**Министерство образования Республики Беларусь**

**Учреждение образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

**Отчет**

По дисциплине: Проектирование программного обеспечения в интеллектуальных системах

Лабораторная работа №1

Выполнил: Липский Р. В.,

гр. 121701

Проверил: Никифоров С. А.

**Минск 2022**

**Цель**:изучить основы объектно-ориентированного программирования на языке программирования С++

**Задание:** Реализовать на языке С++ программу, написать и сгенерировать документацию при помощи doxygen. Для возможности тестирования классов написать тестовую программу с меню или набор unit-тестов. В случае написания unit-тестов необходимо проверить не менее 30 тестов случаев с использованием библиотеки GoogleTest.

**Индивидуальное задание:** Описать класс, реализующий тип данных "Вещественная матрица". Класс должен реализовывать следующие возможности:

∙ матрица произвольного размера с динамическим выделением памяти;

∙ пре- и постинкремент (++), пре- и постдекремент (--);

∙ сложение двух матриц (операторы +, +=);

∙ сложение матрицы с числом (операторы +, +=);

∙ вычитание двух матриц (операторы -, -=);

∙ вычитание из матрицы числа (операторы -, -=);

∙ произведение двух матриц (оператор \*);

∙ произведение матрицы на число (операторы \*, \*=);

∙ деление матрицы на число (операторы /, /=);

∙ возведение матрицы в степень (оператор ^, ^=);

∙ вычисление детерминанта;

∙ вычисление нормы;

**Ход выполнения:**

*1. Реализация класса ltrx::Matrix*

template<typename T>class Matrix {private: std::vector<std::vector<T>> matrix; size\_t width; size\_t height;

[[nodiscard]] std::pair<int, int> size() const { return {width, height};}

void set(size\_t horizontalIndex, size\_t verticalIndex, T value) {  
 matrix[horizontalIndex][verticalIndex] = value;  
}

T get(size\_t horizontalIndex, size\_t verticalIndex) const {  
 return matrix[horizontalIndex][verticalIndex];  
}

Для хранения матрицы было решено использовать стандартный контейнер std::vector<std::vector<T>>, где T — тип элементов матрицы. Также класс хранит информацию о собственном размере в переменных типа size\_t. Созданы методы ltrx::Matrix::size(), который возвращает пару std::pair<size\_t, size\_t>, первый элемент которой означает длину матрицы, а второй — ширину. Методы get(size\_t line, size\_t row) и set(size\_t, size\_t, T) получают и устаналивают элементы матрицы соответственно.

2. *Реализация класса ltrx::MatrixError*

class MatrixError : public std::runtime\_error {public: explicit MatrixError(std::string msg) : std::runtime\_error(msg) { }};

Для удобной обработки ошибок в работе, создан класс ltrx::MatrixError, наследующий класс std::runtime\_error, стандартное исключение из библиотеки stdexcept.h. Добавлен конструктор, принимающий на вход строук (std::string) для вывода сообщения с подробностями об ошибке.

Модификатор explicit используется в конструкторе для предотвращения неявного приведения типов.

*3. Реализация класса ltrx::Matrix::AbstractFunctor, методов ltrx::Matrix::map(ltrx::Matrix::AbstractFunctor&) и ltrx::Matrix::mapLine(size\_t line, ltrx::Matrix::AbstractFunctor&), статических классов ltrx::Matrix::IncFunctor и ltrx::Matrix::DecFunctor.*

template <typename K>class AbstractFunctor {public: virtual K invoke(T& elem) = 0;};

Данный класс необходим для включения в класс возможности использовать элементы функционального стиля программирования. Он содержит один виртуальный метод invoke(T) → K, который должен производить некоторое действие над элементом матрицы и возвращать результат действия.

template<typename K>  
Matrix<K> map(AbstractFunctor<K>& f) {  
 Matrix<K> result = Matrix<K>(width, height);  
 for (int i = 0; i < width; i++) {  
 for (int j = 0; j < height; j++) {  
 std::cout << matrix[i][j] << " ";  
 result.set(i, j, f.invoke(matrix[i][j]));  
 }  
 }  
 return result;  
}

Метод ltrx::Matrix::map(…) производит некоторое действие над каждым элементом матрицы, создавая матрицу такого же размера, каждый элемент которой равен результату некоторого действия над соответствующим элементом изначальной матрицы.

template<typename K>  
Matrix<K>& mapLine(size\_t lineIndex, AbstractFunctor<K>& f) {  
 for (auto& elem : matrix[lineIndex]) {  
 elem = f.invoke(elem);  
 }  
 return \*this;  
}

Метод ltrx::Matrix::mapLine(…) производит некоторое действие над каждым элементов матрицы в определённой строке, изменяя сам объект.

functor(IncFunctor, T, T, {  
 elem++;  
 return elem;  
}) s\_incFunctor;  
  
functor(DecFunctor, T, T, {  
 elem--;  
 return elem;  
}) s\_decFunctor;

Классы ltrx::Matrix::IncFunctor и ltrx::Matrix:DecFunctor реализуют класс ltrx::Matrix::AbstractFunctor, увеличивают каждый элемент матрицы при помощи метода T::operator++(int), где T — тип элемента матрицы.

