ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО"

Отчёт по лабораторной работе №2 по учебной дисциплине «Алгоритмы и структуры данных»

Выполнил:

Студент Щербаков Н.А.

группы 3824Б1ФИ2

Нижний Новгород – 2025 г.

1.Постановка задачи

Цель данной работы — создания программных средств, поддерживающих эффективное хранение матриц специального вида (верхнетреугольных) и выполнение основных операций над ними: сложение, вычитание, копирование, сравнение. То есть написание реализации классов TVector и TMatrix, а также автоматических тестов Google Test.

2.Описание программной реализации

2.1.Класс TVector

Класс TVector представляет шаблонный вектор произвольного типа данных, предназначенный для хранения и обработки последовательности элементов с возможностью задания смещения индекса первого элемента вектора. Класс обеспечивает выполнение основных арифметических векторных операций и операции со скаляром (сложение векторов, вычитание векторов, прибавление скаляра, вычитание скаляра, умножение на скаляр, скалярное произведение), а также операции сравнение, присваивание и ввод-вывод данных.

Поля класса:

- **int** Size – размер вектора (количество значимых элементов вектора).
- **ValType** *pVector - память для представления вектора.
- **int** StartIndex – индекс первого элемента вектора, то есть индекс первого значимого элемента вектора.

Методы класса:

1) TVector(**int** s = 10, **int** si = 0)

Функционал: Конструктор. Создаёт вектор размера s, начиная с индекса si. Инициализирует вектор, устанавливая все элементы вектора по умолчанию. При выходе за пределы допустимых значений s и si: s = 10 и si = 0.

Параметры: s – размер значимых элементов вектора, si – индекс первого значимого элемента вектора.

Сложность: O(s) - где s - количество элементов в векторе (под них выделяется память).

2) TVector(**const TVector&** v)

Функционал: Конструктор копирования. Создаёт копию существующего вектора v.

Параметры: v - копируемый объект класса TVector.

Сложность: O(v.Size) - так как копируем каждый элемент массива v.pVector в массив pVector в цикле от 0 до v.Size (также выделяется память для элементов текущего вектора - O(v.Size)).

3) ~TVector()

Функционал: Деструктор. Освобождает память, выделенную под элементы вектора.

4) **int** GetSize()

Функционал: Возвращает размер вектора (число значимых элементов).

Возвращаемое значение: int, значение поля Size.

5) **int** GetStartIndex()

Функционал: Возвращает индекс первого элемента вектора (индекс первого значимого элемента вектора).

Возвращаемое значение: int, значение поля StartIndex.

6) **ValType&** operator[](**int** pos)

Функционал: Возвращает значение соответствующего элемента вектора по индексу в диапазоне [StartIndex; StartIndex + Size – 1].

Параметры: pos - индекс элемента вектора.

Возвращаемое значение: ValType&, ссылка на элемент вектора по индексу pos.

7) **bool** operator==(**const TVector&** v) **const**

Функционал: Сравнивает два вектора на равенство. Есть проверка на несовпадение Size и/или StartIndex.

Параметры: v - вектор для сравнения с текущим вектором.

Возвращаемое значение: bool, true - вектора равны, false - вектора не равны или не совпадают Size и/или StartIndex у векторов.

Сложность: O(Size) - так как сравниваем вектора поэлементно в цикле от 0 до Size.

8) **bool** operator!=(**const TVector&** v) **const**

Функционал: Сравнивает два вектора на неравенство. Используем перегрузку оператора ==, но инвертируем результат.

Параметры: v - вектор для сравнения с текущим вектором.

Возвращаемое значение: bool, false - вектора равны, true - вектора не равны или не совпадают Size и/или StartIndex у векторов.

Сложность: O(Size) - так как сравниваем вектора поэлементно в цикле от 0 до Size.

9) **TVector&** operator=(**const TVector&** v)

Функционал: Оператор присваивания. Копирует данные из одного вектора в другой. Есть проверка на самоприсваивание.

Параметры: v - источник данных для копирования.

Возвращаемое значение: TVector&, ссылка на текущий вектор.

Сложность: O(Size) - так как вызываем конструктор копирования для векторов класса TVector. (Все три std::swap имеют сложность O(1) в данной перегрузке).

10) **TVector** operator+(**const ValType&** val)

Функционал: Прибавление скаляра к вектору.

Параметры: val, скаляр.

Возвращаемое значение: TVector, возвращает новый вектор - результат сложения текущего вектора со скаляром.

