);

(\*n)++;

round(&a, &b, &c);

return (\*n);

}

int Triad::decrementAndGet(TriadNumber triadNumber) {

int\* n = getInt(triadNumber);

(\*n)--;

round(&a, &b, &c);

return (\*n);

}

int Triad::get(TriadNumber triadNumber) {

return \*getInt(triadNumber);

}

bool Triad::check(int a,int b,int c) {

return (a <= maxA && b <= maxB && c <= maxC);

}

void Triad::round(int\* a, int\* b, int\* c) {

if (\*a > maxA) {

specialRound(a, b, c, INCREMENT);

(\*b)++;

(\*a)=0;

specialRound(a, b, c, NULL\_INCREMENT);

}

if (\*b > maxB) {

specialRound(a, b, c, INCREMENT);

(\*c)++;

(\*b)=0;

specialRound(a, b, c, NULL\_INCREMENT);

}

if (\*c > maxC) {

specialRound(a, b, c, INCREMENT);

\*c = 0;

overloadedFlag = true;

specialRound(a, b, c, NULL\_INCREMENT);

}

specialRound(a, b, c, NULL\_DECREMENT);

if (\*a < 0) {

(\*b)--;

\*a = maxA;

specialRound(a, b, c, DECREMENT);

}

specialRound(a, b, c, NULL\_DECREMENT);

if (\*b < 0) {

(\*c)--;

\*b = maxB;

specialRound(a, b, c, DECREMENT);

}

specialRound(a, b, c, NULL\_DECREMENT);

if (\*c < 0) {

specialRound(a, b, c, DECREMENT);

(\*c) = maxC;

}

}

void Triad::clear() {

a = 0;

b = 0;

c = 0;

overloadedFlag = false;

}

## 4.3 Time.h

#pragma once

#include "Triad.h"

class Time :

public Triad

{

private:

static const int MAX\_A = 59;

static const int MAX\_B = 59;

static const int MAX\_C = 23;

public:

Time(int a, int b, int c) : Triad(MAX\_A,MAX\_B,MAX\_C,a,b,c) {

}

};

## 4.4 Date.h

#pragma once

#include "Triad.h"

#include "SpecialRoundType.h"

class Date :

public Triad

{

private:

public:

static const int MAX\_A = 31;

static const int MAX\_B = 12;

static const int MAX\_C = 10000;

Date(int a, int b, int c) : Triad(MAX\_A, MAX\_B, MAX\_C, a, b, c) {

}

void specialRound(int\* a, int\* b, int\* c, SpecialRoundType flag) override;

};

## 4.5 Date.cpp

#include "Date.h"

void Date::specialRound(int\* a, int\* b, int\* c, SpecialRoundType flag) {

if (flag == DECREMENT) {

if (\*a >= 31)

switch (\*b) {

case 2:

if (\*c % 4 == 0 && (\*c % 100 != 0 || \*c % 100 == 0)) {

(\*a) = 29;

}

else {

(\*a) = 28;

case 4:

case 6:

case 9:

case 11:

(\*a) = 30;

default:

break;

}

}

}

else if (flag == INCREMENT) {

switch (\*b)

{

case 2:

if (\*a == 29 && !(\*c % 4 == 0 && (\*c % 100 != 0 || \*c % 100 == 0))) {

(\*a) += 1;

}

case 4:

case 6:

case 9:

case 11:

if (\*a == 30) {

(\*a) += 2;

}

default:

break;

}

}

else if (flag == NULL\_DECREMENT) {

if (\*a <= 0) {

\*a = -1;

}

if (\*b <= 0) {

\*b = -1;

}

if (\*c <= 0) {

\*c = -1;

}

}

else if (flag == NULL\_INCREMENT) {

if (\*a <= 0) {

\*a = 1;

}

if (\*b <= 0) {

\*b = 1;

}

if (\*c <= 0) {

\*c = 1;

}

}

}

## 4.6 Pare.h

#pragma once

#include "Triad.h"

#include <cstddef>

class Pare {

private:

