

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»**

Институт № 7
«Робототехнические и интеллектуальные системы»
Кафедра 703
«Системное проектирование авиакомплексов»

ОТЧЕТ
о курсовой работе по курсу
«Программирование на языках высокого уровня»

Разработал:
Студент группы М70-406С-22
Лапшин А. А.

Принял:
Старший преподаватель
кафедры 703
Барчев Н. Б.

Москва, 2025

Содержание

1. Задание	5
1.1. Назначение проекта	5
1.2. Основные требования	5
2. Псевдокод	6
3. Сведения о программной реализации	11
3.1. Язык программирования и среда разработки	11
3.2. Описание входных и выходных данных	11
3.3. Программный интерфейс	11
3.3.1. Файл <code>operations.cpp</code>	11
3.3.2. Файл <code>graph.cpp</code>	14
3.3.3. Файл <code>parser.cpp</code>	20
3.3.3.1. Пространство имен <code>config</code>	21
3.3.4. Пространство имен <code>table</code>	23
3.3.5. Файл <code>utils.cpp</code>	28
3.3.5.1. Пространство имен <code>logger</code>	29
3.3.5.2. Пространство имен <code>tui</code>	30
3.3.5.3. Глобальные функции	34
3.3.6. Файл <code>xrouter.cpp</code>	36
3.3.6.1. Пространство имен <code>SimpleDAG</code>	36
3.3.6.2. Пространство имен <code>SimpleDAG::Internal</code>	37
4. Инструкция пользователя	43
4.1. Обзор системы <code>SimpleDAG</code>	43
4.2. Подготовка к работе	43
4.2.1. Структура директорий	43
4.2.2. Подготовка конфигурационного файла	43
4.2.3. Подготовка CSV-файла	44
4.3. Рабочий процесс	44
4.3.1. Шаг 1: Выбор конфигурационного файла	44
4.3.2. Шаг 2: Проверка конфигурации	45
4.3.3. Шаг 3: Просмотр доступных операций	45
4.3.4. Шаг 4: Ввод схемы графа	45
4.3.5. Шаг 5: Валидация схемы	46
4.3.6. Шаг 6: Выполнение операций	47
4.3.7. Шаг 7: Просмотр результатов	47
4.4. Работа с ошибками	47

4.4.1. Ошибки загрузки конфигурации	47
4.4.2. Ошибки ввода схемы	48
4.4.3. Ошибки выполнения операций	48
4.5. Пример полного сеанса работы	48
4.6. Клавиши управления	49
4.7. Ограничения системы	49
5. Листинг	50
5.1. Заголовочные файлы	50
5.1.1. graph.h	50
5.1.2. operations.h	52
5.1.3. parser.h	54
5.1.4. router.h	57
5.1.5. utils.h	59
5.1.6. graphics.h	62
5.2. Исходные файлы	64
5.2.1. main.cpp	64
5.2.2. graph.cpp	64
5.2.3. operations.cpp	73
5.2.4. parser.cpp	75
5.2.5. router.cpp	84
5.2.6. utils.cpp	89
5.3. Конфигурационные файлы	95
5.3.1. Makefile (основной)	95
5.4. Тесты	96
5.4.1. tests/graph/test_graph.cpp	96
5.4.2. tests/operations/test_operations.cpp . .	104
5.4.3. tests/parser/test_parser.cpp	104
5.4.4. tests/utils/test_utils.cpp	110
5.5. Скрипты	116
5.5.1. buildTests.sh	116
5.5.2. runTests.sh	117
5.5.3. .github/workflows/ci.yml	118
6. Тестирование	120
6.1. Среда тестирования	120
6.2. Ручное тестирование	120
6.2.1. Проверка директории data на пустоту . .	120
6.2.2. Выбор файла конфигурации	121

6.2.2.1. Некорректный ввод (буквы, спецсимволы)	121
6.2.2.2. Введенное число вне диапазона	122
6.2.2.3. Пустой ввод	123
6.2.2.4. Ввод «0» (выход)	123
6.2.3. Ввод схемы графа	124
6.2.3.1. Недопустимые символы (буквы, спецсимволы)	124
6.2.3.2. Разделитель в начале/конце строки .	125
6.2.3.3. ID операции вне диапазона	126
6.2.3.4. Пустая строка	126
6.2.3.5. Корректный ввод схемы	127
6.2.4. Проверка конфигурации	127
6.2.4.1. Недоступная операция в YAML	127
6.2.4.2. Номер столбца вне диапазона	129
6.2.5. Создание и выполнение графа	130
6.2.5.1. Граф без ветвлений	130
6.2.5.2. Граф с ветвлениями	131
6.3. Автоматическое тестирование	133
6.3.1. Среда разработки	133
6.3.2. GitHub Actions (CI)	136

1. Задание

1.1. Назначение проекта

Разработать программную систему – упрощенный аналог Apache Airflow для автоматической обработки табличных данных в формате CSV. Система должна обеспечивать построение направленного ациклического графа (DAG) операций и их последовательное выполнение над данными.

1.2. Основные требования

- Конфигурация через YAML
- Обработка табличных данных
- Графовое представление операций
- Логирование результатов
- C++ (стандарт C++17 или выше)
- Минимизация дублирования кода
- Соблюдение принципов SOLID

2. Псевдокод

```
1 Запустить программу SimpleDAG
2 | Показать логотип приложения
3
4 ШАГ 1: Загрузка конфигурации
5 | Получить список YAML-файлов в директории data/
6 | ЕСЛИ файлов не найдено:
7 |   | Вывести ошибку: «Не найдены конфигурационные
8 |   |   | файлы»
9 |   | Завершить программу с кодом ошибки 1
10 | ИНАЧЕ:
11 |   | Отобразить меню выбора файла с нумерацией
12 |   | ПОКА выбор не валиден:
13 |   |   | Запросить ввод номера файла от пользователя
14 |   |   | ЕСЛИ ввод == 0:
15 |   |   |   | Завершить программу (пользователь выбрал
16 |   |   |   |   | выход)
17 |   |   | ИНАЧЕ ЕСЛИ номер вне диапазона:
18 |   |   |   | Вывести сообщение об ошибке
19 |   |   |   | Повторить запрос ввода
20 |   | ИНАЧЕ:
21 |   | ШАГ 2: Проверка конфигурации
22 |   | Проверить наличие указанных функций
23 |   | ЕСЛИ найдены неизвестные функции:
24 |   |   | Вывести список неизвестных функций
25 |   |   | Вывести предупреждение
26 |   |   | Вывести список поддерживаемых
27 |   |   | Завершить программу с кодом ошибки 1
28
29 | ИНАЧЕ:
```

```
30 | Получить список ID операций
31 | Получить путь к CSV-файлу
32 | ЕСЛИ путь пустой:
33 |   | Вывести сообщение об ошибке
34 |   | Завершить программу с кодом ошибки 1
35 |
36 ШАГ 3: Загрузка данных
37 | Сформировать полный путь к CSV: data/ + имя_файла
38 | Прочитать CSV-файл
39 | ЕСЛИ файл не удалось открыть:
40 |   | Вывести ошибку: «Не удалось открыть файл
41 |   | данных»
42 |   | Завершить программу с кодом ошибки 1
43 | ИНАЧЕ:
44 |   | Отобразить список доступных операций с номерами
45 |   | ДЛЯ каждой операции в конфигурации:
46 |     | Вывести номер: [N] ID_операции
47 |
48 ШАГ 4: Инициализация логирования
49 | Получить текущее время
50 | Создать имя лог-файла с временной меткой
51 | Открыть лог-файл для записи
52 |
53 ШАГ 5: Ввод и обработка схемы графа
54 | Вывести инструкции по формату ввода схемы
55 | ПОВТОРЯТЬ до получения валидной схемы:
56 |   | Вывести приглашение «Ввод»
57 |   | ПОКА не введена строка «end»:
58 |     | Считать строку из стандартного ввода
59 |     | Добавить строку в вектор схемы
60 |
```

61 | Проверить схему
62 | **ЕСЛИ** найдены ошибки:
63 | | Вывести сообщение об ошибке
64 | | **ДЛЯ** каждой строки с ошибками:
65 | | | Вывести номер строки и текст ошибки
66 | | | Очистить вектор схемы
67 | | | Вывести «▲ Пожалуйста, исправьте ошибки и
введите схему заново:»
68 |
69 | **ИНАЧЕ**:
70 | | Вывести «Схема графа корректна!»
71 | | Создать граф через
72 | | Прервать цикл проверки
73 |
74 | ШАГ 6: Выполнение операций
75 | | Выполнить обход графа в глубину через
76 | | **ДЛЯ** каждого узла графа в порядке обхода:
77 | | | Получить ID операции из узла
78 | | | Получить тип операции
79 | | | Получить номер столбца
80 | | | Определить тип данных столбца
81 |
82 | | **ЕСЛИ** тип столбца == NUMERIC:
83 | | | Считать числовые данные
84 | | | Вызвать операцию
85 | | | Записать результат в лог
86 |
87 | | **ИНАЧЕ ЕСЛИ** тип столбца == STRING:
88 | | | Считать строковые данные
89 | | | Вызвать операцию
90 | | | Записать результат в лог
91 |

```
92 |     | ИНАЧЕ ЕСЛИ тип столбца == UNKNOWN:  
93 |     |     Записать предупреждение в лог  
94 |     |     Пропустить эту операцию
```

95

96 ШАГ 7: Завершение работы

97 | Освободить ресурсы конфигурации

98 Очистить табличные данные

99 | Закрыть лог-файл

100 Очистить граф:

101 | Завершить программу с кодом 0 (успех)

102

103 Алгоритм проверки схемы:

104 | Вход: вектор строк схемы, вектор ID операций

105 для каждой строки в схеме:

106 | Инициализировать вектор ошибок для строки

ЕСЛИ строка пустая: добавить ошибку «пустая строка»

ЕСЛИ разделитель «-» в начале или конце:
добавить «неожиданный разделитель»

109 **ЕСЛИ** есть символы кроме цифр и «-»: добавить «неожиданные символы»

110 Разделить строку на индексы по «-»

111 **для** каждого индекса:

Попытаться преобразовать в число

113 **ЕСЛИ** преобразование удалось **И** индекс вне
диапазона [1, размер ID]:

114 | | | Добавить ошибку «индекс вне диапазона»

115 | Выход: вектор векторов ошибок для каждой строки

116

117 Алгоритм создания графа:

118 | Вход: вектор строк схемы, вектор ID операций

119 | **ДЛЯ** каждой строки в схеме:
120 | Разделить строку на индексы по «-»
121 | **ДЛЯ** каждого индекса *i*:
122 | Получить ID операции
123 | **ЕСЛИ** узел с таким ID еще не создан:
124 | Создать новый узел
125 | Добавить узел в граф
126 | **ЕСЛИ** *i* > 0:
127 | Создать связь между предыдущим и текущим
128 | узлом
128 | Выход: построенный граф в глобальных переменных
129

130 | **Алгоритм определения типа столбца:**
131 | Вход: номер столбца
132 | **ДЛЯ** каждой строки в таблице:
133 | **ЕСЛИ** столбец существует в строке:
134 | **ЕСЛИ** значение является числом: поднять флаг
134 | «есть числа»
135 | **ИНАЧЕ**: «есть строки»
136 | **ЕСЛИ** «есть числа» - правда И «есть строки» -
136 | ложь: вернуть «ЧИСЛО»
137 | **ЕСЛИ** поднят флаг «есть строки» вернуть «СТРОКА»
138 | **ИНАЧЕ**: вернуть «НЕИЗВЕСТНО»

3. Сведения о программной реализации

3.1. Язык программирования и среда разработки

Программа написана на языке программирования C++ (стандарт ISO/IEC 14882:2017) в среде разработки с открытым исходным кодом LazyVim (готовая конфигурация NeoVim).

3.2. Описание входных и выходных данных

Входные данные:

- конфигурационные файлы в формате .yaml;
- таблицы в формате .csv;
- ввод с клавиатуры.

Выходные данные:

- логирование результата выполнения операций в текстовый файл;
- вывод в консоль.

3.3. Программный интерфейс

3.3.1. Файл operations.cpp

Глобальные переменные:

- `map<string, FunctionVariant> operation_map` – словарь операций, сопоставляющий имена операций с их реализациями

Типы данных:

```
1 // Тип variant для хранения различных типов функций
2 using FunctionVariant = variant<
3     function<float(vector<float>)>,
4     function<string(vector<string>)>
5 >;
```

cpp

Поддерживаемые операции в словаре:

1. **sum** – вычисление суммы числовых значений
 2. **average** – вычисление среднего арифметического числовых значений
 3. **concatenate** – конкатенация строковых значений
-

```
1 float callOperation(const string &op_name,  
2                           vector<float> arg);
```

cpp

Описание

Вызывает числовую операцию из словаря операций.

Параметры:

- `op_name` – имя вызываемой операции
- `arg` – вектор числовых аргументов для операции

Возвращаемое значение:

Результат выполнения операции в виде числа с плавающей точкой.

Исключения:

- `runtime_error` если операция не найдена в словаре
 - `runtime_error` если тип операции не соответствует числовому типу
-

```
1 string callOperation(const string &op_name,  
2                           vector<string> arg);
```

cpp

Описание

Вызывает строковую операцию из словаря операций.

Параметры:

- `op_name` – имя вызываемой операции
- `arg` – вектор строковых аргументов для операции

Возвращаемое значение:

Результат выполнения операции в виде строки.

Исключения:

- `runtime_error` если операция не найдена в словаре

- `runtime_error` если тип операции не соответствует строковому типу
-

```
1 float sum(vector<float> source);
```

cpp

Описание

Вычисляет сумму всех элементов числового вектора.

Параметры:

- `source` – входной вектор числовых данных

Возвращаемое значение:

Сумма всех элементов вектора.

Сложность:

$O(n)$, где n – количество элементов в векторе.

```
1 float average(vector<float> source);
```

cpp

Описание

Вычисляет среднее арифметическое элементов числового вектора.

Параметры:

- `source` – входной вектор числовых данных

Возвращаемое значение:

Среднее значение элементов вектора.

Примечание:

Функция не проверяет, что вектор не пустой. Вызов с пустым вектором приведет к делению на ноль.

Сложность:

$O(n)$, где n – количество элементов в векторе.

```
1 string concatenate(vector<string> source);
```

cpp

Описание

Объединяет все строки из вектора в одну строку (конкатенация).

Параметры:

- `source` – входной вектор строк

Возвращаемое значение:

Результирующая объединенная строка.

Примечание:

Для пустого вектора возвращается пустая строка.

Сложность:

$O(n \times m)$, где n – количество строк, m – средняя длина строк.

3.3.2. Файл `graph.cpp`

Глобальные переменные:

- `Node *first_head_ptr = nullptr` – указатель на первый (корневой) узел графа
- `Node *last_head_ptr = nullptr` – указатель на последний узел в списке заголовков
- `int nodes_total = 0` – общее количество узлов в графе

```
1 int nodesTotal();
```

cpp**Описание**

Возвращает общее количество узлов в графе.

Возвращаемое значение:

Количество узлов в графе.

```
1 Node *firstNode();
```

cpp**Описание**

Возвращает указатель на первый (корневой) узел графа.

Возвращаемое значение:

Указатель на корневой узел или `nullptr`, если граф пуст.

```
1 Node *lastNode();
```

cpp

Описание

Возвращает указатель на последний узел графа в списке заголовков.

Возвращаемое значение:

Указатель на последний узел или nullptr, если график пуст.

```
1 bool graphIsEmpty();
```

cpp

Описание

Проверяет, содержит ли график какие-либо узлы.

Возвращаемое значение:

true если график пуст, false если содержит узлы.

```
1 Node *createNode(string id);
```

cpp

Описание

Создает новый узел графа с заданным идентификатором.

Параметры:

- `id` – уникальный строковый идентификатор узла

Возвращаемое значение:

Указатель на созданный узел.

```
1 Adjent *createAdjent(Node *node);
```

cpp

Описание

Создает новый элемент списка смежности, указывающий на заданный узел.

Параметры:

- `node` – указатель на заголовок, соответствующий этой вершине

Возвращаемое значение:

Указатель на созданный элемент списка смежности.

```
1 bool alreadyInGraph(Node *node);
```

cpp

Описание

Проверяет, содержится ли определенный узел в графе.

Параметры:

- `node` – указатель на проверяемый узел

Возвращаемое значение:

`true` если узел найден в графе, `false` в противном случае.

```
1 bool alreadyInGraph(string id);
```

cpp

Описание

Проверяет, содержится ли узел с заданным идентификатором в графе.

Параметры:

- `id` – уникальный идентификатор узла

Возвращаемое значение:

`true` если узел с таким идентификатором найден, `false` в противном случае.

```
1 Node *getNodeById(string id);
```

cpp

Описание

Находит узел по его идентификатору.

Параметры:

- `id` – уникальный идентификатор узла

Возвращаемое значение:

Указатель на найденный узел или `nullptr`, если узел не найден.

```
1 vector<Node *> adjentNodes(Node *node);
```

cpp

Описание

Получает вектор всех узлов, следующих за указанным узлом (смежные узлы).

Параметры:

- `node` – указатель на исходный узел

Возвращаемое значение:

Вектор указателей на следующие узлы.

```
1 void connect(Node *node,  
2                 vector<Node *> to_another_nodes);
```

cpp

Описание

Связывает узел с другими узлами, создавая направленные ребра.

Параметры:

- `node` – указатель на исходный узел
- `to_another_nodes` – вектор указателей на узлы, в которые нужно попасть из исходного узла

Особенности:

Функция проверяет, что оба узла уже добавлены в граф.

```
1 void connect(string id, vector<string> to_another_ids);
```

cpp

Описание

Связывает узел с другими узлами по их идентификаторам.

Параметры:

- `id` – уникальный идентификатор исходного узла
- `to_another_ids` – вектор идентификаторов узлов, в которые нужно попасть

Особенности:

Функция проверяет, что все узлы уже добавлены в граф.

```
1 Adjent *getPreviousAdjentInList(Node *node,  
2                                     Adjent *adjent);
```

cpp

Описание

Получает предыдущий элемент списка смежности для заданного элемента.

Параметры:

- `node` – указатель на заголовок узла
- `adjent` – элемент списка смежности этого заголовка

Возвращаемое значение:

Указатель на предыдущий элемент списка смежности или `nullptr`, если элемент первый.

```
1 Node *getPreviousNodeInList(Node *node);
```

cpp

Описание

Получает предыдущий узел в списке заголовков графа.