Сложность: O(Size) - так как прибавляем скаляр к каждому элементу вектора в цикле от 0 до Size. Также вызывается конструктор копирования текущего вектора со сложностью O(Size).

11) **TVector** operator-(**const ValType&** val)

Функционал: Вычитание скаляра из вектора.

Параметры: val, скаляр.

Возвращаемое значение: TVector, возвращает новый вектор - результат вычитания из текущего вектора скаляра.

Сложность: O(Size) - так как вычитаем скаляр из каждого элемента вектора в цикле от 0 до Size. Также вызывается конструктор копирования текущего вектора со сложностью O(Size).

12) **TVector** operator*(**const ValType&** val)

Функционал: Умножение вектора на скаляр.

Параметры: val, скаляр.

Возвращаемое значение: TVector, возвращает новый вектор - результат умножения текущего вектора на скаляра.

Сложность: O(Size) - так как умножаем на скаляр каждый элемент вектора в цикле от 0 до Size. Также вызывается конструктор копирования текущего вектора со сложностью O(Size).

13) **TVector** operator+(**const TVector&** v)

Функционал: Сложение двух векторов. Есть проверка на несовпадение Size и/или StartIndex.

Параметры: v - второй вектор для сложения.

Возвращаемое значение: TVector, возвращает новый вектор - результат сложения двух векторов.

Сложность: O(Size) - так как складываем элементы двух векторов в цикле от 0 до Size. Также вызывается конструктор копирования текущего вектора со сложностью O(Size).

14) **TVector** operator-(**const TVector&** v)

Функционал: Вычитание двух векторов. Есть проверка на несовпадение Size и/или StartIndex.

Параметры: v - второй вектор для вычитания.

Возвращаемое значение: TVector, возвращает новый вектор - результат вычитания двух векторов.

Сложность: O(Size) - так как вычитаем элементы двух векторов в цикле от 0 до Size. Также вызывается конструктор копирования текущего вектора со сложностью O(Size).

15) **ValType** operator*(**const TVector&** v)

Функционал: Скалярное произведение двух векторов. Есть проверка на несовпадение Size и/или StartIndex.

Параметры: v - второй вектор для скалярного произведения.

Возвращаемое значение: ValType, возвращает результат скалярного произведения того же типа, что и элементы векторов.

Сложность: O(Size) - так как попарно перемножаем элементы векторов и прибавляем к Res в цикле от 0 до Size.

16) friend **std::istream&** operator>>(**std::istream&** in, **TVector&** v)

Функционал: Оператор ввода. Считывает вектор из потока поэлементно (до разделителя). Есть проверка на некорректный ввод различного характера. Формат ввода: elem1_elem2_elem3_... где _ - некоторое число разделителей.

Параметры: in - входной поток, v - объект для записи вектора из потока.

Возвращаемое значение: std::istream&, ссылка на входной поток.

Сложность: O(v.Size * 2) ~ O(v.Size) - так как в первом цикле от 0 до v.Size очищаем вектор v от прошлых значений элементов, а во втором цикле от 0 до v.Size считываем элементы из потока.

17) friend **std::ostream&** operator<<(**std::ostream&** out, **const TVector&** v)

Функционал: Оператор вывода. Выводит вектор в поток поэлементно, проставляя разделители.

Параметры: out - выходной поток, v - выводимый вектор.

Возвращаемое значение: std::ostream&, ссылка на выходной поток.

Сложность: O(v.StartIndex + v.Size) - так как первый цикл от 0 до v.StartIndex выводит незначимые элементы вектора (то есть те, что по индексу меньше StartIndex) - 0; а второй цикл от 0 до v.Size выводит значимые элементы вектора.

2.2.Класс TMatrix

Класс TMatrix реализует шаблонную матрицу на основе вектора векторов, представленного классом TVector. Каждый элемент матрицы является вектором, что обеспечивает удобное хранение и обработку данных. Класс поддерживает основные операции над матрицами — сложение, вычитание, сравнение и присваивание, а также ввод и вывод в поток. Реализация позволяет создавать матрицы произвольного типа данных и эффективно выполнять вычисления. Благодаря верхнетреугольной структуре обеспечивается экономия памяти и вычислительных ресурсов, поскольку хранятся только элементы на и выше главной диагонали.