Triad\* key = NULL;

Triad\* value = NULL;

public:

Pare();

Pare(Triad\* key, Triad\* value);

Triad\* getKey();

Triad\* getValue();

void setKey(Triad\* triad);

void setValue(Triad\* triad);

~Pare();

};

## 4.7 Pare.cpp

#include "Pare.h"

Pare::Pare() {}

Pare::Pare(Triad\* key, Triad\* value) {

this->key = key;

this->value = value;

}

Triad\* Pare::getKey() {

return key;

}

Triad\* Pare::getValue() {

return value;

}

void Pare::setKey(Triad\* triad) {

key = triad;

}

void Pare::setValue(Triad\* triad) {

value = triad;

}

Pare::~Pare() {

if (key)

delete key;

if (value)

delete value;

key = NULL;

value = NULL;

}

## 4.8 Memories.h

#pragma once

#include "Pare.h"

class Memories {

private:

Pare\*\* pareArray;

int maxSize;

public:

Memories(int size);

Pare\* getPare(int index);

void setPare(int index, Triad\* t1, Triad\* t2);

void setPare(int index, Pare\* pare);

void removePare(int index);

int getMaxSize();

~Memories();

};

## 4.9 Memories.cpp

#include "Memories.h"

#include "Exceptions.h"

Memories::Memories(int size) {

if (size <= 0) {

throw MemoriesException("Incorrect size!");

}

this->maxSize = size;

pareArray = new Pare \* [size] {NULL};

}

Pare\* Memories::getPare(int index) {

if (index < 0 || index >=maxSize) {

throw MemoriesException("Incorrect index!");

}

Pare\* p = pareArray[index];

return pareArray[index];

}

void Memories::setPare(int index, Triad\* t1, Triad\* t2) {

if (index < 0 || index >= maxSize) {

throw MemoriesException("Incorrect index!");

}

this->setPare(index, new Pare(t1, t2));

}

void Memories::setPare(int index, Pare\* pare) {

if (index < 0 || index >= maxSize) {

throw MemoriesException("Incorrect index!");

}

if (pareArray[index] != NULL) {

delete pareArray[index];

}

pareArray[index] = pare;

}

void Memories::removePare(int index) {

if (index < 0 || index >= maxSize) {

throw MemoriesException("Incorrect index!");

}

delete pareArray[index];

pareArray[index] = NULL;

}

Memories::~Memories() {

for (int i = 0; i < maxSize; i++) {

if (pareArray[i])

delete pareArray[i];

pareArray[i] = NULL;

}

}

int Memories::getMaxSize() {

return maxSize;

}

## 4.10 Interaction.h

#pragma once

#include "Memories.h"

#include <cstddef>

class Interaction

{

private:

Memories\* memories = NULL;

void menu1();

bool menu2(int x);

int scan();

void printData();

bool runningFlag = true;

public:

Interaction(Memories\* memories);

Interaction() {}

~Interaction() { if (memories) delete memories; }

void start();

};

## 4.11 Interaction.cpp

#include "Interaction.h"

#include <iostream>

#include <string>

#include "Memories.h"

#include "Date.h"

#include "Time.h"

#include "Exceptions.h"

Interaction::Interaction(Memories\* memories) {

this->memories = memories;

}

void Interaction::menu1() {

std::cout << "1 - Create \"Memories\" \n";

std::cout << "2 - Create \"Pare\" and put into \"Memories\" \n";

std::cout << "3 - increment\n";

std::cout << "4 - decrement\n";

std::cout << "0 - exit\n";

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

printData();

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

}

bool Interaction::menu2(int x) {

bool incrementFlag = false;

switch (x)

{

case (0):

return false;

case (1):

std::cout << "size: ";

int size;

size = scan();

if (memories) {

delete memories;

}

memories = new Memories(size);

break;

case (2):

if (memories) {

std::cout << "Pare #1 (3 nums, Date, DD-MM-YYYY) - ";

int a = scan();

int b = scan();

int c = scan();