Параметры:

- `node` – указатель на текущий узел

Возвращаемое значение:

Указатель на предыдущий узел или `nullptr`, если узел первый.

```
1 void addNode(Node *node);
```

cpp

Описание

Добавляет новый узел в граф, не связывая его с другими узлами.

Параметры:

- `node` – указатель на добавляемый узел
-

```
1 void deleteAdjencyList(Node *node);
```

cpp

Описание

Полностью удаляет список смежности узла.

Параметры:

- `node` – заголовок, список смежности которого нужно удалить
-

```
1 void deleteAdjent(Node *node, Adjent *adjent);
```

cpp

Описание

Удаляет конкретный элемент из списка смежности узла.

Параметры:

- `node` – узел графа, из которого можно попасть в удаляемый элемент
 - `adjent` – элемент списка смежности для удаления
-

```
1 void deleteNodeFromList(Node *node);
```

cpp

Описание

Удаляет узел из списка заголовков графа.

Параметры:

- `node` – заголовок для удаления из списка
-

```
1 void deleteNode(Node *node);
```

cpp

Описание

Полностью удаляет узел из графа и все связанные с ним связи.

Параметры:

- `node` – указатель на удаляемый узел

Особенности:

Удаляет все связи, ведущие к этому узлу из других узлов, а затем сам узел.

```
1 void clearGraph();
```

cpp

Описание

Полностью очищает граф, удаляя все узлы и связи.

```
1 void deepFirstSearchRecursive(cpp
2     void (*procedure)(Node *node),
3     Node *node,
4     Node **visited_nodes,
5     int *last_visited
6 );
```

Описание

Вспомогательная функция рекурсивного обхода графа в глубину.

Параметры:

- `procedure` – указатель на функцию операции над узлом
 - `node` – указатель на посещаемый узел
 - `visited_nodes` – массив указателей на посещенные узлы
 - `last_visited` – индекс последнего посещенного узла в массиве
-

```
1 void deepFirstSearch(cpp void (*procedure)(Node *node));
```

Описание

Выполняет обход графа в глубину, применяя заданную функцию к каждому узлу.

Параметры:

- `procedure` – указатель на функцию операции над узлом

Особенности:

Функция гарантирует, что каждый узел будет посещен только один раз.

3.3.3. Файл `parser.cpp`

Глобальные константы:

- `const string SYMBOLS = "0123456789->"` – допустимые символы для схемы графа

Пространства имен:

1. config – работа с конфигурационными файлами YAML
 2. table – обработка табличных данных (CSV)
 3. Глобальные функции – работа со схемами графов
-

3.3.3.1. Пространство имен config

Глобальные переменные:

- `std::unique_ptr<TINY_YAML::Yaml> root` – объект дерева конфигурации
 - `const string OPERATIONS = "operations"` – поле доступных операций
 - `const string SOURCE = "source"` – поле файлов CSV для обработки
 - `const string FUNC = "func"` – поле названия функции
 - `const string COLUMN = "column"` – поле номера столбца
 - `const string PATH = "path"` – поле пути к файлу CSV
-

```
1 void load(string path);
```

cpp

Описание

Загружает конфигурацию из YAML-файла.

Параметры:

- `path` – путь к конфигурационному файлу YAML

Исключения:

Ловит и выводит в `stderr` ошибки загрузки конфигурации.

```
1 void clear();
```

cpp

Описание

Освобождает память, занятую объектом конфигурации.

```
1 std::vector<std::string> getIds();
```

cpp

Описание

Получает идентификаторы всех доступных операций из конфигурации.

Возвращаемое значение:

Вектор строковых идентификаторов операций.

Примечание:

Возвращает пустой вектор, если конфигурация не загружена.

```
1 map<string, string> checkFunctions();
```

cpp

Описание

Проверяет наличие операций, указанных в конфигурации, в глобальном словаре операций.

Возвращаемое значение:

Словарь ненайденных функций, где ключ – id операции, значение – имя функции.

Логика работы:

1. Получает все ID операций из конфигурации
 2. Для каждого ID получает имя функции
 3. Проверяет наличие функции в operation_map
 4. Возвращает отсутствующие функции
-

```
1 int getColumnById(string id);
```

cpp

Описание

Получает номер столбца для операции по её идентификатору.

Параметры:

- `id` – уникальный идентификатор операции

Возвращаемое значение:

Номер столбца (0-based) или -1 при ошибке.

Исключения:

Ловит исключения преобразования строки в число.

```
1 string getFuncById(string id);
```

cpp

Описание

Получает имя функции для операции по её идентификатору.

Параметры:

- `id` – уникальный идентификатор операции

Возвращаемое значение:

Имя функции или пустая строка при ошибке.

```
1 string getCSV();
```

cpp

Описание

Получает путь к CSV-файлу для обработки из конфигурации.

Возвращаемое значение:

Путь к CSV-файлу или пустая строка при ошибке.

3.3.4. Пространство имен `table`

Глобальные переменные:

- `vector<vector<string>> table` – двумерный массив для хранения табличных данных

Типы данных:

```
1 enum ColumnType {  
2     NUMERIC,    // числовой тип данных  
3     STRING,     // строковый тип данных  
4     UNKNOWN      // неизвестный тип данных  
5 };
```

cpp

```
1 void read(string path, char delimiter);
```

cpp

Описание

Читает табличные данные из CSV-файла.

Параметры:

- `path` – путь к CSV-файлу
- `delimiter` – разделитель полей

Формат файла:

Каждая строка файла – строка таблицы, поля разделены указанным разделителем.

```
1 void clear();
```

cpp

Описание

Очищает таблицу, освобождая память.

```
1 bool isNumneric(string &s);
```

cpp

Описание

Проверяет, является ли строка числом.

Параметры:

- `s` – проверяемая строка

Возвращаемое значение:

`true` если строка может быть преобразована в число, `false` в противном случае.

Алгоритм:

Использует `std::stod` для проверки возможности преобразования.

```
1 ColumnType getTypeOfColumn(int column);
```

cpp

Описание

Определяет тип данных в указанном столбце таблицы.

Параметры:

- `column` – номер столбца (0-based)

Возвращаемое значение:

Тип данных столбца: `NUMERIC`, `STRING` или `UNKNOWN`.

Логика работы:

Анализирует все строки столбца:

- Если все значения – числа → NUMERIC
 - Если есть хотя бы одна строка → STRING
 - В противном случае → UNKNOWN
-

```
1 vector<float> readNumericColumn(int column);
```

cpp

Описание

Считывает столбец как вектор чисел с плавающей точкой.

Параметры:

- `column` – номер столбца (0-based)

Возвращаемое значение:

Вектор значений типа `float`.

Примечание:

Пропускает строки, где столбец выходит за границы.

```
1 vector<string> readStringColumn(int column);
```

cpp

Описание

Считывает столбец как вектор строк.

Параметры:

- `column` – номер столбца (0-based)

Возвращаемое значение:

Вектор строковых значений.

Примечание:

Пропускает строки, где столбец выходит за границы.

Глобальные функции

```
1 vector<string> split(const string &s,
2                           const string &delimiter);
```

cpp

Описание

Разделяет строку на подстроки по указанному разделителю.

Параметры:

- `s` – исходная строка
- `delimiter` – разделитель

Возвращаемое значение:

Вектор подстрок.

Алгоритм:

Итеративно находит все вхождения разделителя.

```
1 int countSubstrOccurrences(string str1, string str2);
```

cpp

Описание

Подсчитывает количество вхождений подстроки в строку (рекурсивная реализация).

Параметры:

- `str1` – строка для поиска
- `str2` – искомая подстрока

Возвращаемое значение:

Количество вхождений подстроки.

Рекурсивный алгоритм:

Базовый случай: если строка короче подстроки, возвращает 0. Рекурсивный шаг: проверяет начало строки и вызывает себя для остатка.

```
1 vector<string> getScheme(std::istream &is);
```

cpp

Описание

Считывает схему графа из входного потока.

Параметры:

- `is` – входной поток (например, `std::cin`)

Возвращаемое значение:

Вектор строк, представляющих схему графа.

Формат ввода:

Строкичитываются до тех пор, пока не встретится строка «end».

Типы ошибок схемы:

```

1 enum SchemeError {
2     EMPTY,           // пустая строка
3     UNEXPECTED_DELIMITER, // разделитель в начале или конце
4     UNEXPECTED_SYMBOLS, // недопустимые символы
5     INDEX_OUT_OF_RANGE // индекс вне диапазона
6 };

```

```

1 vector<vector<SchemeError>> checkScheme(           cpp
2     vector<string> &scheme, vector<string> &ids);

```

Описание

Проверяет корректность схемы графа.

Параметры:

- `scheme` – вектор строк схемы
- `ids` – вектор доступных идентификаторов операций

Возвращаемое значение:

Двумерный вектор ошибок для каждой строки схемы.

Проверки:

1. Пустая строка
 2. Разделитель «-» в начале или конце
 3. Недопустимые символы (не из SYMBOLS)
 4. Индексы вне диапазона [1, `ids.size()`]
-

```

1 void createGraphFromScheme(vector<string> scheme);           cpp

```

Описание

Создает граф по схеме, где узлы обозначены идентификаторами.

Параметры:

- `scheme` – вектор строк схемы

Формат схемы:

"id1->id2->id3" – создает цепочку узлов с указанными идентификаторами.

Алгоритм:

1. Создает узлы по идентификаторам
 2. Добавляет их в граф
 3. Создает связи между последовательными узлами
-

```
1 void createGraphFromScheme(vector<string> scheme,           cpp
2                           vector<string> ids);
```

Описание

Создает граф по схеме, где узлы обозначены индексами идентификаторов.

Параметры:

- `scheme` – вектор строк схемы с индексами
- `ids` – вектор идентификаторов операций

Формат схемы:

"1->2->3" – создает цепочку узлов с идентификаторами `ids[0]`, `ids[1]`, `ids[2]`.

Примечание:

Индексы в схеме начинаются с 1 (человекочитаемый формат).

3.3.5. Файл `utils.cpp`

Глобальные константы:

- `const string SETUP = "setup.conf"` – имя файла конфигурации путей

Пространства имен:

1. `logger` – логирование результатов операций
2. `tui` – текстовый пользовательский интерфейс (меню, ввод, валидация)
3. Глобальные функции – вспомогательные утилиты

Используемые библиотеки:

- `<filesystem>` – работа с файловой системой
 - `<fstream>` – работа с файлами
 - `<unistd.h>` – системные вызовы (для некоторых платформ)
-

3.3.5.1. Пространство имен logger

Глобальные переменные:

- `fstream log` – поток для записи логов

Назначение:

Ведение журнала выполнения операций, запись результатов и предупреждений.

```
1 void openLog(string path);
```

cpp

Описание

Открывает файл лога для записи.

Параметры:

- `path` – путь к файлу лога

Особенности:

Открывает файл в режиме записи (`std::ios::out`). При ошибке открытия выводит сообщение в `std::cerr`.

```
1 void writeResult(string id, float res);
```

cpp

Описание

Записывает результат числовой операции в лог.

Параметры:

- `id` – уникальный идентификатор операции
- `res` – результат выполнения операции (число с плавающей точкой)

Формат записи:

`id >> res`

Пример:

`operation1 >> 42.5`

```
1 void writeResult(string id, string res);
```

cpp

Описание

Записывает результат строковой операции в лог.

Параметры:

- `id` – уникальный идентификатор операции
- `res` – результат выполнения операции (строка)

Формат записи:

`id >> res`

Пример:

```
operation2 >> "concatenated_string"
```

```
1 void warning(string message);
```

cpp

Описание

Записывает предупреждающее сообщение в лог.

Параметры:

- `message` – текст предупреждения

Использование:

Для сообщений об ошибках типа данных, пропущенных операциях и т.д.

```
1 void close();
```

cpp

Описание

Закрывает файл лога.

Особенности:

Проверяет, открыт ли файл, перед закрытием.

3.3.5.2. Пространство имен `tui`

Назначение:

Реализация текстового пользовательского интерфейса для выбора файлов и валидации ввода.

```
1 void displayFileMenu(  
2     const std::vector<std::string> &files);
```

cpp

Описание

Отображает список файлов в виде нумерованного меню.

Параметры:

- `files` – вектор строк с именами файлов

Формат вывода:

```
1 === Доступные файлы ===  
2 [1] file1.csv  
3 [2] file2.csv  
4 [0] Выход  
5 =====
```

```
1 bool isNumber(const std::string &str);
```

cpp

Описание

Проверяет, состоит ли строка только из цифр.

Параметры:

- `str` – входная строка для проверки

Возвращаемое значение:

`true` если строка не пустая и содержит только цифры,
`false` в противном случае.

Алгоритм:

Итеративно проверяет каждый символ с помощью
`std::isdigit`.

```
1 std::string trimString(const std::string &str);
```

cpp

Описание

Удаляет пробельные символы в начале и конце строки.

Параметры:

- `str` – строка для обработки

Возвращаемое значение:

Строка без пробелов по краям или пустая строка, если исходная состояла только из пробелов.

Удаляемые символы:

Пробелы и табуляции (" \t").

```
1 bool processUserInput(std::string &input);
```

cpp

Описание

Обрабатывает ввод пользователя: считывает строку, обрезает пробелы, проверяет на числовой формат.

Параметры:

- `input` – ссылка на строку для сохранения ввода

Возвращаемое значение:

`true` если ввод корректен (число), `false` если нужно повторить ввод.

Последовательность действий:

1. Считывает строку из `std::cin`
 2. Обрезает пробелы
 3. Проверяет, является ли числом
-

```
1 bool safeStringToInt(const std::string &str,
2                           int &result);
```

cpp

Описание

Безопасно преобразует строку в целое число с обработкой исключений.

Параметры:

- `str` – строка для преобразования
- `result` – ссылка для сохранения результата

Возвращаемое значение:

`true` если преобразование успешно, `false` если произошло исключение.

Использует:

std::stoi с обработкой исключений.

```
1 bool isInRange(int number, int min, int max);
```

cpp

Описание

Проверяет, находится ли число в заданном диапазоне.

Параметры:

- `number` – число для проверки
- `min` – минимальное допустимое значение
- `max` – максимальное допустимое значение

Возвращаемое значение:

`true` если $\min \leq \text{number} \leq \max$, `false` в противном случае.

Дополнительно:

При выходе за диапазон выводит сообщение об ошибке.

```
1 std::string selectFileFromList(
```

cpp

```
2 const std::vector<std::string> &files);
```

Описание

Основная функция для интерактивного выбора файла из списка.

Параметры:

- `files` – вектор путей к файлам

Возвращаемое значение:

Выбранный путь к файлу или пустая строка, если выбран выход.

Последовательность действий:

1. Проверяет, не пуст ли список файлов
 2. Отображает меню файлов
 3. Получает выбор пользователя
 4. Возвращает выбранный файл или пустую строку при выборе выхода
-

```
1 int getValidatedNumberInput(const std::string &prompt,  
2                               int min, int max);
```

cpp

Описание

Получает валидированный числовый ввод от пользователя.

Параметры:

- `prompt` – сообщение для ввода (выводится перед ожиданием ввода)
- `min` – минимальное допустимое значение
- `max` – максимальное допустимое значение

Возвращаемое значение:

Валидное целое число в диапазоне $[min, max]$.

Алгоритм валидации:

1. Выводит сообщение
2. Обрабатывает ввод пользователя
3. Проверяет, является ли ввод числом
4. Преобразует строку в число
5. Проверяет диапазон
6. Повторяет до получения валидного ввода

3.3.5.3. Глобальные функции

```
1 vector<string> collectFiles(string path,  
2                                string extension);
```

cpp

Описание

Собирает все файлы с указанным расширением в директории.

Параметры:

- `path` – путь к директории для поиска
- `extension` – расширение файлов (например, «.txt», «.csv»)

Возвращаемое значение:

Вектор полных путей к найденным файлам.

Использует:

`std::filesystem::directory_iterator` для обхода директории.

Примечание:

Проверяет наличие подстроки с расширением в имени файла (не строгое сравнение).

```
1 void procedure(Node *node);
```

c++

Описание

Основная процедура выполнения операций над узлами графа.

Параметры:

- `node` – указатель на узел графа

Алгоритм:

1. Получает идентификатор операции из узла
2. Определяет тип операции через `config::getFuncById`
3. Получает номер столбца через `config::getColumnById`
4. Определяет тип данных в столбце через `table::getTypeOfColumn`
5. В зависимости от типа:
 - Для `NUMERIC`: считывает числовой столбец и вызывает числовую операцию
 - Для `STRING`: считывает строковый столбец и вызывает строковую операцию
 - Для `UNKNOWN`: записывает предупреждение в лог
6. Записывает результат в лог через `logger::writeResult`

Используемые компоненты:

- Конфигурация (`config::`)
- Табличные данные (`table::`)
- Операции (`callOperation`)
- Логирование (`logger::`)

Обработка ошибок:

При неизвестном типе данных записывает предупреждение и пропускает операцию.

Ключевая роль:

Связывает граф операций с фактическим выполнением над данными.

3.3.6. Файл `router.cpp`

Глобальные константы:

- `const string WORKING_DIR = "data"` – рабочая директория для хранения конфигурационных файлов и данных

Пространства имен:

1. `SimpleDAG` – основное пространство имен приложения
2. `SimpleDAG::Internal` – внутренние вспомогательные функции

Назначение файла:

Основной маршрутизатор приложения, координирующий загрузку конфигурации, данных, обработку схемы графа и выполнение операций.

3.3.6.1. Пространство имен `SimpleDAG`

1 `int run();`

`cpp`

Описание

Главная функция запуска приложения `SimpleDAG`.

Возвращаемое значение:

Код завершения программы:

- 0 – успешное выполнение
- 1 – ошибка выполнения

Последовательность выполнения:

1. Загрузка конфигурации (`Internal::loadConfiguration`)
2. Проверка конфигурации (`Internal::validateConfiguration`)
3. Загрузка данных (`Internal::loadDataTable`)
4. Отображение доступных операций
(`Internal::displayAvailableOperations`)
5. Инициализация логирования (`Internal::initializeLogging`)
6. Обработка схемы графа (`Internal::processGraphScheme`)

7. Выполнение операций обходом в глубину
(deepFirstSearch(procedure))
8. Очистка ресурсов (Internal::cleanup)

Ключевые особенности:

- Интегрирует все компоненты системы
- Обеспечивает последовательное выполнение этапов
- Обрабатывает ошибки на каждом этапе

3.3.6.2. Пространство имен SimpleDAG::Internal

Назначение:

Внутренние вспомогательные функции, скрывающие детали реализации от основного интерфейса.

```
1 bool loadConfiguration(const std::string &workingDir); cpp
```

Описание

Загружает конфигурацию из YAML файла в указанной рабочей директории.