Поля класса:

Собственных полей у данного класса нет, но он наследуется от TVector<TVector<ValType>>. Поэтому все необходимые данные хранятся в полях класса TVector. Эти поля можно интерпретировать следующим образом:

- **int** Size – размер квадратной верхнетреугольной матрицы (число строк и столбцов матрицы). У TMatrix Size также является размером вектора векторов, то есть количеством строк в матрице. Для строки (внутреннего вектора) с индексом i размер Size равен s - i, где s = Size.

- **int** StartIndex – индекс первого элемента вектора векторов. У TMatrix StartIndex всегда равно 0. Для строки (внутреннего вектора) с индексом i StartIndex равен i.
- **TVector<ValType>** *pVector - память для представления матрицы. Это массив из векторов (вектор векторов). Каждый элемент этого массива является объектом класса TVector<ValType>.

Методы класса:

1) TMatrix(**int** s = 10)

Функционал: Конструктор. Инициализирует верхнетреугольную матрицу размера s. Вызывается конструктор класса TVector для каждой строки матрицы (внутреннего вектора). При выход за пределы допустимых значений s: s = 10.

Параметры: s - размер верхнетреугольной матрицы.

Сложность: $O(s^2)$ - так как в цикле от 0 до s вызываем конструктор класса TVector, который имеет сложность O(s). Также создаём вектор векторов (TVector<TVector<ValType>>(s, 0)) – $O(s^2)$.

2) TMatrix(**const TMatrix&** mt)

Функционал: Конструктор копирования. Создаёт новую верхнетреугольную матрицу - копию верхнетреугольной матрицы mt.

Параметры: mt – исходная верхнетреугольная матрица (откуда идёт копирование).

Сложность: $O((mt.Size)^2)$ - так как в цикле от 0 до mt.Size применяем перегруженный оператор присваивания для копирования векторов (строк матрицы). Оператор присваивания имеет сложность O(mt.Size).

3) TMatrix(**const TVector<TVector<ValType>>&** mt)

Функционал: Конструктор преобразования типа. Создаём не константный объект mt_temp типа TVector<TVector<ValType>> - копию константного объекта mt, чтобы применять не константную перегрузку доступа к элементам вектора [] класса TVector к объекту mt_temp. Есть проверки на соответствие верхнетреугольному виду матрицы.

Параметры: mt - переданный вектор векторов, который имеет вид верхнетреугольной матрицы.

Сложность: $O(mtSize^2)$ – так как в цикле от 0 до mtSize применяем перегруженный оператор присваивания для копирования векторов (элементов mt). Оператор присваивания имеет сложность O(mtSize). mtSize = mt_temp.GetSize().

4) **bool** operator==(**const TMatrix&** mt) **const**

Функционал: Сравнивает две верхнетреугольные матрицы на равенство. Есть проверка на несовпадение размеров матриц.

Параметры: mt - матрицы для сравнения с текущей матрицей.

Возвращаемое значение: bool, true - матрицы равны, false - матрицы не равны или не совпадают размеры матриц.

Сложность: $O(Size^2)$ – так как в цикле от 0 до Size применяем перегруженный оператор сравнения != для векторов (строк матрицы). Оператор сравнения != имеет сложность O(Size).

5) **bool** operator!=(**const TMatrix&** mt) **const**

Функционал: Сравнивает две верхнетреугольные матрицы на неравенство. Используем перегрузку оператора сравнения == для матриц, но инвертируем результат.

Параметры: mt - матрицы для сравнения с текущей матрицей.

Возвращаемое значение: bool, false - матрицы равны, true - матрицы не равны или не совпадают размеры матриц.

Сложность: $O(Size^2)$ - сложность перегруженного оператора сравнения == для матриц.

6) **TMatrix&** operator= (**const TMatrix&** mt)

Функционал: Оператор присваивания. Копирует данные из одной верхнетреугольной матрицы в другую. Есть проверка на самоприсваивание.

Параметры: mt - источник данных для копирования.

Возвращаемое значение: TMatrix&, ссылка на текущую верхнетреугольную матрицу.

Сложность: $O(Size^2)$ - так как вызываем конструктор копирования для матриц класса TMatrix. (Все три std::swap имеют сложность O(1) в данной перегрузке).

7) **TMatrix** operator+ (**const TMatrix&** mt)

Функционал: Сложение двух верхнетреугольных матриц. Есть проверка на несовпадение размеров матриц.

Параметры: mt - вторая матрица для сложения.

Возвращаемое значение: TMatrix, возвращает новую верхнетреугольную матрицу - результат сложения двух матриц.