4. *Реализация математических операторов для двух матриц*

Matrix& add(const Matrix& other) {  
 checkSizeEquals(other);  
 for (size\_t i = 0; i < width; i++) {  
 for (size\_t j = 0; j < height; j++) {  
 this->set(i, j, this->matrix[i][j] + other.matrix[i][j]);  
 }  
 }  
 return \*this;  
}

Каждый математический оператор реализован подобным образом: существует метод (к примеру, add(const Matrix& other)), который к каждому элементу матрицы прибавляет соответствующий элемент другой матрицы.

5. *Реализация математических операторов для матрицы и числа*

Matrix& add(const T& value) {  
 for (size\_t i = 0; i < width; i++) {  
 for (size\_t j = 0; j < height; j++) {  
 this->set(i, j, this->matrix[i][j] + value);  
 }  
 }  
 return \*this;  
}

Каждый математический оператор реализован подобным образом: существует метод (к примеру add(const T& value), где T — тип элементов матрицы), который к каждому элементу матрицы прибавляет данное число.

6. Реализация методов ltrx::Matrix::deteminator() и ltrx::Matrix::norm()

Matrix minor(int line, int row) {  
 size\_t l = line;  
 size\_t r = row;  
 return minor(l, r);  
}  
  
T determinator() {  
 if (height != width || height < 1) {  
 throw new MatrixError("Determinator is not defined for this matrix.");  
 }  
  
 if (height == 2) {  
 return get(1, 1) \* get(0, 0) - get(1, 0) \* get(0, 1);  
 }  
  
 T result = 0;  
 size\_t zero = 0;  
 for (size\_t i = 0; i < width; i++) {  
 Matrix aMinor = minor(i, zero);  
 result += (getSign(i, 0) ? 1 : -1) \* get(i, 0) \* aMinor.determinator();  
 }  
  
 return result;  
}  
  
T norm() {  
 T result = 0;  
 for (int i = 0; i < height; i++) {  
 T lineSum = 0;  
 for (int j = 0; j < width; j++) {  
 lineSum += get(j, i);  
 }  
 if (result < lineSum) {  
 result = lineSum;  
 }  
 }  
 return result;  
}

7. Документация

Файл конфигурации doxygen был сгенерирован при помощи командой утилиты (doxygen -g). Сложностей с описанием классов и методов не возникло, примеры:

/\*\*  
 \* @brief Get an element of a matrix  
 \*   
 \* @param horizontalIndex location of an element horizontally  
 \* @param verticalIndex location of an element vertically  
 \* @return T element  
 \*   
 \* @author R. Lipski  
 \*/

/\*\*  
 \* @brief Get the height of a matrix  
 \*   
 \* @return size\_t height of a matrix  
 \*   
 \* @author R. Lipski  
 \*/

/\*\*  
 \* @brief map each element of a certain line of a matrix to a functor  
 \*   
 \* @tparam K type of new elements of a matrix  
 \* @param lineIndex number of line  
 \* @param f functor  
 \* @return Matrix<K>& reference to this object  
 \*   
 \* @author R. Lipski  
 \*   
 \* @warning trying to use a functor with return type different from current will lead to undefined behaviour.  
 \*/

8. Тесты

Использовалась библиотека GoogleTest, добавленная в репозиторий в качестве подмодуля. Сложностей с написанием тестов не возникло благодаря подробной документации. Примеры тестов:

TEST(MatrixEquals, MustReturnTrueWhenMatrixesAreEqual) {  
 auto m1 = ltrx::Matrix<double>(1, 4, 4);  
 auto m2 = ltrx::Matrix<double>(1, 4, 4);  
  
 ASSERT\_EQ(m1, m2);  
}  
  
TEST(MatrixPostIncrement, MustIncreaseEveryElementOfMatrixByOne) {  
 auto m1 = ltrx::Matrix<double>(0, 5, 5);  
 m1++;  
  
 for (int i = 0; i < m1.getWidth(); i++) {  
 for (int j = 0; j < m1.getHeight(); j++) {  
 ASSERT\_EQ(1, m1.get(i, j));  
 }  
 }  
}  
  
TEST(MatrixPreIncrement, MustIncreaseEveryElementOfMatrixByOne) {  
 auto m1 = ltrx::Matrix<double>(0, 5, 5);  
 ++m1;  
  
 for (int i = 0; i < m1.getWidth(); i++) {  
 for (int j = 0; j < m1.getHeight(); j++) {  
 ASSERT\_EQ(1, m1.get(i, j));  
 }  
 }  
}

Список использованных источников:

1. CppReference - <http://cppreference.com/>

2. GoogleTest User’s Guide - <https://google.github.io/googletest/>

3. Документируем код эффективно при помощи Doxygen **-** <https://habr.com/ru/post/252101/>