Параметры:

- `workingDir` – рабочая директория для поиска конфигурационных файлов

Возвращаемое значение:

`true` если конфигурация успешно загружена, `false` в случае ошибки.

Алгоритм:

1. Собирает все файлы с расширением `.yaml` в рабочей директории
2. Отображает меню выбора файла через `tui::selectFileFromList`
3. Загружает выбранный файл через `config::load`

Сообщения об ошибках:

- «Не найдены конфигурационные файлы в директории ...»
 - «Файл конфигурации не выбран»
-

```
1 bool validateConfiguration();
```

cpp

Описание

Проверяет валидность загруженной конфигурации.

Возвращаемое значение:

true если конфигурация валидна, false если найдены неизвестные функции.

Алгоритм:

1. Проверяет функции через config::checkFunctions
2. Если найдены неизвестные функции, выводит их список
3. Сообщает о необходимости использовать только доступные функции

Формат вывода ошибок:

```
1 Найдены неизвестные функции в файле конфигурации:  
2 id: op1, функция: unknown_func  
3 id: op2, функция: another_unknown
```

```
1 bool loadDataTable();
```

cpp

Описание

Загружает данные из CSV файла, указанного в конфигурации.

Возвращаемое значение:

true если данные успешно загружены, false в случае ошибки.

Алгоритм:

1. Получает имя CSV файла из конфигурации через config::getCSV
2. Формирует полный путь: WORKING_DIR + "/" + csvFile
3. Загружает таблицу через table::read

Сообщения об ошибках:

- «Не указан файл данных в конфигурации»

```
1 void displayAvailableOperations();
```

cpp

Описание

Отображает список доступных операций из конфигурации.

Формат вывода:

```
1 === Доступные операции ===  
2 [1] operation1  
3 [2] operation2  
4 [3] operation3  
5 =====
```

Использует:

config::getIds() для получения списка идентификаторов операций.

```
1 void initializeLogging();
```

cpp

Описание

Инициализирует систему логирования с временной меткой в имени файла.

Алгоритм:

1. Получает текущее время через time(0)
2. Преобразует в строку через ctime
3. Открывает лог-файл в рабочей директории с именем, содержащим временную метку

Пример имени файла:

data/Thu Mar 14 10:30:00 2024

Использует:

logger::openLog() для открытия файла лога.

```
1 bool processGraphScheme(std::vector<std::string> &ids);
```

cpp

Описание

Обрабатывает ввод и валидацию схемы графа от пользователя.

Параметры:

- `ids` – список идентификаторов доступных операций

Возвращаемое значение:

`true` если схема успешно обработана, `false` в случае ошибки.

Последовательность действий:

1. Выводит инструкции (глобальная константа `INSTRUCTIONS`)
2. В цикле:
 - Запрашивает ввод схемы через `getScheme(std::cin)`
 - Проверяет валидность через `validateAndDisplayScheme`
 - Если есть ошибки, просит исправить и повторить ввод
 - Если схема валидна, выходит из цикла
3. Создает граф по схеме через `createGraphFromScheme`

Формат ввода:

Многострочный ввод, завершающийся строкой «`end`»

Интерактивный диалог:

«`Ввод`» → пользователь вводит схему → проверка → сообщение об ошибках или успехе

```
1 bool validateAndDisplayScheme(vector<string> &scheme,           cpp
2                               vector<string> &ids);
```

Описание

Проверяет схему на валидность и отображает ошибки.

Параметры:

- `scheme` – вектор строк схемы графа
- `ids` – список идентификаторов доступных операций

Возвращаемое значение:

`true` если схема валидна, `false` если найдены ошибки.

Алгоритм:

1. Проверяет схему через `checkScheme(scheme, ids)`
2. Для каждой строки с ошибками вызывает `displaySchemeErrors`

3. Возвращает true только если ни в одной строке нет ошибок

Сообщения:

- «Обнаружены ошибки в схеме:» – при наличии ошибок
- «Схема графа корректна!» – при успешной валидации

```
1 void displaySchemeErrors(const std::string &schemeLine,           cpp
2                               const std::vector<SchemeError>
3                               &errors,
4                               int lineNumber);
```

Описание

Отображает ошибки в конкретной строке схемы графа.

Параметры:

- `schemeLine` – строка схемы с ошибкой
- `errors` – вектор ошибок для данной строки
- `lineNumber` – номер строки в схеме (начиная с 1)

Формат вывода:

Строка X "схема": ошибка1; ошибка2; ...

Типы ошибок и их описание:

- `EMPTY` → «пустая строка»
- `UNEXPECTED_DELIMITER` → «неожиданный разделитель»
- `UNEXPECTED_SYMBOLS` → «недопустимые символы»
- `INDEX_OUT_OF_RANGE` → «индекс вне диапазона»

```
1 void cleanup();           cpp
```

Описание

Освобождает все ресурсы и очищает глобальное состояние программы.

Последовательность очистки:

- Конфигурация: `config::clear()`
- Табличные данные: `table::clear()`
- Логирование: `logger::close()`
- Граф: `clearGraph()`

Назначение:

Обеспечивает корректное завершение программы, предотвращая утечки памяти.

Глобальная константа:

```
1 // Константа с инструкциями для пользователя           cpp
2 const string INSSTRUCTIONS = "Введите схему графа в формате:\n"
3                               "1->2->3\n"
4                               "4->5\n"
5                               "... \n"
6                               "end\n"
7                               "Где числа - номера операций из
     списка выше";
```

Структура взаимодействия:

```
1 run()
2   └─ loadConfiguration()
3   └─ validateConfiguration()
4   └─ loadDataTable()
5   └─ displayAvailableOperations()
6   └─ initializeLogging()
7   └─ processGraphScheme()
8     └─ validateAndDisplayScheme()
9       └─ displaySchemeErrors()
10      └─ createGraphFromScheme()
11   └─ deepFirstSearch(procedure)
12   └─ cleanup()
```

4. Инструкция пользователя

4.1. Обзор системы SimpleDAG

SimpleDAG – система для автоматической обработки табличных данных через направленный ациклический граф (DAG) операций. Программа позволяет определить последовательность операций над CSV-данными и выполнить их в указанном порядке.

Основные возможности:

- Загрузка конфигурации операций из YAML-файлов
- Обработка CSV-таблиц с автоматическим определением типов данных
- Построение графа зависимостей между операциями
- Выполнение операций с логированием результатов
- Интерактивный текстовый интерфейс

4.2. Подготовка к работе

4.2.1. Структура директорий

Перед запуском программы убедитесь, что у вас есть следующая структура директорий:

```
1 path/
2 └── SimpleDAG.out          # Исполняемый файл
3 └── data/                  # Директория с данными
4   └── config.yaml          # Файл конфигурации
5   └── data.csv              # CSV-файл с данными
6 └── logs/                  # Директория для логов (создается
    автоматически)
```

4.2.2. Подготовка конфигурационного файла

Создайте YAML-файл конфигурации в директории `data/`.

Пример `config.yaml`:

```
1 operations:                      yaml
2   total_sales:
3     func: sum
4     column: 0
```

```
5   average_price:  
6     func: average  
7     column: 1  
8   product_names:  
9     func: concatenate  
10    column: 2  
11  
12  path: data.csv
```

Поля конфигурации:

- operations – словарь операций, где ключ – уникальный ID операции
 - func – тип операции (sum, average, concatenate)
 - column – номер столбца в CSV (начиная с 0)
- path – имя CSV-файла в директории data/

4.2.3. Подготовка CSV-файла

Создайте CSV-файл в директории data/. Пример data.csv:

```
1 10.5,25.99,ProductA  
2 15.2,32.50,ProductB  
3 8.7,18.75,ProductC
```

csv

Требования к CSV:

- Разделитель – запятая
- Первая строка – данные (заголовки не поддерживаются)
- Колонки могут содержать числа или строки
- Кодировка – UTF-8

4.3. Рабочий процесс

4.3.1. Шаг 1: Выбор конфигурационного файла

Программа автоматически найдет все YAML-файлы в директории data/ и отобразит меню выбора:

```
1 === Доступные файлы ===  
2 [1] config1.yaml  
3 [2] config2.yaml  
4 [0] Выход
```

5 =====

6 Выберите файл (0-2):

Действия:

- Введите номер нужного файла и нажмите Enter
- Для выхода введите 0

4.3.2. Шаг 2: Проверка конфигурации

После выбора файла система:

1. Проверит корректность YAML-синтаксиса
2. Убедится, что указанные функции существуют
3. Проверит наличие указанного CSV-файла

В случае ошибок будут выведены соответствующие сообщения.

4.3.3. Шаг 3: Просмотр доступных операций

Система отобразит список операций из конфигурации:

1 === Доступные операции ===

2 [1] total_sales

3 [2] average_price

4 [3] product_names

5 =====

4.3.4. Шаг 4: Ввод схемы графа

Введите схему зависимостей между операциями. Система отобразит инструкции:

1 Введите схему графа в формате:

2 1->2->3

3 4->5

4 ...

5 end

Где числа - номера операций из списка выше

Формат ввода:

- Каждая строка – цепочка операций

- \rightarrow – обозначает зависимость (операция слева должна выполниться перед операцией справа)
- end – завершение ввода

Примеры:

1. Линейная цепочка:

```
1 1->2->3
2 end
```

Выполнит операции 1, 2, 3 последовательно.

2. Несколько независимых цепочек:

```
1 1->2
2 3->4
3 end
```

Выполнит цепочку (1,2), но (3,4) пропустит. У графа операций первый узел должен быть общим.

3. Сложная схема:

```
1 1->2->4
2 1->3->4
3 end
```

Операция 1 выполнится первой, затем 2, 3 и только потом 4.

4.3.5. Шаг 5: Валидация схемы

Система проверит введенную схему на:

1. Корректность формата
2. Корректность номеров операций

При обнаружении ошибок будут выведены сообщения.

Пример:

```
1 × Обнаружены ошибки в схеме:
2 Стока 1 "1->2->": неожиданный разделитель;
3 Стока 3 "5->6": индекс вне диапазона;
```

Исправьте ошибки и введите схему заново.

4.3.6. Шаг 6: Выполнение операций

После успешной валидации система:

1. Построит граф операций
2. Выполнит операции в правильном порядке
3. Запишет результаты в лог-файл

Отобразится сообщение: Схема графа корректна!

4.3.7. Шаг 7: Просмотр результатов

Результаты выполнения сохраняются в файл лога в директории `data/` с именем, содержащим временную метку (например: `data/Mon Dec 1 22:04:19 2025`).

Содержимое лог-файла:

```
1 total_sales >> 34.4
2 average_price >> 25.7467
3 product_names >> ProductAProductBProductC
```

Формат: ID_операции >> результат

4.4. Работа с ошибками

4.4.1. Ошибки загрузки конфигурации

Ошибка	Причина	Решение
«Не найдены конфигурационные файлы»	В директории <code>data/</code> нет YAML-файлов	Разместите конфигурационный файл в <code>data/</code>
«Файл конфигурации не выбран»	Пользователь выбрал выход (0)	Запустите программу заново
«Найдены неизвестные функции»	В конфигурации указана несуществующая функция	Используйте только <code>sum</code> , <code>average</code> , <code>concatenate</code>

4.4.2. Ошибки ввода схемы

Ошибка	Пример	Исправление
Пустая строка	(пустая строка)	Удалите пустую строку
Неожиданный разделитель	->2->3 или 1->	Убедитесь, что разделитель не в начале/конце
Недопустимые символы	1->a->3	Используйте только цифры и ->
Индекс вне диапазона	1->10 при 5 операциях	Используйте номера из списка операций

4.4.3. Ошибки выполнения операций

Ситуация	Поведение системы
Нечисловые данные в числовой колонке	Пропуск строки, предупреждение в лог
Пустая колонка	Возврат нуля/пустой строки
Отсутствующий CSV-файл	Сообщение об ошибке, завершение работы

4.5. Пример полного сеанса работы

```
1 $ ./SimpleDAG.out
2
3 === Доступные файлы ===
4 [1] sales_config.yaml
5 [2] inventory_config.yaml
6 [0] Выход
7 =====
8 Выберите файл (0-2): 1
9
10 === Доступные операции ===
11 [1] calculate_total
12 [2] find_average
```

```
13 [3] concatenate_names
14 =====
15
16 Введите схему графа в формате:
17 1->2->3
18 4->5
19 ...
20 end
21 Где числа - номера операций из списка выше
22
23 Ввод
24 1->2->3
25 end
26
27 Схема графа корректна!
```

4.6. Клавиши управления

Клавиша	Действие	Контекст
0-9	Выбор пункта меню	Меню выбора файла
Enter	Подтверждение ввода	Все поля ввода
Ctrl+C	Аварийный выход	В любой момент

4.7. Ограничения системы

1. Максимальное количество операций – ограничено памятью
2. Размер CSV-файла – ограничено памятью
3. Типы данных – только числа и строки
4. Операции – только предопределенный набор
5. Параллельное выполнение – не поддерживается (последовательное)

5. ЛИСТИНГ

5.1. Заголовочные файлы

5.1.1. graph.h

```
1  #ifndef GRAPH_H
2  #define GRAPH_H
3
4  #include <string>
5  #include <vector>
6
7  using std::string;
8  using std::vector;
9
10 struct Adjent;
11 struct Node;
12
13 /**
14  * @brief Структура элемента списка заголовков
15  */
16 struct Node {
17     string id;
18     Adjent *adjency_list_head;
19     Adjent *adjency_list_tail;
20     Node *next_head; // Указатель на следующую вершину в списке заголовков
21 };
22
23 /**
24  * @brief Структура вершины графа
25  */
26 struct Adjent {
27     Node *my_head; // Указатель на заголовок этой вершины
28     Adjent *next_adjent; // Указатель на следующий узел
29 };
30
31 /**
32  * @brief Создает новый узел графа с заданным идентификатором
33  * @param id Уникальный идентификатор узла
34  * @return Указатель на созданный узел
35  */
36 Node *createNode(string id);
37
38 /**
39  * @brief Создает новый элемент списка смежности
40  * @param node Указатель на заголовок, соответствующий этой вершине
41  * @return Указатель на созданный узел
42  */
43 // Adjent *createAdjent(Node *node);
44
```

```

45  /**
46  * @brief Получить количество узлов графа
47  * @return количество узлов графа
48  */
49  int nodesTotal();
50
51 /**
52 * @brief Возвращает указатель на первый (корневой) узел графа
53 * @return Указатель на корневой узел
54 */
55 Node *firstNode();
56
57 /**
58 * @brief Возвращает указатель на последний узел графа в списке заголовков
59 * @return Указатель на последний узел в списке заголовков
60 */
61 Node *lastNode();
62
63 /**
64 * @brief Проверить, есть ли в графе определенный узел
65 * @param node указатель на узел
66 * @return true, если узел найден, false - иначе
67 */
68 bool alreadyInGraph(Node *node);
69
70 /**
71 * @brief Проверить, есть ли в графе определенный узел
72 * @param id уникальный идентификатор узла
73 * @return true, если узел найден, false - иначе
74 */
75 bool alreadyInGraph(string id);
76
77 /*
78 * @brief Получить узел по id
79 * @param id уникальный идентификатор узла
80 * @return указатель на узел, если найден, nullptr - если нет
81 */
82 Node *getNodeById(string id);
83
84 /**
85 * @brief Получает вектор всех узлов, следующих за указанным узлом
86 * @param node Указатель на исходный узел
87 * @return Вектор указателей на следующие узлы
88 */
89 vector<Node *> adjentNodes(Node *node);
90
91 /**
92 * @brief Связать узел с другими
93 * @param node указатель на узел
94 * @param to_another_nodes узлы, в которые нужно попасть из node

```

```

95  /*
96  void connect(Node *node, vector<Node *> to_another_nodes);
97
98  /*
99   * @brief Связать узел с другими
100  * @param id уникальный идентификатор узла
101  * @param to_another_ids вектор кникальных идентификаторов узлов, в которые
102  * нужно попасть из узла с идентификатором id
103  */
104 void connect(string id, vector<string> to_another_ids);
105
106 /**
107  * @brief Добавляет новый узел в граф, не связывая его с другими узлами
108  * @param node Указатель на добавляемый узел
109  */
110 void addNode(Node *node);
111
112 /**
113  * @brief Удаляет узел из графа и все связанные с ним связи
114  * @param node Указатель на удаляемый узел
115  */
116 void deleteNode(Node *node);
117
118 /**
119  * @brief Полностью очищает граф, удаляя все узлы
120  */
121 void clearGraph();
122
123 /**
124  * @brief Проверяет, содержит ли граф какие-либо узлы
125  * @return true если граф пуст, false если содержит узлы
126  */
127 bool graphIsEmpty();
128
129 /**
130  * @brief Пройти один раз по всем узлам графа
131  * @param procedure указатель на функцию операции над узлом
132  */
133 void deepFirstSearch(void (*procedure)(Node *node));
134
135 #endif // !GRAPH_H

```

5.1.2. operations.h

```

1  #ifndef OPERATIONS_H
2  #define OPERATIONS_H
3
4  #include <functional>
5  #include <map>
6  #include <string>

```

cpp

```

7  #include <variant>
8  #include <vector>
9
10 #include "parser.h"
11
12 using std::function;
13 using std::map;
14 using std::string;
15 using std::variant;
16 using std::vector;
17
18 /*
19  * @brief Определение возможных типов функций для словаря операций
20  */
21 using FunctionVariant =
22     variant<function<float(vector<float>)>, // Функции для числовых данных
23             function<string(vector<string>)> // Функции для строковых данных
24         >;
25
26 /**
27  * @brief Словарь операций, поддерживающий различные типы данных через variant
28  * Ключ: идентификатор операции
29  * Значение: функция соответствующего типа
30  */
31 extern map<string, FunctionVariant> operation_map;
32
33 /**
34  * @brief Вызвать операцию из словаря операций
35  * @param op_name указатель на ключ операции
36  * @param arg входной параметр вызываемой операции
37  * @return результат работы вызываемой функции
38  */
39 string callOperation(const string &op_name, vector<string> arg);
40
41 /**
42  * @brief Вызвать операцию из словаря операций
43  * @param op_name указатель на ключ операции
44  * @param arg входной параметр вызываемой операции
45  * @return результат работы вызываемой функции
46  */
47 float callOperation(const string &op_name, vector<float> arg);
48
49 /**
50  * @brief Вычисляет сумму элементов числового вектора
51  * @param source Входной вектор числовых данных
52  * @return Сумма всех элементов
53  */
54 float sum(vector<float> source);
55
56 /**