Сложность: $O(Size^2)$ - так как в цикле от 0 до Size применяем перегруженный оператор сложения двух векторов класса TVector, который имеет сложность O(Size).

8) **TMatrix** operator- (**const TMatrix&** mt)

Функционал: Вычитание двух верхнетреугольных матриц. Есть проверка на несовпадение размеров матриц.

Параметры: mt - вторая матрица для вычитания.

Возвращаемое значение: TMatrix, возвращает новую верхнетреугольную матрицу - результат вычитания двух матриц.

Сложность: $O(Size^2)$ - так как в цикле от 0 до Size применяем перегруженный оператор вычитания двух векторов класса TVector, который имеет сложность O(Size).

9) friend **std::istream&** operator>>(**std::istream&** in, **TMatrix&** mt)

Функционал: Оператор ввода. Считывает верхнетреугольную матрицу из потока поэлементно (до разделителя), применяя перегруженный оператор ввода вектора класса TVector. Есть проверка на некорректный ввод различного характера (от перегруженного ввода вектора класса TVector). Формат ввода: elem1_elem2_elem3_... где _ - некоторое число разделителей.

Параметры: in - входной поток, mt - объект для записи верхнетреугольной матрицы из потока.

Возвращаемое значение: std::istream&, ссылка на входной поток.

Сложность: $O((mt.Size)^2 * 2) \sim O((mt.Size)^2)$ - так как в первом цикле от 0 до mt.Size очищаем матрицу mt от прошлых значений элементов, а во втором цикле от 0 до mt.Size считываем элементы из потока. При этом в каждой итерации циклов применяется перегруженный оператор ввода вектора класса TVector, который имеет сложность O(v.Size).

10) friend **std::ostream&** operator<<(**std::ostream&** out, **const TMatrix&** mt)

Функционал: Оператор вывода. Выводит верхнетреугольную матрицу в поток с помощью перегруженного оператора вывода вектора класса TVector.

Параметры: out - выходной поток, mt – выводимая верхнетреугольная матрица.

Возвращаемое значение: std::ostream&, ссылка на выходной поток.

Сложность: $O((mt.Size)^2)$ – так как в цикле от 0 до mt.Size применяем перегруженный оператор вывода вектора класса TVector, который имеет сложность O(v.StartIndex + v.Size) ~ O(mt.Size).

3.Краткие комментарии к тестам

Для проверки правильности работы классов TVector и TMatrix были созданы тесты с использованием фреймворка Google Test, охватывающие основные операции, граничные ситуации и обработку ошибок.

3.1.Тесты класса TVector

- **CreateVectorWithinAcceptableValues** – Проверяет, что конструкторы TVector (по умолчанию, с параметрами и копирования) корректно создают объекты, с корректным размеров вектора и индексом первого элемента вектора. В тесте создаются векторы с разными размерами и стартовыми индексами, включая граничные случаи размера вектора (1 и MAX_VECTOR_SIZE). Также проверяется возможность создания вектора, у которого размер меньше стартового индекса, и копирование уже существующего вектора.
- **CreateVectorBeyondAcceptableValues** – Проверяет работу конструкторов класса TVector при выходе за пределы допустимых значений размера вектора и/или индекса первого элемента вектора:
  - отрицательным размером;
  - нулевым размером;
  - отрицательным стартовым индексом;
  - размером, превышающим MAX_VECTOR_SIZE.

  При всех таких случаях размер вектора и индекс первого элемента вектора устанавливаются по умолчанию: s = 10, si = 0. Таким образом, тест подтверждает корректность валидации аргументов конструктора.

- **GetSizeAndStartIndex** – Проверяет корректность работы методов GetSize() и GetStartIndex(). После создания вектора с заданным размером и стартовым индексом убеждается, что методы возвращают те же значения, что были переданы конструктору.