Date\* date = new Date(a,b,c);

std::cout << "Pare #1 (3 nums, Timme, HH-MM-SS) - ";

c = scan();

b = scan();

a = scan();

Time\* time = new Time(a,b,c);

Pare\* p = new Pare(date, time);

std::cout << "Index - ";

int index = scan();

memories->setPare(index, p);

std::cout << "\nSuccessful!\n";

}

else {

std::cout << "\"Memories\" == NULL !\n";

}

break;

case (3):

incrementFlag = true;

case (4):

if (memories) {

int type;

int position;

int index;

TriadNumber enumPosition;

std::cout << "Index > ";

index = scan();

if (!memories->getPare(index)) {

std::cout << "\nPare == NULL!\n";

break;

}

std::cout << "1 - Date, 2 - Time\n";

type = scan();

if (type == 1)

std::cout << "1 - Day, 2 - Month, 3 - Year\n";

else if (type == 2)

std::cout << "1 - Seconds, 2 - Minutes, 3 - Hours\n";

else {

std::cout << "Incorrect type!\n";

break;

}

position = scan();

if (position == 1)

enumPosition = FIRST;

else if (position == 2)

enumPosition = SECOND;

else if (position == 3)

enumPosition = THIRD;

else {

break;

}

if (incrementFlag) {

if (type == 1) {

memories->getPare(index)->getKey()->incrementAndGet(enumPosition);

}

else {

memories->getPare(index)->getValue()->incrementAndGet(enumPosition);

}

}

else {

if (type == 1) {

memories->getPare(index)->getKey()->decrementAndGet(enumPosition);

}

else {

memories->getPare(index)->getValue()->decrementAndGet(enumPosition);

}

}

}

else {

std::cout << "\"Memories\" == NULL !\n";

}

break;

default:

break;

}

return true;

}

int Interaction::scan() {

int x;

std::cin >> x;

return x;

}

void Interaction::printData()

{

if (memories == NULL) {

std::cout << "MEMORIES == NULL\n";

}

else {

for (int i = 0; i < memories->getMaxSize(); i++) {

Pare\* p = memories->getPare(i);

if (p) {

Triad\* t1 = p->getKey();

Triad\* t2 = p->getValue();

if (t1 && t2) {

std::cout << i << ") ";

printf\_s("%2.2d-%2.2d-%4.4d ", t1->get(FIRST), t1->get(SECOND), t1->get(THIRD));

printf\_s("%2.2d:%2.2d:%2.2d\n", t2->get(THIRD), t2->get(SECOND), t2->get(FIRST));

}

else {

std::cout << i << ") NULL\n";

}

}

else {

std::cout << i << ") NULL\n";

}

}

}

}

void Interaction::start() {

while (runningFlag) {

menu1();

try {

runningFlag = menu2(scan());

}

catch (MemoriesException &e) {

std::cout << "/n" << e.what() << "\n";

}

catch (TriadDataException& e1) {

std::cout << "/n" << e1.what() << "\n";

}

std::cout << "\n";

}

}

## 4.12 Exceptions.h

#pragma once

#include "exception"

class TriadDataException : public std::exception {

public:

const char\* what() const noexcept override {

return "Bad data";

}

};

class MemoriesException : public std::exception {

private:

const char\* reason;

public:

MemoriesException(const char\* reason) {

this->reason = reason;

}

const char\* what() const noexcept override {

return reason;

}

};

## 4.13 SpecialRoundType.h

#pragma once

enum SpecialRoundType {

INCREMENT, DECREMENT, NULL\_INCREMENT, NULL\_DECREMENT

};

## 4.14 TriadNumber.h

#pragma once

enum TriadNumber

{

FIRST,

SECOND,

THIRD

};

## 4.15 lab2.cpp

#include <vld.h>

#include <iostream>

#include "Memories.h"

#include "Time.h"

#include "Date.h"

#include "TriadNumber.h"

#include "Interaction.h"

int main()

{

Interaction\* interaction = new Interaction();

interaction->start();

delete interaction;

}