```

```

57  * @brief Вычисляет среднее арифметическое элементов числового вектора
58  * @param source Входной вектор числовых данных
59  * @return Среднее значение элементов
60  */
61 float average(vector<float> source);
62
63 /**
64  * @brief Объединяет строки из вектора в одну строку
65  * @param source Входной вектор строк
66  * @return Результирующая объединенная строка
67  */
68 string concatenate(vector<string> source);
69
70 /**
71  * @brief Выполняет поиск и замену в векторе строк
72  * @param source Входной вектор строк
73  * @param find Стока для поиска
74  * @param replace Стока для замены
75  * @return Вектор с выполненными заменами
76  */
77 vector<string> findReplace(vector<string> source, string find, string replace);
78
79 #endif // !OPERATIONS_H

```

5.1.3. parser.h

```

1  #ifndef PARSER_H
2  #define PARSER_H
3
4  #include "../libs/Tiny_Yaml/yaml/yaml.hpp"
5  #include <map>
6  #include <string>
7  #include <vector>
8
9  using std::map;
10 using std::string;
11 using std::vector;
12
13 /**
14  * @brief Перечисление типов данных столбцов таблицы
15  */
16 enum ColumnType {
17     NUMERIC, // Столбец содержит только числовые данные
18     STRING, // Столбец содержит только строковые данные
19     UNKNOWN, // Тип данных не определен
20 };
21
22 /**
23  * @brief Перечисление типов возможных ошибок при создании схемы графа
24  */

```

```

25 enum SchenmeError {
26     INDEX_OUT_OF_RANGE, // не возможно извлечь id по индексу
27     UNEXPECTED_DELIMITER, // разделитель найден на в начале или конце строки ввода
28     EMPTY, // пустой ввод
29     UNEXPECTED_SYMBOLS, // найден символ, не являющийся цифрой или разделителем
30 };
31
32 namespace config {
33
34 /**
35 * @brief Загрузить конфигурацию
36 * @param path путь к конфигурационному файлу
37 */
38 void load(string path);
39
40 /**
41 * Очистить объект конфигурации
42 */
43 void clear();
44
45 /**
46 * @brief Проверить наличие операций из конфигурации
47 * @return словарь ненайденных функций, где ключ - id операции, значение -
48 * функция
49 */
50 map<string, string> checkFunctions();
51
52 /**
53 * @brief Получить id всех доступных операций
54 */
55 vector<string> getIds();
56
57 /**
58 * @brief Получить параметры (номера столбцов) по id
59 * @param id уникальный идентификатор операции
60 * @return номер столбца
61 */
62 int getColumnById(string id);
63
64 /**
65 * @brief Получить тип операции
66 * @param id уникальный идентификатор операции
67 * @return тип операции
68 */
69 string getFuncById(string id);
70
71 /**
72 * @brief Получить путь к файлу csv для обработки
73 * @return путь к файлу
74 */

```

```
75  string getCSV();
76  }; // namespace config
77
78  namespace table {
79
80  /**
81   * @brief Инициализировать объект документа
82   * @param path путь к документу
83   */
84  void read(string path, char delimiter = ',');
85
86  /**
87   * @brief Очистить таблицу
88   */
89  void clear();
90
91  /*
92   * @brief Получить id всех доступных операций
93   */
94  vector<string> getId();
95
96  /**
97   * @brief Определить, является ли строка числом
98   * @param s строка
99   * @return true если строка является числом, false - иначе
100  */
101 bool isNummeric(string &s);
102
103 /**
104  * @brief Определяет тип данных столбца
105  * @param reader указатель на объект обработки файла csv
106  * @param head Заголовок столбца
107  * @return Тип данных столбца
108  */
109 ColumnType getTypeOfColumn(int column);
110 /*
111  * @brief Считать столбец чисел из файла
112  * @param reader указатель на "документ"
113  * @param column номер столбца
114  * @return вектор чисел столбца
115  */
116 vector<float> readNumericColumn(int column);
117
118 /*
119  * @brief Считать столбец строк из файла
120  * @param reader указатель на "документ"
121  * @param column номер столбца
122  * @return вектор чисел столбца
123  */
124 vector<string> readStringColumn(int column);
```

```

125
126 }; // namespace table
127
128 /**
129 * @brief Разделить строку на подстроки по разделителю
130 * @param s строка
131 * @param delimiter разделитель
132 * @return вектор подстрок
133 */
134 vector<string> split(const string &s, const string &delimiter);
135
136 /**
137 * @brief Считать схему графа из потокового ввода
138 * @param is указатель на поток ввода @brief Считать схему графа из потокового
139 * ввода
140 */
141 vector<string> getScheme(std::istream &is);
142
143 /**
144 * @brief Проверить ввод схемы
145 * @param scheme схема графа
146 * @param ids id операций
147 * @return вектор ошибок для каждой ветви схемы
148 */
149 vector<vector<SchemeError>> checkScheme(vector<string> &scheme,
150                                         vector<string> &ids);
151
152 /**
153 * @brief Создать график по схеме
154 * @param scheme массив строк
155 */
156 void createGraphFromScheme(vector<string> scheme);
157
158 /**
159 * @brief Создать график по схеме индексов id
160 * @param scheme вектор строк с индексами id операций
161 * @param ids вектор id операций
162 */
163 void createGraphFromScheme(vector<string> scheme, vector<string> ids);
164
165 #endif // !PARSER_H

```

5.1.4. router.h

```

1 #ifndef ROUTER_H
2 #define ROUTER_H
3
4 #include "parser.h"
5 #include <string>
6 #include <vector>

```

cpp

```

7
8 namespace SimpleDAG {
9
10 /**
11  * @brief Главная функция запуска приложения SimpleDAG
12  * @param workingDir Рабочая директория с конфигурационными файлами и данными
13  * @return Код завершения программы (0 - успех, 1 - ошибка)
14 */
15 int run();
16
17 namespace Internal {
18
19 /**
20  * @brief Загружает конфигурацию из YAML файла
21  * @param workingDir Рабочая директория для поиска конфигурационных файлов
22  * @return true если конфигурация успешно загружена, false в случае ошибки
23 */
24 bool loadConfiguration(const std::string &workingDir);
25
26 /**
27  * @brief Проверяет валидность загруженной конфигурации
28  * @return true если конфигурация валидна, false если найдены неизвестные
29  * функции
30 */
31 bool validateConfiguration();
32
33 /**
34  * @brief Загружает данные из CSV файла указанного в конфигурации
35  * @return true если данные успешно загружены, false в случае ошибки
36 */
37 bool loadDataTable();
38
39 /**
40  * @brief Отображает список доступных операций из конфигурации
41 */
42 void displayAvailableOperations();
43
44 /**
45  * @brief Инициализирует систему логирования с временной меткой
46 */
47 void initializeLogging();
48
49 /**
50  * @brief Обрабатывает ввод и валидацию схемы графа от пользователя
51  * @param ids Список идентификаторов доступных операций
52  * @return true если схема успешно обработана, false в случае ошибки
53 */
54 bool processGraphScheme(std::vector<std::string> &ids);
55
56 /**

```

```

57  * @brief Отображает ошибки в строке схемы графа
58  * @param schemeLine Стока схемы с ошибкой
59  * @param errors Вектор ошибок для данной строки
60  * @param lineNumber Номер строки в схеме
61  */
62 void displaySchemeErrors(const std::string &schemeLine,
63                         const std::vector<SchemeError> &errors,
64                         int lineNumber);
65
66 /**
67  * @brief Проверяет схему на валидность и отображает ошибки
68  * @param scheme Вектор строк схемы графа
69  * @param ids Список идентификаторов доступных операций
70  * @return true если схема валидна, false если найдены ошибки
71  */
72 bool validateAndDisplayScheme(std::vector<std::string> &scheme,
73                               std::vector<std::string> &ids);
74
75 /**
76  * @brief Освобождает ресурсы и очищает глобальное состояние
77  */
78 void cleanup();
79 } // namespace Internal
80 } // namespace SimpleDAG
81
82 #endif // !ROUTER_H

```

5.1.5. utils.h

```

1  #ifndef UTILS_H
2  #define UTILS_H
3
4  #include "../include/graph.h"
5  #include "../include/operations.h"
6  #include "../include/parser.h"
7  #include <filesystem>
8  #include <fstream>
9  #include <iostream>
10 #include <ostream>
11 #include <string>
12 #include <unistd.h>
13 #include <vector>
14
15 using std::fstream;
16 using std::ifstream;
17 using std::string;
18 using std::vector;
19
20 namespace logger {
21

```

```

22  /*
23   * @brief Открыть лог
24   * @param путь к файлу лога
25   */
26  void openLog(string path);
27
28  /*
29   * @brief Записать результат выполнения операции над числовым столбцом
30   * @param id уникальный идентификатор операции
31   * @param res результат выполнения
32   */
33  void writeResult(string id, float res);
34
35  /*
36   * @brief Записать результат выполнения операции над строковым столбцом
37   * @param id уникальный идентификатор операции
38   * @param res результат выполнения
39   */
40  void writeResult(string id, string res);
41
42  /*
43   * @brief Записать предупреждения
44   * @param message сообщение
45   */
46  void warning(string message);
47
48  /*
49   * @brief Закрыть лог
50   */
51  void close();
52 } // namespace logger
53
54 // =====
55
56 namespace tui {
57 /**
58  * @brief Напечатать стилизованную надпись в консоли
59  */
60 void printLogo();
61
62 /**
63  * @brief Вывести инструкцию о вводе схемы графа
64  */
65 void printSchemeInstruction();
66
67 /**
68  * @brief Отображает список файлов в виде нумерованного меню
69  * @param files вектор строк с именами файлов
70  */
71 void displayFileMenu(const std::vector<std::string> &files);

```

```

72
73  /**
74   * @brief Проверяет, является ли строка числом
75   * @param str входная строка для проверки
76   * @return true если строка представляет собой число, иначе false
77   */
78  bool isNumber(const std::string &str);
79
80  /**
81   * @brief Удаляет пробелы в начале и конце строки
82   * @param str строка для обработки
83   * @return обработанная строка без пробелов по краям
84   */
85  std::string trimString(const std::string &str);
86
87  /**
88   * @brief Обрабатывает ввод пользователя и проверяет его корректность
89   * @param input ссылка на строку для сохранения ввода
90   * @return true если ввод корректен, false если нужно повторить
91   */
92  bool processUserInput(std::string &input);
93
94  /**
95   * @brief Преобразует строку в число с обработкой исключений
96   * @param str строка для преобразования
97   * @param result ссылка для сохранения результата
98   * @return true если преобразование успешно, false если произошла ошибка
99   */
100 bool safeStringToInt(const std::string &str, int &result);
101
102 /**
103  * @brief Проверяет, находится ли число в допустимом диапазоне
104  * @param number число для проверки
105  * @param min минимальное допустимое значение
106  * @param max максимальное допустимое значение
107  * @return true если число в диапазоне, false если нет
108  */
109 bool isInRange(int number, int min, int max);
110
111 /**
112  * @brief Получает числовой ввод от пользователя с валидацией
113  * @param prompt приглашение для ввода
114  * @param min минимальное допустимое значение
115  * @param max максимальное допустимое значение
116  * @return валидное число в заданном диапазоне
117  */
118 int getValidatedNumberInput(const std::string &prompt, int min, int max);
119
120 /**
121  * @brief Основная функция для выбора файла из списка

```

```

122 * @param files вектор строк с путями к файлам
123 * @return выбранный путь к файлу как строка, или пустая строка если выбран
124 * выход
125 */
126 std::string selectFileFromList(const std::vector<std::string> &files);
127
128 /**
129 * @brief Получает числовой ввод от пользователя с валидацией (версия без
130 * continue)
131 * @param prompt приглашение для ввода
132 * @param min минимальное допустимое значение
133 * @param max максимальное допустимое значение
134 * @return валидное число в заданном диапазоне
135 */
136 int getValidatedNumberInputNoContinue(const std::string &prompt, int min,
137                                         int max);
138 }; // namespace tui
139
140 // =====
141
142 /*
143 * @brief Получить путь к файлу конфигурации, путь к файлу результатов
144 * выполнения операций
145 * @param config путь к файлу конфигурации
146 * @param res_file путь к файлу результатов выполнения операций
147 */
148 void getSourceFiles(string &config, string &res_file);
149
150 /*
151 * @brief Получить все пути файлов в директории с определенным расширением
152 * @param path директория
153 * @param extension расширение
154 * @return вектор путей к найденным файлам
155 */
156 vector<string> collectFiles(string path, string extension);
157
158 /*
159 * @brief Процедура запуска операции в узле графа
160 * @param node указатель на узел графа
161 */
162 void procedure(Node *node);
163
164 #endif // !UTILS_H

```

5.1.6. graphics.h

```

1 #ifndef GRAPHICS_H
2 #define GRAPHICS_H
3
4 #include <string>

```

cpp

```

5
6  using std::string;
7
8  const string LOGO = R"(
9   ██████████ ██████████ ██████████ ██████████
10  ████ ████ ████ ████ ████ ████ ████ ████
11  ████ ████ ████ ████ ████ ████ ████ ████
12  ████ ████ ████ ████ ████ ████ ████ ████
13  ████ ████ ████ ████ ████ ████ ████ ████
14  ████ ████ ████ ████ ████ ████ ████ ████
15 )";
16
17 const string INSTRUCTIONS = R"(
18
19          ВВОД СХЕМЫ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ
20
21
22  ФОРМАТ: индекс1->индекс2->индекс3
23  ПРИМЕР: 1->2->3     или     1->3->5->2
24
25  ИНСТРУКЦИЯ:
26  1. Используйте номера операций из списка выше
27  2. Разделяйте номера стрелками '->'
28  3. Каждая строка - отдельная цепочка выполнения
29  4. Операции выполняются в порядке слева направо
30  5. Для завершения ввода введите 'end'
31
32  ВИЗУАЛИЗАЦИЯ ГРАФА:
33
34  Ввод: 1->2->4
35  1->3->5
36  2->4
37
38  Строит граф:
39
40
41  1 → 2 → 4
42
43  |
44  |
45  ▼
46
47  3 → 5
48
49
50  ПОРЯДОК ВЫПОЛНЕНИЯ:
51  1 → 2 → 4
52  1 → 3 → 5
53  2 → 4 (уже выполнена в первой цепочке)
54

```

```

55  || ▲ ВАЖНО:
56  || • Нумерация операций начинается с 1
57  || • Убедитесь, что все номера существуют в списке операций
58  || • Граф будет построен автоматически на основе связей
59  || • Повторяющиеся связи игнорируются
60
61
62 );";
63
64 #endif // !GRAPHICS_H

```

5.2. Исходные файлы

5.2.1. main.cpp

```

1 #include "include/router.h"
2
3 int main() {
4
5     SimpleDAG::run();
6
7     return 0;
8 }

```

cpp

5.2.2. graph.cpp

```

1  #include "../include/graph.h"
2  #include <string>
3  #include <vector>
4
5  using std::string;
6  using std::vector;
7
8  Node *first_head_ptr = nullptr;
9  Node *last_head_ptr = nullptr;
10 int nodes_total = 0;
11
12 /**
13  * @brief Получить количество узлов графа
14  * @return количество узлов графа
15  */
16 int nodesTotal() { return nodes_total; }
17
18 /**
19  * @brief Возвращает указатель на первый (корневой) узел графа
20  * @return Указатель на корневой узел
21  */
22 Node *firstNode() { return first_head_ptr; }
23
24 /**

```

cpp

```

25  * @brief Возвращает указатель на последний узел графа в списке заголовков
26  * @return Указатель на последний узел в списке заголовков
27  */
28 Node *lastNode() { return last_head_ptr; }
29
30 /**
31  * @brief Проверяет, содержит ли граф какие-либо узлы
32  * @return true если граф пуст, false если содержит узлы
33  */
34 bool graphIsEmpty() { return first_head_ptr == nullptr; }
35
36 /**
37  * @brief Создает новый узел графа с заданным идентификатором
38  * @param id Уникальный идентификатор узла
39  * @return Указатель на созданный узел
40  */
41 Node *createNode(string id) {
42     Node *new_node = new Node;
43     new_node->id = id;
44     new_node->next_head = nullptr;
45     new_node->adjacency_list_head = nullptr;
46     new_node->adjacency_list_tail = nullptr;
47     return new_node;
48 }
49
50 /**
51  * @brief Создает новый элемент списка смежности
52  * @param ptr Указатель на заголовок, соответствующих этой вершине
53  * @return Указатель на созданный узел
54  */
55 Adjent *createAdjent(Node *node) {
56     Adjent *new_adjent = new Adjent;
57     new_adjent->next_adjent = nullptr;
58     new_adjent->my_head = node;
59     return new_adjent;
60 }
61
62 /**
63  * @brief Проверить, есть ли в графе определенный узел
64  * @param node указатель на узел
65  * @return true, если узел найден, false - иначе
66  */
67 bool alreadyInGraph(Node *node) {
68     bool found = false;
69     Node *next = first_head_ptr;
70
71     while (next and (not found)) {
72         if (next == node)
73             found = true;
74         next = next->next_head;

```

```

75     }
76     return found;
77 }
78
79 /**
80  * @brief Проверить, есть ли в графе определенный узел
81  * @param id уникальный идентификатор узла
82  * @return true, если узел найден, false - иначе
83 */
84 bool alreadyInGraph(string id) {
85     bool found;
86     Node *next = first_head_ptr;
87
88     while (next and (not found)) {
89         if (next->id == id)
90             found = true;
91         next = next->next_head;
92     }
93     return found;
94 }
95
96 /**
97  * @brief Получить узел по id
98  * @param id уникальный идентификатор узла
99  * @return указатель на узел, если найден, nullptr - если нет
100 */
101 Node *getNodeById(string id) {
102     bool found = false;
103     Node *next = first_head_ptr;
104     Node *node_by_id = nullptr;
105
106     while (next and (not found)) {
107         if (next->id == id) {
108             found = true;
109             node_by_id = next;
110         }
111         next = next->next_head;
112     }
113     return node_by_id;
114 }
115
116 /**
117  * @brief Получает вектор всех узлов, следующих за указанным узлом
118  * @param node Указатель на исходный узел
119  * @return Вектор указателей на следующие узлы
120 */
121 vector<Node *> adjentNodes(Node *node) {
122     vector<Node *> nodes;
123     Adjent *adjent = node->adjency_list_head;
124