- **AccessToTheElementsNoThrow** – Проверяет корректность доступа к элементам вектора через перегруженный оператор []. Убеждается, что чтение и запись элементов в диапазоне [StartIndex; StartIndex + Size - 1] происходят без ошибок. Также проверяется возможность инициализации всех элементов значением.
- **AccessToTheElementsAnyThrow** – Проверяет, что попытка обратиться к элементам за пределами допустимого диапазона (pos < StartIndex или pos > StartIndex + Size - 1) вызывает исключение std::out_of_range. Таким образом тест подтверждает безопасность доступа по индексу.
- **CompareVector** – Проверяет корректность работы операторов сравнения == и !=. Тестирует несколько случаев:
  - векторы разного размера или разного стартового индекса считаются неравными;
  - векторы одинаковой структуры (одинаковый размер и стартовый индекс), но с разными элементами – не равны;
  - векторы одинаковой структуры (одинаковый размер и стартовый индекс), с одинаковыми элементами – равны.
- **AssignmentVector** – Проверяет корректность работы оператора присваивания =. Сначала выполняется присваивание между векторами разных размеров и стартовых индексов. После присваивания проверяется, что векторы стали равны. Также тестирует, что копирование элементов происходит корректно при выполнении оператора присваивания.
- **VectorOperationsWithScalar** – Проверяет работу арифметических операций вектора со скаляром (+, -, *):
  - прибавление скаляра ко всем элементам;
  - вычитание скаляра из всех элементов;
  - умножение всех элементов на скаляр.

  Сравнивает результаты операций с вручную подготовленными эталонными векторами.
- **VectorOperationsWithVectorNoThrow** – Проверяет корректность выполнения операций между двумя векторами одинакового размера и стартового индекса:
  - сложение двух векторов;
  - вычитание двух векторов;
  - скалярное произведение двух векторов.

  Сравнивает каждый элемент результата с ожидаемым значением. Также проверяются граничные случаи – операции вектора с самим собой и скалярное произведение нулевых векторов.
- **VectorOperationsWithVectorAnyThrow** – Проверяет, что при попытке выполнить арифметические операции (+, -, *) между векторами, имеющими разные размеры или разные стартовые индексы, выбрасывается исключение std::invalid_argument. Это гарантирует корректную проверку совместимости векторов перед операциями.

3.2.Тесты класса TMatrix

- **CreateMatrixWithinAcceptableValues** – Проверяет корректное создание объектов класса TMatrix разными конструкторами:
  - конструктор по умолчанию;

- конструктор с параметром;
- конструктор копирования.

Также тестирует граничные случаи размера матрицы (1 и MAX_MATRIX_SIZE). Проверяется, что при копировании матрицы с помощью конструктора копирования сохраняются значения элементов.

- **CreateMatrixBeyondAcceptableValues** – Проверяет, что при создании матрицы с некорректным размером (отрицательным, нулевым, больше MAX_MATRIX_SIZE) размер матрицы устанавливается по умолчанию: s = 10.
- **VectorOfVectorsConversionToMatrixNoThrow** – Проверяет корректность конструктора преобразования типа - TMatrix(const TVector<TVector<ValType>>& mt). Создаёт вектор векторов, структура которого соответствует верхнетреугольной матрице (строка i имеет размер s - i и стартовый индекс i). Убеждается, что из такого вектора векторов матрица создаётся без выброса исключений.
- **VectorOfVectorsConversionToMatrixAnyThrow** – Проверяет, что при нарушении структуры верхнетреугольной матрицы (неправильный StartIndex вектора векторов или неверные размеры строк (элементов вектора векторов)) при попытке преобразования вектора векторов к верхнетреугольной матрице выбрасывается исключение std::invalid_argument.
- **CompareMatrix** – Проверяет корректность операторов сравнения == и != для верхнетреугольных матриц:
  - матрицы разного размера считаются неравными;
  - матрицы одинаковые по размеру и значениям элементов считаются равными;
  - матрицы, у которых присутствует различие хотя бы в одном элементе, считаются неравными.
- **AssignmentMatrix** – Проверяет работу оператора присваивания = для верхнетреугольных матриц. В тесте выполняется присваивание между матрицами разных размеров, а затем проверяется, что структура и значения совпадают с помощью оператора сравнения матриц. Также проверяется корректная работа копирования элементов матрицы при выполнении оператора присваивания.
- **MatrixOperationsAdditionAndSubtractionNoThrow** – Проверяет корректность операций сложения и вычитания верхнетреугольных матриц одинакового размера. Матрицы предварительно инициализируются значениями, затем сравниваются результаты операций + и - матриц с ожидаемыми значениями элементов матрицы. Также проверяются операции сложения и вычитания матрицы с самой собой.
- **MatrixOperationsAdditionAndSubtractionAnyThrow** – Проверяет, что при попытке сложить или вычесть матрицы разных размеров выбрасывается исключение std::invalid_argument. Подтверждает корректную проверку совместимости матриц перед операциями.