```

```

125     while (adjent) {
126         nodes.push_back(adjent->my_head);
127         adjent = adjent->next_adjoint;
128     }
129     return nodes;
130 }
131
132 /*
133 * @brief Связать узел с другими
134 * @param node указатель на узел
135 * @param to_another_nodes вектор уникальных идентификаторов узлов, в которые
136 * нужно попасть из node
137 */
138 void connect(Node *node, vector<Node *> to_another_nodes) {
139     Adjoint *adjent = nullptr;
140
141     // Проверяем, что node добавлен в граф
142     if (!alreadyInGraph(node)) {
143         return;
144     }
145
146     for (Node *to_node : to_another_nodes) {
147
148         if (alreadyInGraph(to_node)) {
149             adjent = createAdjoint(to_node);
150
151             if (!node->adjency_list_head) {
152                 node->adjency_list_head = adjent;
153                 node->adjency_list_tail = adjent;
154             } else {
155                 node->adjency_list_tail->next_adjoint = adjent;
156                 node->adjency_list_tail = adjent;
157             }
158         }
159     }
160 }
161
162 /*
163 * @brief Связать узел с другими
164 * @param id уникальный идентификатор узла
165 * @param to_another_ids вектор уникальных идентификаторов узлов, в которые
166 * нужно попасть из узла с идентификатором id
167 */
168 void connect(string id, vector<string> to_another_ids) {
169     Adjoint *adjent = nullptr;
170     Node *to_node;
171     Node *node = getNodeById(id);
172
173     // Проверяем, что node добавлен в граф
174     if (!alreadyInGraph(id)) {

```

```

175     return;
176 }
177
178 for (string to_id : to_another_ids) {
179
180     if (alreadyInGraph(to_id)) {
181         to_node = getNodeById(to_id);
182         adjent = createAdjent(to_node);
183
184         if (!node->adjency_list_head) {
185             node->adjency_list_head = adjent;
186             node->adjency_list_tail = adjent;
187         } else {
188             node->adjency_list_tail->next_adjent = adjent;
189             node->adjency_list_tail = adjent;
190         }
191     }
192 }
193 }
194
195 /*
196 * @brief Получить предыдущий элемент списка смежности
197 * @param node указатель на заголовок
198 * @elem элемент списка смежности этого заголовка
199 * @return предыдущий элемент списка смежности
200 */
201 Adjent *getPreviousAdjentInList(Node *node, Adjent *adjent) {
202     if (!node || !adjent || node->adjency_list_head == adjent) {
203         return nullptr;
204     }
205
206     Adjent *current = node->adjency_list_head;
207     while (current && current->next_adjent != adjent) {
208         current = current->next_adjent;
209     }
210     return current;
211 }
212
213 /*
214 * @brief Получить предыдущий элемент списка смежности
215 * @param node указатель на заголовок
216 * @elem элемент списка смежности этого заголовка
217 * @return предыдущий элемент списка смежности
218 */
219 Node *getPreviousNodeInList(Node *node) {
220     if (!first_head_ptr || first_head_ptr == node) {
221         return nullptr;
222     }
223
224     Node *current = first_head_ptr;

```

```

225     while (current && current->next_head != node) {
226         current = current->next_head;
227     }
228     return current;
229 }
230
231 /**
232 * @brief Добавляет новый узел в граф, не связывая его с другими узлами
233 * @param atom Указатель на добавляемый узел
234 * @param to Указатель на узел назначения (по умолчанию - корневой узел)
235 */
236 void addNode(Node *node) {
237     // если граф был пуст
238     if (!first_head_ptr) {
239         first_head_ptr = node;
240         last_head_ptr = node;
241     }
242     // если в графе уже были узлы
243     else {
244         last_head_ptr->next_head = node;
245         last_head_ptr = node;
246     }
247     nodes_total++;
248 }
249
250 /*
251 * @brief Удалить список смежности
252 * @param head заголовок этого списка
253 */
254 void deleteAdjencyList(Node *node) {
255     Adjent *next = node->adjency_list_head;
256     Adjent *current;
257
258     while (next) {
259         current = next;
260         next = next->next_adjent;
261         delete current;
262     }
263     node->adjency_list_head = nullptr;
264     node->adjency_list_tail = nullptr;
265 }
266
267 /*
268 * @brief Удалить элемент списка смежности
269 * @param head узел графа, из которого можно попасть в adjent
270 * @param adjent элемент списка смежности
271 */
272 void deleteAdjent(Node *node, Adjent *adjent) {
273     if (!node || !adjent)
274         return;

```

```

275
276     // Если это первый элемент в списке
277     if (node->adjency_list_head == adjent) {
278         node->adjency_list_head = adjent->next_adjoint;
279         // Если это также последний элемент
280         if (node->adjency_list_tail == adjent) {
281             node->adjency_list_tail = nullptr;
282         }
283     } else {
284         // Ищем предыдущий элемент
285         Adjent *prev = getPreviousAdjointInList(node, adjent);
286         if (prev) {
287             prev->next_adjoint = adjent->next_adjoint;
288             // Если удаляем последний элемент
289             if (node->adjency_list_tail == adjent) {
290                 node->adjency_list_tail = prev;
291             }
292         }
293     }
294     delete adjent;
295 }
296
297 /*
298 * @brief Удалить элемент списка заголовков
299 * @param head заголовок
300 */
301 void deleteNodeFromList(Node *node) {
302     if (!first_head_ptr)
303         return;
304
305     // Если это первый элемент
306     if (first_head_ptr == node) {
307         first_head_ptr = node->next_head;
308         // Если это также последний элемент
309         if (last_head_ptr == node) {
310             last_head_ptr = nullptr;
311         }
312     } else {
313         // Ищем предыдущий элемент
314         Node *prev = getPreviousNodeInList(node);
315         if (prev) {
316             prev->next_head = node->next_head;
317             // Если удаляем последний элемент
318             if (last_head_ptr == node) {
319                 last_head_ptr = prev;
320             }
321         }
322     }
323 }
324

```

```

325 /**
326 * @brief Удаляет узел из графа и все связанные с ним связи
327 * @param element Указатель на удаляемый узел
328 */
329 void deleteNode(Node *node) {
330     if (!node || !alreadyInGraph(node))
331         return;
332
333     // Удаляем все связи, ведущие к этому узлу из других узлов
334     Node *current_head = first_head_ptr;
335     while (current_head) {
336         if (current_head != node) {
337             Adjent *current_adjent = current_head->adjency_list_head;
338             Adjent *prev_adjent = nullptr;
339
340             while (current_adjent) {
341                 if (current_adjent->my_head == node) {
342                     Adjent *to_delete = current_adjent;
343                     current_adjent = current_adjent->next_adjent;
344
345                     if (prev_adjent) {
346                         prev_adjent->next_adjent = current_adjent;
347                     } else {
348                         current_head->adjency_list_head = current_adjent;
349                     }
350
351                     // Обновляем tail если нужно
352                     if (current_head->adjency_list_tail == to_delete) {
353                         current_head->adjency_list_tail = prev_adjent;
354                     }
355
356                     delete to_delete;
357                 } else {
358                     prev_adjent = current_adjent;
359                     current_adjent = current_adjent->next_adjent;
360                 }
361             }
362         }
363         current_head = current_head->next_head;
364     }
365
366     deleteAdjentyList(node);
367     deleteNodeFromList(node);
368     delete node;
369     nodes_total--;
370 }
371
372 /**
373 * @brief Полностью очищает граф, удаляя все узлы
374 */

```

```

375 void clearGraph() {
376     if (!first_head_ptr)
377         return;
378
379     Node *next_head = first_head_ptr;
380     Node *current_head;
381     Adjent *next_adjent = first_head_ptr->adjency_list_head;
382     Adjent *current_adjent;
383
384     while (next_head) {
385
386         while (next_adjent) {
387             current_adjent = next_adjent;
388             next_adjent = next_adjent->next_adjent;
389             delete current_adjent;
390         }
391         current_head = next_head;
392         next_head = next_head->next_head;
393         delete current_head;
394     }
395     first_head_ptr = nullptr;
396     last_head_ptr = nullptr;
397     nodes_total = 0;
398 }
399
400 /*
401 * @brief Вспомогательная функция рекурсивного обхода графа
402 * @param procedure указатель на функцию операции над узлом
403 * @param node указатель на посещаемый узел
404 * @param visited_nodes массив указателей на посещенные узлы
405 * @param last_visited индекс последнего посещенного узла в массиве (изначально
406 * -1)
407 */
408 void deepFirstSearchRecursive(void (*procedure)(Node *node), Node *node,
409                               Node **visited_nodes, int *last_visited) {
410     Adjent *next = node->adjency_list_head;
411
412     for (int i = 0; i <= *last_visited; i++) {
413         if (visited_nodes[i] == node) {
414             return;
415         }
416     }
417     procedure(node);
418     ++(*last_visited);
419     visited_nodes[(*last_visited)] = node;
420
421     while (next) {
422         deepFirstSearchRecursive(procedure, next->my_head, visited_nodes,
423                                 last_visited);
424         next = next->next_adjent;

```

```

425 }
426 }
427 /**
428 * @brief Пройти один раз по всем узлам графа
429 * @param procedure указатель на функцию операции над узлом
430 */
431
432 void deepFirstSearch(void (*procedure)(Node *node)) {
433     if (!first_head_ptr)
434         return;
435
436     Node **visited_nodes = new Node *[nodes_total];
437     int last_visited = -1;
438
439     deepFirstSearchRecursive(procedure, first_head_ptr, visited_nodes,
440                             &last_visited);
441
442     delete[] visited_nodes;
443 }
```

5.2.3. operations.cpp

```

1 #include "../include/operations.h"                                         cpp
2 #include <functional>
3 #include <stdexcept>
4 #include <string>
5 #include <variant>
6 #include <vector>
7
8 using std::function;
9 using std::get_if;
10 using std::runtime_error;
11 using std::string;
12 using std::vector;
13
14 map<string, FunctionVariant> operation_map = {
15     {"sum", function<float(vector<float>)>(sum)},
16     {"average", function<float(vector<float>)>(average)},
17     {"concatinate", function<string(vector<string>)>(concatinate)},
18 };
19
20 /*
21 * @brief Вызвать операцию из словаря операций
22 * @param op_name указатель на ключ операции
23 * @param arg входной параметр вызываемой операции
24 * @return результат работы вызываемой функции
25 */
26 float callOperation(const string &op_name, vector<float> arg) {
27     auto map_elem = operation_map.find(op_name);
28     if (map_elem == operation_map.end())
```

```

29     throw runtime_error("Operation not found");
30
31     auto *func = get_if<function<float(vector<float>)>>(&map_elem->second);
32     if (!func)
33         throw std::runtime_error("Invalid operation type");
34
35     return (*func)(arg);
36 }
37
38 /**
39  * @brief Вызвать операцию из словаря операций
40  * @param op_name указатель на ключ операции
41  * @param arg входной параметр вызываемой операции
42  * @return результат работы вызываемой функции
43 */
44 string callOperation(const string &op_name, vector<string> arg) {
45     auto map_elem = operation_map.find(op_name);
46     if (map_elem == operation_map.end())
47         throw runtime_error("Operation not found");
48
49     auto *func = get_if<function<string(vector<string>)>>(&map_elem->second);
50     if (!func)
51         throw std::runtime_error("Invalid operation type");
52
53     return (*func)(arg);
54 }
55
56 /**
57  * @brief Вычисляет сумму элементов числового вектора
58  * @param source Входной вектор числовых данных
59  * @return Сумма всех элементов
60 */
61 float sum(vector<float> source) {
62     float sum = 0;
63     for (float num : source) {
64         sum += num;
65     }
66     return sum;
67 }
68
69 /**
70  * @brief Вычисляет среднее арифметическое элементов числового вектора
71  * @param source Входной вектор числовых данных
72  * @return Среднее значение элементов
73 */
74 float average(vector<float> source) {
75     float sum = 0;
76     for (float num : source) {
77         sum += num;
78     }

```

```

79     return sum / source.size();
80 }
81
82 /**
83  * @brief Объединяет строки из вектора в одну строку
84  * @param source Входной вектор строк
85  * @return Результирующая объединенная строка
86 */
87 string concatenate(vector<string> source) {
88     string result = "";
89     for (string str : source) {
90         result += str;
91     }
92     return result;
93 }
```

5.2.4. parser.cpp

```

1  #include "../include/parser.h"                                     cpp
2  #include "../include/graph.h"
3  #include "../include/operations.h"
4  #include "../libs/Tiny_Yaml/yaml/yaml.hpp"
5  #include <cstdio>
6  #include <exception>
7  #include <fstream>
8  #include <map>
9  #include <memory>
10 #include <iostream>
11 #include <sstream>
12 #include <string>
13 #include <vector>
14
15 using std::string;
16 using std::vector;
17
18 const string SYMBOLS = "0123456789->";
19
20 namespace config {
21
22     std::unique_ptr<TINY_YAML::Yaml> root; // объект дерева конфигурации
23     const string OPERATIONS = "operations"; // поле доступных операций
24     const string SOURCE = "source";          // поле файлов csv для обработки
25     const string FUNC = "func";             // поле - название функции
26     const string COLUMN = "column";        // поле - номер столбца
27     const string PATH = "path";            // поле - путь к файлу csv для обработки
28
29 /*
30  * @brief Загрузить конфигурацию
31  * @param path путь к конфигурационному файлу
32 */
```

```

33 void load(string path) {
34     try {
35         root = std::make_unique<TINY_YAML::Yaml>(path);
36     } catch (const std::exception &e) {
37         std::cerr << "Error loading config: " << e.what() << std::endl;
38         root.reset();
39     }
40 }
41
42 /*
43 * Очистить объект конфигурации
44 */
45 void clear() { root.reset(); }
46
47 /*
48 * @brief Получить id всех доступных операций
49 * @param вектор id операций
50 */
51 std::vector<std::string> getId() {
52     std::vector<std::string> ids;
53
54     if (!root) {
55         std::cerr << "Config not loaded" << std::endl;
56         return ids;
57     }
58
59     try {
60         TINY_YAML::Node &operationsNode = (*root)[OPERATIONS];
61         ids = operationsNode.getChildIds();
62
63     } catch (const std::exception &e) {
64         std::cerr << "Error getting operation IDs: " << e.what() << std::endl;
65     }
66
67     return ids;
68 }
69
70 /*
71 * @brief Проверить наличие операций из конфигурации
72 * @return словарь ненайденных функций, где ключ - id операции, значение -
73 * функция
74 */
75 map<string, string> checkFunctions() {
76     map<string, string> unknown;
77     string func;
78
79     if (!root) {
80         std::cerr << "Config not loaded" << std::endl;
81         return unknown;
82     }

```

```

83
84     try {
85         vector<string> ids = getId();
86
87         for (string id : ids) {
88             func = getFuncById(id);
89
90             if (operation_map.count(func) == 0) {
91                 unknown.insert({id, func});
92             }
93         }
94     } catch (const std::exception &e) {
95         std::cerr << "Error checking functions: " << e.what() << std::endl;
96     }
97
98     return unknown;
99 }
100 /*
101 * @brief Получить параметры (номера столбцов) по id
102 * @param id уникальный идентификатор операции
103 * @return номер столбца
104 */
105
106 int getColumnById(string id) {
107     if (!root) {
108         std::cerr << "Config not loaded" << std::endl;
109         return -1;
110     }
111
112     try {
113         TINY_YAML::Node &operation = (*root)[OPERATIONS][id];
114
115         if (!operation.hasChild(COLUMN)) {
116             std::cerr << "Operation " << id << " has no column defined" << std::endl;
117             return -1;
118         }
119
120         std::string columnStr = operation[COLUMN].getData<std::string>();
121         return std::stoi(columnStr);
122
123     } catch (const std::exception &e) {
124         std::cerr << "Error getting column for operation " << id << ":" << e.what()
125             << std::endl;
126         return -1;
127     }
128 }
129
130 /*
131 * @brief Получить тип операции
132 * @param id уникальный идентификатор операции

```

```

133     * @return тип операции
134     */
135 string getFuncById(string id) {
136     if (!root) {
137         std::cerr << "Config not loaded" << std::endl;
138         return "";
139     }
140
141     try {
142         TINY_YAML::Node &operation = (*root)[OPERATIONS][id];
143
144         if (!operation.hasChild(FUNC)) {
145             std::cerr << "Operation " << id << " has no function defined"
146                         << std::endl;
147             return "";
148         }
149         return operation[FUNC].getData<std::string>();
150
151     } catch (const std::exception &e) {
152         std::cerr << "Error getting function for operation " << id << ": "
153                         << e.what() << std::endl;
154         return "";
155     }
156 }
157
158 /*
159  * @brief Получить путь к файлу csv для обработки
160  * @return путь к файлу
161  */
162 string getCSV() {
163     if (!root) {
164         std::cerr << "Config not loaded" << std::endl;
165         return "";
166     }
167
168     try {
169         TINY_YAML::Node &data = (*root)[PATH];
170         return data.getData<string>();
171     } catch (const std::exception &e) {
172         std::cerr << "Error getting path to data" << std::endl;
173         return "";
174     }
175 }
176
177
178 }; // namespace config
179
180 // =====
181
182 namespace table {

```

```

183
184 vector<vector<string>> table;
185
186 /**
187 * @brief Инициализировать объект документа
188 * @param path путь к документу
189 */
190 void read(string path, char delimiter) {
191     std::ifstream file(path);
192     string buffer;
193     vector<string> row;
194
195     if (not file.is_open()) {
196         std::cerr << "Failed to open file " << path << "in table::read"
197             << std::endl;
198         return;
199     }
200
201     while (std::getline(file, buffer)) {
202         std::stringstream s(buffer);
203
204         while (std::getline(s, buffer, delimiter)) {
205             row.push_back(buffer);
206         }
207
208         table.push_back(row);
209         row.clear();
210     }
211     file.close();
212 }
213
214 /**
215 * @brief Очистить таблицу
216 */
217 void clear() { table.clear(); }
218
219 /**
220 * @brief Определить, является ли строка числом
221 * @param s строка
222 * @return true если строка является числом, false - иначе
223 */
224 bool isNumeric(string &s) {
225     try {
226         size_t pos;
227         std::stod(s, &pos);
228         return pos == s.length();
229     } catch (const std::invalid_argument &e) {
230         return false;
231     } catch (const std::out_of_range &e) {
232         return false;

```

```

233     }
234 }
235
236 /**
237 * @brief Определяет тип данных столбца
238 * @param reader указатель на объект обработки файла csv
239 * @param head Заголовок столбца
240 * @return Тип данных столбца
241 */
242 ColumnType getTypeOfColumn(int column) {
243     bool has_numbers = false;
244     bool has_strings = false;
245
246     for (vector<string> &row : table) {
247
248         if (column < row.size()) {
249
250             string field = row[column];
251             if (isNumeric(row[column])) {
252                 has_numbers = true;
253             } else {
254                 has_strings = true;
255             }
256         }
257         if (has_numbers and has_strings == false)
258             return NUMERIC;
259         if (has_strings)
260             return STRING;
261     }
262
263     return UNKNOWN;
264 }
265
266 /**
267 * @brief Считать столбец чисел из файла
268 * @param column номер столбца
269 * @return вектор чисел столбца
270 */
271 vector<float> readNumericColumn(int column) {
272     vector<float> values;
273
274     for (vector<string> &row : table) {
275         if (column < row.size())
276             values.push_back(stod(row[column]));
277     }
278     return values;
279 }
280
281 /**
282 * @brief Считать столбец строк из файла

```

```

283 * @param column номер столбца
284 * @return вектор чисел столбца
285 */
286 vector<string> readStringColumn(int column) {
287     vector<string> values;
288
289     for (vector<string> &row : table) {
290         if (column < row.size())
291             values.push_back(row[column]);
292     }
293     return values;
294 }
295 }; // namespace table
296
297 // =====
298
299 /**
300 * @brief Разделить строку на подстроки по разделителю
301 * @param s строка
302 * @param delimiter разделитель
303 * @return вектор подстрок
304 */
305 vector<string> split(const string &s, const string &delimiter) {
306     vector<string> tokens;
307     size_t start = 0;
308     size_t end = s.find(delimiter);
309
310     while (end != string::npos) {
311         tokens.push_back(s.substr(start, end - start));
312         start = end + delimiter.length();
313         end = s.find(delimiter, start);
314     }
315     // добавить последний токен или всю строку, если разделитель не найден
316     tokens.push_back(s.substr(start));
317     return tokens;
318 }
319
320 /*
321 * @brief Подсчитать количество вхождений подстроки в строку
322 * @param n1 строка
323 * @param n2 подстрока
324 * @return количество вхождений
325 */
326 int countSubstrOccurrences(string str1, string str2) {
327     int n1 = str1.length();
328     int n2 = str2.length();
329
330     // основное условие
331     if (n1 == 0 || n1 < n2)
332         return 0;

```

```

333
334     // условие для выхода рекурсии
335     // проверить первую подстроку
336     if (str1.substr(0, n2).compare(str2) == 0)
337         return countSubstrOccurrences(str1.substr(1), str2) + 1;
338
339     // либо вернуть оставшийся индекс
340     return countSubstrOccurrences(str1.substr(1), str2);
341 }
342
343 /*
344 * @brief Считать схему графа из потокового ввода
345 * @param is указатель на поток ввода @brief Считать схему графа из потокового
346 * ввода
347 */
348 vector<string> getScheme(std::istream &is) {
349     vector<string> lines;
350     string line;
351
352     while (std::getline(is, line) && line != "end") {
353         lines.push_back(line);
354     }
355     return lines;
356 }
357
358 /*
359 * @brief Проверить ввод схемы
360 * @param scheme схема графа
361 * @param ids id операций
362 * @return вектор ошибок для каждой ветви схемы
363 */
364 vector<vector<SchenmeError>> checkScheme(vector<string> &scheme,
365                                              vector<string> &ids) {
366     vector<vector<SchenmeError>> total_errors;
367
368     for (string s : scheme) {
369         vector<SchenmeError> errors;
370
371         if (s.empty()) {
372             errors.push_back(EMPTY);
373         } else {
374             // Проверка разделителя в начале или конце
375             if (s.find("->") == 0 || s.rfind("->") == s.length() - 2) {
376                 errors.push_back(UNEXPECTED_DELIMITER);
377             }
378
379             // Проверка на недопустимые символы
380             if (s.find_first_not_of(SYMBOLS) != string::npos) {
381                 errors.push_back(UNEXPECTED_SYMBOLS);
382             }

```

```

383
384     // Проверка индексов
385     vector<string> indexes = split(s, "->");
386     for (string index : indexes) {
387         try {
388             int idx = std::stoi(index);
389             if (idx < 1 || idx > static_cast<int>(ids.size())) {
390                 errors.push_back(INDEX_OUT_OF_RANGE);
391             }
392         } catch (const std::exception &e) {
393             // Ошибка преобразования - индекс не число
394         }
395     }
396 }
397     total_errors.push_back(errors);
398 }
399     return total_errors;
400 }
401
402 /**
403 * @brief Создать граф по схеме
404 * @param scheme массив строк
405 */
406 void createGraphFromScheme(vector<string> scheme) {
407     Node *atom;
408     vector<string> id;
409
410     for (const string node : scheme) {
411         id = split(node, "->");
412         atom = createNode(id[0]);
413         addNode(atom);
414
415         for (size_t i = 1; i < id.size(); i++) {
416             if (!alreadyInGraph(id[i])) {
417                 atom = createNode(id[i]);
418                 addNode(atom);
419             }
420             connect(id[i - 1], {id[i]});
421         }
422     }
423 }
424
425 /**
426 * @todo "1-2", "1>2", "1->2->3", 4->5"
427 * @brief Создать граф по схеме индексов id
428 * @param scheme вектор строк с индексами id операций
429 * @param ids вектор id операций
430 */
431 void createGraphFromScheme(vector<string> scheme, vector<string> ids) {
432     Node *atom;

```

```

433     vector<string> indexes;
434
435     for (const string node : scheme) {
436         indexes = split(node, "->");
437         atom = createNode(ids[std::stoi(indexes[0]) - 1]);
438         addNode(atom);
439
440         for (size_t i = 1; i < indexes.size(); i++) {
441             if (!alreadyInGraph(ids[std::stoi(indexes[i]) - 1])) {
442                 atom = createNode(ids[std::stoi(indexes[i]) - 1]);
443                 addNode(atom);
444             }
445             connect(ids[std::stoi(indexes[i - 1]) - 1],
446                     {ids[std::stoi(indexes[i]) - 1]});
447         }
448     }
449 }
```

5.2.5. router.cpp

```

1  #include "../include/router.h"                                         cpp
2  #include "../include/graph.h"
3  #include "../include/graphics.h"
4  #include "../include/operations.h"
5  #include "../include/parser.h"
6  #include "../include/utils.h"
7  #include <ctime>
8  #include <iostream>
9  #include <string>
10 #include <vector>
11
12 using std::cout;
13 using std::endl;
14 using std::string;
15 using std::vector;
16
17 namespace SimpleDAG {
18
19 const string WORKING_DIR = "data";
20
21 /**
22 * @brief Главная функция запуска приложения SimpleDAG
23 * @param workingDir Рабочая директория с конфигурационными файлами и данными
24 * @return Код завершения программы (0 - успех, 1 - ошибка)
25 */
26 int run() {
27     // Вывести лого
28     cout << LOGO << endl;
29
30     // Загрузить конфигурацию
```

```

31     if (!Internal::loadConfiguration(WORKING_DIR)) {
32         return 1;
33     }
34
35     // Проверить конфигурацию
36     if (!Internal::validateConfiguration()) {
37         return 1;
38     }
39
40     // Загрузить данные
41     if (!Internal::loadDataTable()) {
42         return 1;
43     }
44
45     // Показать доступные операции
46     Internal::displayAvailableOperations();
47
48     // Инициализировать логирование
49     Internal::initializeLogging();
50
51     // Обработать схему графа
52     vector<string> ids = config::getIds();
53     if (!Internal::processGraphScheme(ids)) {
54         Internal::cleanup();
55         return 1;
56     }
57
58     // Запустить выполнение операций
59     deepFirstSearch(procedure);
60
61     Internal::cleanup();
62     return 0;
63 }
64
65 namespace Internal {
66
67 /**
68 * @brief Загружает конфигурацию из YAML файла
69 * @param workingDir Рабочая директория для поиска конфигурационных файлов
70 * @return true если конфигурация успешно загружена, false в случае ошибки
71 */
72 bool loadConfiguration(const std::string &workingDir) {
73     vector<string> configs = collectFiles(workingDir, ".yaml");
74     if (configs.empty()) {
75         cout << "Не найдены конфигурационные файлы в директории " << workingDir
76             << endl;
77         return false;
78     }
79
80     string config = tui::selectFileFromList(configs);

```

```

81     if (config.empty()) {
82         cout << "Файл конфигурации не выбран" << endl;
83         return false;
84     }
85
86     config::load(config);
87     return true;
88 }
89
90 /**
91 * @brief Проверяет валидность загруженной конфигурации
92 * @return true если конфигурация валидна, false если найдены неизвестные
93 * функции
94 */
95 bool validateConfiguration() {
96     auto unknown_functions = config::checkFunctions();
97     if (!unknown_functions.empty()) {
98         cout << "Найдены неизвестные функции в файле конфигурации:" << endl;
99
100    for (const auto &func : unknown_functions) {
101        cout << "id: " << func.first << ", функция: " << func.second << endl;
102    }
103
104    cout << "В файле конфигурации могут быть указаны только функции, которые "
105          "есть в этом списке:"
106          << endl;
107
108    for (const auto &func : operation_map) {
109        cout << " - " << func.first << endl;
110    }
111    return false;
112 }
113 return true;
114 }
115
116 /**
117 * @brief Загружает данные из CSV файла указанного в конфигурации
118 * @return true если данные успешно загружены, false в случае ошибки
119 */
120 bool loadDataTable() {
121     string csvFile = config::getCSV();
122     if (csvFile.empty()) {
123         cout << "Не указан файл данных в конфигурации" << endl;
124         return false;
125     }
126
127     string csvPath = WORKING_DIR + "/" + csvFile;
128     table::read(csvPath);
129     return true;
130 }

```

```

131
132 /**
133 * @brief Отображает список доступных операций из конфигурации
134 */
135 void displayAvailableOperations() {
136     cout << endl;
137     vector<string> ids = config::getIds();
138     std::cout << "\n==== Доступные операции ===" << endl;
139     for (size_t i = 0; i < ids.size(); i++) {
140         cout << "[" << i + 1 << "] " << ids[i] << endl;
141     }
142     std::cout << "=====*" << endl;
143 }
144
145 /**
146 * @brief Инициализирует систему логирования с временной меткой
147 */
148 void initializeLogging() {
149     time_t now = time(0);
150     char *dt = ctime(&now);
151     logger::openLog(WORKING_DIR + "/" + dt);
152 }
153
154 /**
155 * @brief Обрабатывает ввод и валидацию схемы графа от пользователя
156 * @param ids Список идентификаторов доступных операций
157 * @return true если схема успешно обработана, false в случае ошибки
158 */
159 bool processGraphScheme(std::vector<std::string> &ids) {
160     cout << INSRUCIONS << endl;
161
162     vector<string> scheme;
163
164     do {
165         std::cout << "Ввод" << std::endl;
166         scheme = getScheme(std::cin);
167
168         // Проверить схему и вывести ошибки если есть
169         if (!validateAndDisplayScheme(scheme, ids)) {
170             cout << "\n△ Пожалуйста, исправьте ошибки и введите схему заново:"
171             << endl;
172         } else {
173             break; // Выходим из цикла если схема валидна
174         }
175
176     } while (true);
177
178     cout << "□ Схема графа корректна!" << endl;
179     createGraphFromScheme(scheme, ids);
180     return true;

```

```

181 }
182
183 /**
184 * @brief Проверяет схему на валидность и отображает ошибки
185 * @param scheme Вектор строк схемы графа
186 * @param ids Список идентификаторов доступных операций
187 * @return true если схема валидна, false если найдены ошибки
188 */
189 bool validateAndDisplayScheme(vector<string> &scheme, vector<string> &ids) {
190     vector<vector<SchenmeError>> schemeErrors = checkScheme(scheme, ids);
191     bool isValid = true;
192
193     for (size_t i = 0; i < schemeErrors.size(); ++i) {
194         if (!schemeErrors[i].empty()) {
195             if (isValid) {
196                 cout << "\n\tОбнаружены ошибки в схеме:" << endl;
197                 isValid = false;
198             }
199             displaySchemeErrors(scheme[i], schemeErrors[i], i + 1);
200         }
201     }
202     return isValid;
203 }
204
205 /**
206 * @brief Отображает ошибки в строке схемы графа
207 * @param schemeLine Стока схемы с ошибкой
208 * @param errors Вектор ошибок для данной строки
209 * @param lineNumber Номер строки в схеме
210 */
211 void displaySchemeErrors(const std::string &schemeLine,
212                         const std::vector<SchenmeError> &errors,
213                         int lineNumber) {
214     cout << "Строка " << lineNumber << " \\" << schemeLine << "\": ";
215
216     for (const auto &error : errors) {
217         switch (error) {
218             case EMPTY:
219                 cout << "пустая строка; ";
220                 break;
221             case UNEXPECTED_DELIMITER:
222                 cout << "неожиданный разделитель; ";
223                 break;
224             case UNEXPECTED_SYMBOLS:
225                 cout << "недопустимые символы; ";
226                 break;
227             case INDEX_OUT_OF_RANGE:
228                 cout << "индекс вне диапазона; ";
229                 break;
230         }

```

```

231     }
232     cout << endl;
233 }
234
235 /**
236 * @brief Освобождает ресурсы и очищает глобальное состояние
237 */
238 void cleanup() {
239     config::clear();
240     table::clear();
241     logger::close();
242     clearGraph();
243 }
244
245 } // namespace Internal
246 } // namespace SimpleDAG

```

5.2.6. utils.cpp

```

1  #include "../include/graph.h"                                     cpp
2  #include "../include/graphics.h"
3  #include "../include/operations.h"
4  #include "../include/parser.h"
5  #include <filesystem>
6  #include <fstream>
7  #include <iostream>
8  #include <ostream>
9  #include <string>
10 #include <unistd.h>
11 #include <vector>
12
13 using std::fstream;
14 using std::string;
15 using std::vector;
16
17 namespace fs = std::filesystem;
18
19 const string SETUP = "setup.conf";
20
21 namespace logger {
22 fstream log;
23
24 /*
25 * @brief Открыть лог
26 * @param путь к файлу лога
27 */
28 void openLog(string path) {
29     log.open(path, std::ios::out); // Исправлено: открываем глобальную переменную
30     if (not log.is_open())
31         std::cerr << "Failed to open log" << std::endl;

```

```

32 }
33
34 /*
35  * @brief Записать результат выполнения операции над числовым столбцом
36  * @param id уникальный идентификатор операции
37  * @param res результат выполнения
38 */
39 void writeResult(string id, float res) {
40     if (log.is_open()) {
41         log << id << " >> " << res << std::endl;
42     }
43 }
44
45 /*
46  * @brief Записать результат выполнения операции над строковым столбцом
47  * @param id уникальный идентификатор операции
48  * @param res результат выполнения
49 */
50 void writeResult(string id, string res) {
51     if (log.is_open()) {
52         log << id << " >> " << res << std::endl;
53     }
54 }
55
56 /*
57  * @brief Записать предупреждения
58  * @param message сообщение
59 */
60 void warning(string message) {
61     if (log.is_open()) {
62         log << message << std::endl;
63     }
64 }
65
66 /*
67  * @brief Закрыть лог
68 */
69 void close() {
70     if (log.is_open()) {
71         log.close();
72     }
73 }
74 }; // namespace logger
75
76 // =====
77
78 namespace tui {
79
80 /**
81  * @brief Отображает список файлов в виде нумерованного меню

```

```

82  * @param files вектор строк с именами файлов
83  */
84  void displayFileMenu(const std::vector<std::string> &files) {
85      std::cout << "\n==== Доступные файлы ====\n";
86      for (size_t i = 0; i < files.size(); ++i) {
87          std::cout << "[" << (i + 1) << "] " << files[i] << "\n";
88      }
89      std::cout << "[0] Выход\n";
90      std::cout << "=====\\n";
91  }
92
93 /**
94  * @brief Проверяет, является ли строка числом
95  * @param str входная строка для проверки
96  * @return true если строка представляет собой число, иначе false
97  */
98  bool isNumber(const std::string &str) {
99      if (str.empty())
100         return false;
101
102     for (char c : str) {
103         if (!std::isdigit(static_cast<unsigned char>(c))) {
104             return false;
105         }
106     }
107     return true;
108 }
109
110 /**
111  * @brief Удаляет пробелы в начале и конце строки
112  * @param str строка для обработки
113  * @return обработанная строка без пробелов по краям
114  */
115 std::string trimString(const std::string &str) {
116     if (str.empty())
117         return str;
118
119     size_t start = str.find_first_not_of(" \t");
120     size_t end = str.find_last_not_of(" \t");
121
122     if (start == std::string::npos)
123         return "";
124
125     return str.substr(start, end - start + 1);
126 }
127
128 /**
129  * @brief Обрабатывает ввод пользователя и проверяет его корректность
130  * @param input ссылка на строку для сохранения ввода
131  * @return true если ввод корректен, false если нужно повторить

```

```

132  /*
133  bool processUserInput(std::string &input) {
134      std::getline(std::cin, input);
135      input = trimString(input);
136
137      if (!isNumber(input)) {
138          std::cout << "x Ошибка: введите число!\n";
139          return false;
140      }
141
142      return true;
143  }
144
145 /**
146 * @brief Преобразует строку в число с обработкой исключений
147 * @param str строка для преобразования
148 * @param result ссылка для сохранения результата
149 * @return true если преобразование успешно, false если произошла ошибка
150 */
151 bool safeStringToInt(const std::string &str, int &result) {
152     try {
153         result = std::stoi(str);
154         return true;
155     } catch (const std::exception &) {
156         std::cout << "x Ошибка: введите корректное число!\n";
157         return false;
158     }
159 }
160
161 /**
162 * @brief Проверяет, находится ли число в допустимом диапазоне
163 * @param number число для проверки
164 * @param min минимальное допустимое значение
165 * @param max максимальное допустимое значение
166 * @return true если число в диапазоне, false если нет
167 */
168 bool isInRange(int number, int min, int max) {
169     if (number >= min && number <= max) {
170         return true;
171     } else {
172         std::cout << "x Ошибка: число должно быть от " << min << " до " << max
173             << "!\n";
174         return false;
175     }
176 }
177
178 /**
179 * @brief Получает числовой ввод от пользователя с валидацией (версия без
180 * continue)
181 * @param prompt приглашение для ввода

```

```

182 * @param min минимальное допустимое значение
183 * @param max максимальное допустимое значение
184 * @return валидное число в заданном диапазоне
185 */
186 int getValidatedNumberInput(const std::string &prompt, int min, int max) {
187     std::string input;
188     int number = min - 1; // Начальное невалидное значение
189
190     while (number < min || number > max) {
191         std::cout << prompt;
192         std::getline(std::cin, input);
193         input = trimString(input);
194
195         if (isNumber(input) && safeStringToInt(input, number)) {
196             if (number >= min && number <= max) {
197                 break; // Валидный ввод - выходим из цикла
198             } else {
199                 std::cout << "✗ Ошибка: число должно быть от " << min << " до " << max
200                         << "!\n";
201                 number = min - 1; // Сбрасываем для продолжения цикла
202             }
203         } else {
204             std::cout << "✗ Ошибка: введите число от " << min << " до " << max
205                         << "!\n";
206         }
207     }
208
209     return number;
210 }
211
212 /**
213 * @brief Основная функция для выбора файла из списка
214 * @param files вектор строк с путями к файлам
215 * @return выбранный путь к файлу как строка, или пустая строка если выбран
216 * выход
217 */
218 std::string selectFileFromList(const std::vector<std::string> &files) {
219     if (files.empty()) {
220         std::cout << "⚠ Список файлов пуст!\n";
221         return "";
222     }
223
224     bool selectionMade = false;
225     std::string selectedFile;
226
227     while (!selectionMade) {
228         displayFileMenu(files);
229
230         std::string prompt =
231             "Выберите файл (0-"
232             + std::to_string(files.size())
233             + "): ";

```

```

232     int choice =
233         getValidatedNumberInput(prompt, 0, static_cast<int>(files.size()));
234
235     if (choice == 0) {
236         std::cout << "Выход из программы.\n";
237         return "";
238     }
239
240     selectedFile = files[choice - 1];
241     std::cout << "Выбран файл: " << selectedFile << "\n";
242     selectionMade = true;
243 }
244
245 return selectedFile;
246 }
247
248 }; // namespace tui
249
250 // =====
251
252 /*
253 * @brief Получить путь к файлу конфигурации, путь к файлу результатов
254 * выполнения операций
255 * @param config путь к файлу конфигурации
256 * @param res_file путь к файлу результатов выполнения операций
257 */
258 void getSourceFiles(string &config, string &res_file) {
259     fstream file(SETUP);
260
261     if (not file.is_open()) {
262         std::cerr << "Failed to open setup file." << std::endl;
263         return;
264     }
265
266     std::getline(file, config);
267     std::getline(file, res_file);
268 }
269
270 /*
271 * @brief Получить все пути файлов в директории с определенным расширением
272 * @param path директория
273 * @param extension расширение
274 * @return вектор путей к найденным файлам
275 */
276 vector<string> collectFiles(string path, string extension) {
277     vector<string> files;
278
279     for (const fs::directory_entry &entry : fs::directory_iterator(path)) {
280         if (entry.path().string().find(extension) != string::npos) {
281             files.push_back(entry.path());

```

```

282     }
283 }
284
285     return files;
286 }
287
288 /*
289 * @brief Процедура запуска операции в узле графа
290 * @param node указатель на узел графа
291 */
292 void procedure(Node *node) {
293     string id = node->id;
294     string type = config::getFuncById(id);
295     int column = config::getColumnById(id);
296
297     switch (table::getTypeOfColumn(column)) {
298
299     case NUMERIC: {
300         vector<float> num_vec = table::readNumericColumn(column);
301         logger::writeResult(id, callOperation(type, num_vec));
302         break;
303     }
304
305     case STRING: {
306         vector<string> str_vec = table::readStringColumn(column);
307         logger::writeResult(id, callOperation(type, str_vec));
308         break;
309     }
310
311     case UNKNOWN: {
312         string message = "Unknown type found in column " + std::to_string(column) +
313                         ". Skipping " + id;
314         logger::warning(message);
315     }
316 }
317 }
```

5.3. Конфигурационные файлы

5.3.1. Makefile (основной)

```

1 SRC = main.cpp \
2       src/router.cpp \
3       src/graph.cpp \
4       src/parser.cpp \
5       src/operations.cpp \
6       src/utils.cpp
7
8 YAML = libs/Tiny_Yaml/yaml/yaml.cpp
9
```

cpp

```
10 default:  
11 g++ -std=c++17 $(SRC) $(YAML) -static -o SimpleDAG.out  
12
```

5.4. Тесты

5.4.1. tests/graph/test_graph.cpp

```
1 #include "../../include/graph.h"  
2 #include "gtest/gtest.h"  
3 #include <algorithm>  
4 #include <string>  
5 #include <vector>  
6  
7 using std::find;  
8 using std::string;  
9 using std::vector;  
10  
11 // =====  
12 // ВСПОМОГАТЕЛЬНЫЕ УТИЛИТЫ ДЛЯ ТЕСТИРОВАНИЯ DFS  
13 // =====  
14 namespace dfs_test_utils {  
15 vector<string> visited_nodes;  
16  
17 void resetVisited() { visited_nodes.clear(); }  
18  
19 void recordVisit(Node *node) { visited_nodes.push_back(node->id); }  
20  
21 bool wasVisited(const string &node_id) {  
22     return find(visited_nodes.begin(), visited_nodes.end(), node_id) !=  
23             visited_nodes.end();  
24 }  
25  
26 int visitCount() { return visited_nodes.size(); }  
27  
28 string firstVisited() { return visited_nodes.empty() ? "" : visited_nodes[0]; }  
29 } // namespace dfs_test_utils  
30  
31 // =====  
32 // ТЕСТЫ СОЗДАНИЯ И БАЗОВЫХ ОПЕРАЦИЙ  
33 // =====  
34  
35 // Тест создания отдельного узла  
36 TEST(GRAPH, CreateNode) {  
37     Node *node = createNode("A");  
38  
39     EXPECT_EQ(node->id, "A");  
40     EXPECT_EQ(node->next_head, nullptr);  
41     EXPECT_EQ(node->adjacency_list_head, nullptr);  
42     EXPECT_EQ(node->adjacency_list_tail, nullptr);
```

```

43
44     delete node;
45 }
46
47 // Тест добавления узлов в граф
48 TEST(GRAPH, AddNode) {
49     Node *node_a = createNode("A");
50     Node *node_b = createNode("B");
51     addNode(node_a);
52     addNode(node_b);
53
54     EXPECT_EQ(firstNode(), node_a);
55     EXPECT_EQ(nodesTotal(), 2);
56     EXPECT_EQ(node_a->next_head, node_b);
57
58     clearGraph();
59 }
60
61 // Тест очистки пустого графа
62 TEST(GRAPH, ClearGraphWithoutConnections) {
63     Node *node = createNode("A");
64     addNode(node);
65     clearGraph();
66
67     EXPECT_EQ(firstNode(), nullptr);
68     EXPECT_EQ(nodesTotal(), 0);
69 }
70
71 // =====
72 // ТЕСТЫ СОЕДИНЕНИЙ И СМЕЖНОСТИ
73 // =====
74
75 // Тест создания связей между узлами
76 TEST(GRAPH, ConnectNodes) {
77     Node *node_a = createNode("A");
78     Node *node_b = createNode("B");
79     Node *node_c = createNode("C");
80     vector<Node *> to_another_nodes = {node_b, node_c};
81
82     addNode(node_a);
83     addNode(node_b);
84     addNode(node_c);
85     connect(node_a, to_another_nodes);
86
87     EXPECT_EQ(node_a->adjacency_list_head->my_head, node_b);
88     EXPECT_EQ(node_a->adjacency_list_head->next_adjent->my_head, node_c);
89     EXPECT_EQ(node_a->adjacency_list_tail->my_head, node_c);
90
91     clearGraph();
92 }

```

```

93
94 // Тест создания связей между узлами по id
95 TEST(GRAPH, ConnectNodesById) {
96     Node *node_a = createNode("A");
97     Node *node_b = createNode("B");
98     Node *node_c = createNode("C");
99     vector<string> to_another_nodes = {"B", "C"};
100
101    addNode(node_a);
102    addNode(node_b);
103    addNode(node_c);
104    connect("A", to_another_nodes);
105
106    EXPECT_EQ(node_a->adjacency_list_head->my_head, node_b);
107    EXPECT_EQ(node_a->adjacency_list_head->next_adjent->my_head, node_c);
108    EXPECT_EQ(node_a->adjacency_list_tail->my_head, node_c);
109
110    clearGraph();
111 }
112
113 // Тест получения списка смежных узлов
114 TEST(GRAPH, GetAdjointNodes) {
115     Node *node_a = createNode("A");
116     Node *node_b = createNode("B");
117     Node *node_c = createNode("C");
118     vector<Node *> to_another_nodes = {node_b, node_c};
119
120     addNode(node_a);
121     addNode(node_b);
122     addNode(node_c);
123     connect(node_a, to_another_nodes);
124     vector<Node *> adjent_nodes = adjentNodes(node_a);
125
126     EXPECT_EQ(adjent_nodes, to_another_nodes);
127
128     clearGraph();
129 }
130
131 // Тест очистки графа со связями
132 TEST(GRAPH, ClearGraphWithConnections) {
133     Node *node_a = createNode("A");
134     Node *node_b = createNode("B");
135     Node *node_c = createNode("C");
136     vector<Node *> to_another_nodes = {node_b, node_c};
137
138     addNode(node_a);
139     addNode(node_b);
140     addNode(node_c);
141     connect(node_a, to_another_nodes);
142     clearGraph();

```

```

143
144     EXPECT_EQ(firstNode(), nullptr);
145     EXPECT_EQ(nodesTotal(), 0);
146 }
147
148 // =====
149 // ТЕСТЫ ПРОВЕРОК СОСТОЯНИЯ ГРАФА
150 // =====
151
152 // Тест проверки наличия узла в графе по указателю
153 TEST(GRAPH, AlreadyInGraph) {
154     Node *node = createNode("A");
155     addNode(node);
156
157     EXPECT_EQ(alreadyInGraph(node), true);
158
159     clearGraph();
160 }
161
162 // Тест проверки наличия узла в графе по указателю
163 TEST(GRAPH, AlreadyInGraphById) {
164     Node *a = createNode("A");
165     Node *b = createNode("B");
166     addNode(a);
167     addNode(b);
168
169     EXPECT_TRUE(alreadyInGraph("A"));
170     EXPECT_TRUE(alreadyInGraph("B"));
171
172     clearGraph();
173 }
174
175 // Тест получения указателя на узел по id
176 TEST(GRAPH, GetById) {
177     Node *a = createNode("A");
178     Node *b = createNode("B");
179     addNode(a);
180     addNode(b);
181
182     EXPECT_EQ(getNodeById("A"), a);
183     EXPECT_EQ(getNodeById("B"), b);
184
185     clearGraph();
186 }
187
188 // Тест проверки пустоты графа
189 TEST(GRAPH, GraphIsEmpty) {
190     Node *node = createNode("A");
191
192     EXPECT_EQ(graphIsEmpty(), true);

```

```

193     addNode(node);
194     EXPECT_EQ(graphIsEmpty(), false);
195
196     clearGraph();
197 }
198
199 // =====
200 // ТЕСТЫ УДАЛЕНИЯ УЗЛОВ (СЛОЖНЫЕ СЦЕНАРИИ)
201 // =====
202
203 // Тест удаления узлов без связей в различных позициях
204 TEST(GRAPH, DeleteNodeWithoutConnections) {
205     Node *node_a = createNode("A");
206     Node *node_b = createNode("B");
207     Node *node_c = createNode("C");
208     Node *node_d = createNode("D");
209
210     addNode(node_a);
211     addNode(node_b);
212     addNode(node_c);
213     addNode(node_d);
214
215     // Тест 1: удаление узла из середины (не последний и не первый)
216     deleteNode(node_c);
217     EXPECT_EQ(node_b->next_head, node_d);
218     EXPECT_EQ(nodesTotal(), 3);
219
220     // Тест 2: удаление последнего узла
221     deleteNode(node_d);
222     EXPECT_EQ(node_b->next_head, nullptr);
223     EXPECT_EQ(lastNode(), node_b);
224     EXPECT_EQ(node_b->next_head, nullptr);
225     EXPECT_EQ(nodesTotal(), 2);
226
227     // Тест 3: удаление первого узла
228     deleteNode(node_a);
229     EXPECT_EQ(firstNode(), node_b);
230     EXPECT_EQ(lastNode(), node_b);
231     EXPECT_EQ(node_b->next_head, nullptr);
232     EXPECT_EQ(nodesTotal(), 1);
233
234     // Тест 4: удаление единственного узла
235     deleteNode(node_b);
236     EXPECT_EQ(firstNode(), nullptr);
237     EXPECT_EQ(lastNode(), nullptr);
238     EXPECT_EQ(nodesTotal(), 0);
239
240     clearGraph();
241 }
242

```

```

243 // Тест удаления узлов с активными связями
244 TEST(GRAPH, DeleteNodeWithConnections) {
245     // Создание сложной структуры связей для тестирования
246     Node *node_a = createNode("A");
247     Node *node_b = createNode("B");
248     Node *node_c = createNode("C");
249     Node *node_d = createNode("D");
250     Node *node_e = createNode("E");
251     vector<Node *> to_nodes_b_c_d_e = {node_b, node_c, node_d, node_e};
252     vector<Node *> to_nodes_d_e = {node_d, node_e};
253     vector<Node *> adjent_nodes;
254
255     addNode(node_a);
256     addNode(node_b);
257     addNode(node_c);
258     addNode(node_d);
259     addNode(node_e);
260     connect(node_a, to_nodes_b_c_d_e);
261     connect(node_b, to_nodes_d_e);
262
263     // Тест: удаление узла из середины списка смежности
264     deleteNode(node_d);
265     adjent_nodes = {node_b, node_c, node_e};
266     EXPECT_EQ(adjentNodes(node_a), adjent_nodes);
267     adjent_nodes = {node_e};
268     EXPECT_EQ(adjentNodes(node_b), adjent_nodes);
269
270     // Тест: удаление последнего узла в списке смежности
271     deleteNode(node_e);
272     adjent_nodes = {node_b, node_c};
273     EXPECT_EQ(adjentNodes(node_a), adjent_nodes);
274     EXPECT_TRUE(adjentNodes(node_b).empty());
275     EXPECT_EQ(node_b->adjency_list_head, nullptr);
276     EXPECT_EQ(node_b->adjency_list_tail, nullptr);
277
278     // Тест: удаление первого узла в списке смежности
279     deleteNode(node_b);
280     adjent_nodes = {node_c};
281     EXPECT_EQ(adjentNodes(node_a), adjent_nodes);
282
283     // Тест: удаление единственного узла в списке смежности
284     deleteNode(node_c);
285     EXPECT_TRUE(adjentNodes(node_a).empty());
286     EXPECT_EQ(node_a->adjency_list_head, nullptr);
287     EXPECT_EQ(node_a->adjency_list_tail, nullptr);
288
289     clearGraph();
290 }
291
292 // =====

```

```

293 // ТЕСТЫ ПОИСКА В ГЛУБИНУ (DFS)
294 // =====
295
296 // Тест посещения всех узлов при обходе
297 TEST(GRAPH, DeepFirstSearch_VisitsAllNodes) {
298     Node *a = createNode("A");
299     Node *b = createNode("B");
300     Node *c = createNode("C");
301
302     addNode(a);
303     addNode(b);
304     addNode(c);
305     connect(a, {b, c});
306
307     dfs_test_utils::resetVisited();
308     deepFirstSearch(dfs_test_utils::recordVisit);
309
310     EXPECT_EQ(dfs_test_utils::visitCount(), 3);
311     EXPECT_TRUE(dfs_test_utils::wasVisited("A"));
312     EXPECT_TRUE(dfs_test_utils::wasVisited("B"));
313     EXPECT_TRUE(dfs_test_utils::wasVisited("C"));
314
315     clearGraph();
316 }
317
318 // Тест начала обхода с корневого узла
319 TEST(GRAPH, DeepFirstSearch_StartsFromRoot) {
320     Node *a = createNode("A");
321     Node *b = createNode("B");
322
323     addNode(a);
324     addNode(b);
325     connect(a, {b});
326
327     dfs_test_utils::resetVisited();
328     deepFirstSearch(dfs_test_utils::recordVisit);
329
330     EXPECT_EQ(dfs_test_utils::firstVisited(), "A");
331
332     clearGraph();
333 }
334
335 // Тест обхода пустого графа
336 TEST(GRAPH, DeepFirstSearch_EmptyGraph) {
337     dfs_test_utils::resetVisited();
338     deepFirstSearch(dfs_test_utils::recordVisit);
339
340     EXPECT_EQ(dfs_test_utils::visitCount(), 0);
341 }
342

```

```

343 // Тест обхода графа с одним узлом
344 TEST(GRAPH, DeepFirstSearch_SingleNode) {
345     Node *a = createNode("A");
346     addNode(a);
347
348     dfs_test_utils::resetVisited();
349     deepFirstSearch(dfs_test_utils::recordVisit);
350
351     EXPECT_EQ(dfs_test_utils::visitCount(), 1);
352     EXPECT_TRUE(dfs_test_utils::wasVisited("A"));
353
354     clearGraph();
355 }
356
357 // Тест обхода сложного графа с множественными связями
358 TEST(GRAPH, DeepFirstSearch_ComplexGraph) {
359     // Создание сложного графа для тестирования обхода:
360     // A -> B -> C
361     // |      |
362     // v      v
363     // D -> E
364     Node *a = createNode("A");
365     Node *b = createNode("B");
366     Node *c = createNode("C");
367     Node *d = createNode("D");
368     Node *e = createNode("E");
369
370     addNode(a);
371     addNode(b);
372     addNode(c);
373     addNode(d);
374     addNode(e);
375
376     connect(a, {b, d});
377     connect(b, {c, e});
378     connect(d, {e});
379
380     dfs_test_utils::resetVisited();
381     deepFirstSearch(dfs_test_utils::recordVisit);
382
383     EXPECT_EQ(dfs_test_utils::visitCount(), 5);
384     EXPECT_TRUE(dfs_test_utils::wasVisited("A"));
385     EXPECT_TRUE(dfs_test_utils::wasVisited("B"));
386     EXPECT_TRUE(dfs_test_utils::wasVisited("C"));
387     EXPECT_TRUE(dfs_test_utils::wasVisited("D"));
388     EXPECT_TRUE(dfs_test_utils::wasVisited("E"));
389     EXPECT_EQ(dfs_test_utils::firstVisited(), "A");
390
391     clearGraph();
392 }

```

5.4.2. tests/operations/test_operations.cpp

```
1 #include "../include/operations.h"                                     cpp
2 #include "gtest/gtest.h"
3 #include <string>
4 #include <vector>
5
6 using std::string;
7 using std::vector;
8
9 TEST(OPERATIONS, Sum) {
10    vector<float> source = {1, 2, 3};
11    EXPECT_EQ(sum(source), 6);
12 }
13
14 TEST(OPERATIONS, Average) {
15    vector<float> source = {1, 2, 3};
16    EXPECT_EQ(average(source), 2);
17 }
18
19 TEST(OPERATIONS, Concatinate) {
20    vector<string> source = {"a", "bc", "def"};
21    EXPECT_EQ(concatinate(source), "abcdef");
22 }
23
24 TEST(OPERATIONS, SumFromMap) {
25    vector<float> source = {1, 2, 3};
26    EXPECT_EQ(callOperation("sum", source), 6);
27 }
28
29 TEST(OPERATIONS, AverageFromMap) {
30    vector<float> source = {1, 2, 3};
31    EXPECT_EQ(callOperation("average", source), 2);
32 }
33
34 TEST(OPERATIONS, ConcatinateFromMap) {
35    vector<string> source = {"a", "bc", "def"};
36    EXPECT_EQ(callOperation("concatinate", source), "abcdef");
37 }
```

5.4.3. tests/parser/test_parser.cpp

```
1 #include "../../include/graph.h"                                         cpp
2 #include "../../include/parser.h"
3 #include "gtest/gtest.h"
4 #include <algorithm>
5 #include <filesystem>
6 #include <fstream>
7 #include <sstream>
8 #include <string>
9 #include <vector>
```

```

10
11  using std::map;
12  using std::string;
13  using std::vector;
14
15  namespace fs = std::filesystem;
16
17  bool containsError(const vector<SchenmeError> &errors, SchenmeError target) {
18      return std::find(errors.begin(), errors.end(), target) != errors.end();
19  }
20
21 // Вспомогательные функции для создания временных файлов
22 void createTestCSV(const string &filename) {
23     std::ofstream file(filename);
24     file << "1,!1,a,1\n"
25         << "1,@,2,b,abc\n"
26         << "1,#,3,c,\n"
27         << "1,$,4,d,\n"
28         << "1,%,5,e,\n"
29         << "1,^,6,f,\n"
30         << "1,&,7,g,\n"
31         << "1,*,8,h,\n"
32         << "1,?,9,i,\n";
33     file.close();
34 }
35
36 void createTestYAML(const string &filename) {
37     std::ofstream file(filename);
38     file << "path: test.csv\n"
39         << "operations:\n"
40         << "    sum_elements_of_column_1:\n"
41         << "        func: sum\n"
42         << "        column: 1\n"
43         << "    sum_elements_of_column_2:\n"
44         << "        func: sum\n"
45         << "        column: 2\n"
46         << "    find_average_of_column_1:\n"
47         << "        func: average\n"
48         << "        column: 1\n"
49         << "    find_average_of_column_2:\n"
50         << "        func: average\n"
51         << "        column: 2\n"
52         << "    concat:\n"
53         << "        func: concatenate\n"
54         << "        column: 3\n";
55     file.close();
56 }
57
58 void createYAMLWithUnknownFunc(const string &filename) {
59     std::ofstream file(filename);

```

```

60     file << "operations:\n"
61         << "    sum_elements_of_column_1:\n"
62             << "        func: sum\n"
63             << "        column: 1\n"
64         << "    sum_elements_of_column_2:\n"
65             << "        func: sum\n"
66             << "        column: 2\n"
67         << "    find_average_of_column_1:\n"
68             << "        func: average\n"
69             << "        column: 1\n"
70         << "    find_average_of_column_2:\n"
71             << "        func: average\n"
72             << "        column: 2\n"
73         << "    concat:\n"
74             << "        func: concatenate\n"
75             << "        column: 3\n"
76         << "    unknown:\n"
77             << "        func: smthg\n"
78             << "        column: 3\n";
79     file.close();
80 }
81
82 // Тест считывания схемы графа из потока ввода
83 TEST(PARSER, GetScheme) {
84     vector<string> scheme = {"A->B->C", "B->D->C"};
85     std::stringstream s("A->B->C\nB->D->C\nend\n");
86
87     EXPECT_EQ(scheme, getScheme(s));
88 }
89
90 // Тест разделителя строки
91 TEST(PARSER, Split) {
92     vector<string> splited = split("A->B", "->");
93
94     EXPECT_EQ(splited[0], "A");
95     EXPECT_EQ(splited[1], "B");
96 }
97
98 // Тест составления графа по схеме
99 TEST(PARSER, CreateGraphFromScheme) {
100     vector<string> scheme = {"A->B->C", "B->D->C"};
101
102     createGraphFromScheme(scheme);
103     Node *a = getNodeById("A");
104     Node *b = getNodeById("B");
105     Node *c = getNodeById("C");
106     Node *d = getNodeById("D");
107     vector<Node *> b_adjents = adjentNodes(b);
108
109     EXPECT_EQ(a->adjency_list_head->my_head, b);

```

```

110    EXPECT_EQ(b_adjents[0], c);
111    EXPECT_EQ(b_adjents[1], d);
112    EXPECT_EQ(d->adjacency_list_head->my_head, c);
113
114    clearGraph();
115 }
116
117 // Тест составления графа по схеме
118 TEST(PARSER, CreateGraphFromSchemeWithIndexes) {
119     vector<string> scheme = {"1->2->3", "2->4->3"};
120     vector<string> ids = {"A", "B", "C", "D"};
121
122     createGraphFromScheme(scheme, ids);
123     Node *a = getNodeById("A");
124     Node *b = getNodeById("B");
125     Node *c = getNodeById("C");
126     Node *d = getNodeById("D");
127     vector<Node *> b_adjents = adjacentNodes(b);
128
129     EXPECT_EQ(a->adjacency_list_head->my_head, b);
130     EXPECT_EQ(b_adjents[0], c);
131     EXPECT_EQ(b_adjents[1], d);
132     EXPECT_EQ(d->adjacency_list_head->my_head, c);
133
134     clearGraph();
135 }
136
137 // Тест определения типа столбца таблицы
138 TEST(PARSER, GetTypeOfColumn) {
139     string path = "test_gettype.csv";
140     createTestCSV(path);
141
142     table::read(path);
143     EXPECT_EQ(NUMERIC, table::getTypeOfColumn(0));
144     EXPECT_EQ(STRING, table::getTypeOfColumn(1));
145     EXPECT_EQ(NUMERIC, table::getTypeOfColumn(2));
146     EXPECT_EQ(STRING, table::getTypeOfColumn(3));
147
148     table::clear();
149     fs::remove(path);
150 }
151
152 // Тест чтения числового столбца таблицы
153 TEST(PARSER, ReadNumericColumn) {
154     string path = "test_numeric.csv";
155     createTestCSV(path);
156
157     table::read(path);
158     vector<float> values1 = {1, 1, 1, 1, 1, 1, 1, 1, 1};
159     vector<float> values2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};

```

```

160
161     EXPECT_EQ(values1, table::readNumericColumn(0));
162     EXPECT_EQ(values2, table::readNumericColumn(2));
163
164     table::clear();
165     fs::remove(path);
166 }
167
168 // Тест чтения строкового столбца таблицы
169 TEST(PARSER, ReadStringColumn) {
170     string path = "test_string.csv";
171     createTestCSV(path);
172
173     table::read(path);
174     vector<string> values1 = {"!", "@", "#", "$", "%", "^", "&", "*", "?"};
175     vector<string> values2 = {"a", "b", "c", "d", "e", "f", "g", "h", "i"};
176     vector<string> values3 = {"1", "abc"};
177
178     EXPECT_EQ(values1, table::readStringColumn(1));
179     EXPECT_EQ(values2, table::readStringColumn(3));
180     EXPECT_EQ(values3, table::readStringColumn(4));
181
182     table::clear();
183     fs::remove(path);
184 }
185
186 // Тест сбора всех id в конфигурации
187 TEST(PARSER, GetIds) {
188     string path = "test_getids.yaml";
189     createTestYAML(path);
190
191     vector<string> ids = {"sum_elements_of_column_1", "sum_elements_of_column_2",
192                           "find_average_of_column_1", "find_average_of_column_2",
193                           "concat"};
194
195     config::load(path);
196     EXPECT_EQ(ids, config::getIds());
197     config::clear();
198
199     fs::remove(path);
200 }
201
202 // Тест получения номер столбца для операции по id
203 TEST(PARSER, GetColumnById) {
204     string path = "test_getcolumn.yaml";
205     createTestYAML(path);
206
207     config::load(path);
208     EXPECT_EQ(1, config::getColumnById("sum_elements_of_column_1"));
209     EXPECT_EQ(2, config::getColumnById("sum_elements_of_column_2"));

```

```

210 EXPECT_EQ(1, config::getColumnById("find_average_of_column_1"));
211 EXPECT_EQ(2, config::getColumnById("find_average_of_column_2"));
212 EXPECT_EQ(3, config::getColumnById("concat"));
213 config::clear();
214
215 fs::remove(path);
216 }
217
218 // Тест получения функции по id
219 TEST(PARSER, GetOpTypeById) {
220     string path = "test_getfunc.yaml";
221     createTestYAML(path);
222
223     config::load(path);
224     EXPECT_EQ("sum", config::getFuncById("sum_elements_of_column_1"));
225     EXPECT_EQ("sum", config::getFuncById("sum_elements_of_column_2"));
226     EXPECT_EQ("average", config::getFuncById("find_average_of_column_1"));
227     EXPECT_EQ("average", config::getFuncById("find_average_of_column_2"));
228     EXPECT_EQ("concatinate", config::getFuncById("concat"));
229     config::clear();
230
231     fs::remove(path);
232 }
233
234 // Тест проверки функций из файла конфигурации
235 TEST(PARSER, CheckFunctions) {
236     string correct = "test_correct.yaml";
237     string incorrect = "test_unknown.yaml";
238
239     map<string, string> unknown = {{"unknown", "smthg"}};
240     map<string, string> empty;
241
242     createTestYAML(correct);
243     config::load(correct);
244     EXPECT_EQ(empty, config::checkFunctions());
245     config::clear();
246     fs::remove(correct);
247
248     createYAMLWithUnknownFunc(incorrect);
249     config::load(incorrect);
250     EXPECT_EQ(unknown, config::checkFunctions());
251     config::clear();
252     fs::remove(incorrect);
253 }
254
255 // Компактный тест проверки схемы
256 TEST(PARSER, CheckScheme) {
257     vector<string> ids = {"op1", "op2", "op3", "op4", "op5"};
258     vector<string> scheme1 = {"1->2->3", "2->4"};
259     vector<string> scheme2 = {"1->4", "1->3"};

```

```

260     vector<string> scheme3 = {"->1->2"};
261     vector<string> scheme4 = {"1->a->3"};
262     vector<string> scheme5 = {"1->6->3"};
263     vector<string> scheme6 = {""};
264
265     // Корректные случаи
266     EXPECT_TRUE(checkScheme(scheme1, ids)[0].empty());
267     EXPECT_TRUE(checkScheme(scheme2, ids)[0].empty());
268
269     // Ошибочные случаи
270     EXPECT_TRUE(
271         containsError(checkScheme(scheme3, ids)[0], UNEXPECTED_DELIMITER));
272     EXPECT_TRUE(containsError(checkScheme(scheme4, ids)[0], UNEXPECTED_SYMBOLS));
273     EXPECT_TRUE(containsError(checkScheme(scheme5, ids)[0], INDEX_OUT_OF_RANGE));
274     EXPECT_TRUE(containsError(checkScheme(scheme6, ids)[0], EMPTY));
275 }
276
277 // Тест получения пути к файлу csv для обработки
278 TEST(PARSER, GetCSV) {
279     string path = "test_getCSV.yaml";
280     string csv_path = "test.csv";
281
282     createTestYAML(path);
283     config::load(path);
284
285     EXPECT_EQ(csv_path, config::getCSV());
286
287     config::clear();
288     fs::remove(path);
289 }
```

5.4.4. tests/utils/test_utils.cpp

```

1  #include "../../include/graph.h"
2  #include "../../include/operations.h"
3  #include "../../include/parser.h"
4  #include "../../include/utils.h"
5  #include "gtest/gtest.h"
6  #include <filesystem>
7  #include <fstream>
8  #include <string>
9
10 using std::fstream;
11 using std::string;
12
13 namespace fs = std::filesystem;
14
15 // Вспомогательные функции для создания временных файлов
16 void createTestCSV(const string &filename) {
17     std::ofstream file(filename);
```

cpp

```

18     file << "1,! ,1,a,1\n"
19         << "1,@,2,b,abc\n"
20         << "1,#,3,c,\n"
21         << "1,$,4,d,\n"
22         << "1,%,5,e,\n"
23         << "1,^,6,f,\n"
24         << "1,&,7,g,\n"
25         << "1,*,8,h,\n"
26         << "1,?,9,i,\n";
27     file.close();
28 }
29
30 void createTestYAML(const string &filename) {
31     std::ofstream file(filename);
32     file << "operations:\n"
33         << "    sum_elements_of_column_1:\n"
34             << "        func: sum\n"
35             << "        column: 1\n"
36         << "    sum_elements_of_column_2:\n"
37             << "        func: sum\n"
38             << "        column: 2\n"
39         << "    find_average_of_column_1:\n"
40             << "        func: average\n"
41             << "        column: 1\n"
42         << "    find_average_of_column_2:\n"
43             << "        func: average\n"
44             << "        column: 2\n"
45         << "    concat:\n"
46             << "        func: concatenate\n"
47             << "        column: 3\n";
48     file.close();
49 }
50
51 // Вспомогательная функция чтения содержимого файла
52 string readTxt(string path) {
53     string result;
54     string line;
55     fstream file(path);
56
57     if (not file.is_open()) {
58         std::cerr << "Failed to open txt file in utils test." << std::endl;
59         return "";
60     }
61
62     result.clear();
63     while (std::getline(file, line)) {
64         if (!result.empty()) {
65             result += "\n";
66         }
67         result += line;

```

```

68     }
69     file.close();
70
71     return result;
72 }
73
74 // Вспомогательная функция создания setup.conf файла
75 void createSetupFile(const string &filename, const string &config = "test.yaml",
76                      const string &result = "result.txt") {
77     std::ofstream file(filename);
78     file << config << "\n" << result << "\n";
79     file.close();
80 }
81
82 // Тест чтения названий файлов конфигурации и логирования
83 TEST(UTILS, GetSourceFiles) {
84     string setup_file = "setup.conf";
85     createSetupFile(setup_file, "test_config.yaml", "test_result.txt");
86
87     string config, res_file;
88     getSourceFiles(config, res_file);
89
90     EXPECT_EQ(config, "test_config.yaml");
91     EXPECT_EQ(res_file, "test_result.txt");
92
93     fs::remove(setup_file);
94 }
95
96 // Тест записи результатов обработки числового столбца
97 TEST(UTILS, WriteNumericResult) {
98     string result_file = "test_numeric_result.txt";
99
100    logger::openLog(result_file);
101    logger::writeResult("sum", 1.5f);
102    logger::close();
103
104    EXPECT_EQ("sum >> 1.5", readTxt(result_file));
105    fs::remove(result_file);
106 }
107
108 // Тест записи результатов обработки строкового столбца
109 TEST(UTILS, WriteStringResult) {
110     string result_file = "test_string_result.txt";
111
112     logger::openLog(result_file);
113     logger::writeResult("concatinate", "abc");
114     logger::close();
115
116     EXPECT_EQ("concatinate >> abc", readTxt(result_file));
117     fs::remove(result_file);

```

```

118 }
119
120 // Тест записи предупреждения
121 TEST(UTILS, Warning) {
122     string result_file = "test_warning_result.txt";
123     string message = "warning message";
124
125     logger::openLog(result_file);
126     logger::warning(message);
127     logger::close();
128
129     EXPECT_EQ("warning message", readTxt(result_file));
130     fs::remove(result_file);
131 }
132
133 // Тест процедуры выполнения операции с числовым столбцом
134 TEST(UTILS, ProcedureNumeric) {
135     string setup_file = "setup.conf";
136     string config_file = "test_proc_config.yaml";
137     string result_file = "test_proc_result.txt";
138     string csv_file = "test_proc_data.csv";
139
140     createSetupFile(setup_file, config_file, result_file);
141     createTestCSV(csv_file);
142
143     // Создаем упрощенную конфигурацию для теста
144     std::ifstream config(config_file);
145     config << "operations:\n"
146         << "    test_sum:\n"
147             << "        func: sum\n"
148             << "        column: 0\n"; // Столбец с числами 1
149     config.close();
150
151     config::load(config_file);
152     table::read(csv_file);
153
154     Node *test_node = createNode("test_sum");
155     logger::openLog(result_file);
156     procedure(test_node);
157     logger::close();
158
159     EXPECT_EQ("test_sum >> 9", readTxt(result_file)); // Сумма 9 единиц
160
161     config::clear();
162     table::clear();
163     clearGraph();
164
165     fs::remove(setup_file);
166     fs::remove(config_file);
167     fs::remove(result_file);

```

```

168     fs::remove(csv_file);
169 }
170
171 // Тест процедуры с строковым столбцом
172 TEST(UTILS, ProcedureString) {
173     string setup_file = "setup.conf";
174     string config_file = "test_proc_string_config.yaml";
175     string result_file = "test_proc_string_result.txt";
176     string csv_file = "test_proc_string_data.csv";
177
178     createSetupFile(setup_file, config_file, result_file);
179     createTestCSV(csv_file);
180
181     std::ifstream config(config_file);
182     config << "operations:\n"
183         << "  test_concat:\n"
184             << "    func: concatenate\n"
185             << "    column: 3\n"; // Столбец с буквами а-и
186     config.close();
187
188     config::load(config_file);
189     table::read(csv_file);
190
191     Node *test_node = createNode("test_concat");
192     logger::openLog(result_file);
193     procedure(test_node);
194     logger::close();
195
196     EXPECT_EQ("test_concat >> abcdefghi", readTxt(result_file));
197
198     config::clear();
199     table::clear();
200     clearGraph();
201
202     fs::remove(setup_file);
203     fs::remove(config_file);
204     fs::remove(result_file);
205     fs::remove(csv_file);
206 }
207
208 // Тест процедуры с вычислением среднего
209 TEST(UTILS, ProcedureAverage) {
210     string setup_file = "setup.conf";
211     string config_file = "test_proc_avg_config.yaml";
212     string result_file = "test_proc_avg_result.txt";
213     string csv_file = "test_proc_avg_data.csv";
214
215     createSetupFile(setup_file, config_file, result_file);
216     createTestCSV(csv_file);
217

```

```

218     std::ifstream config(config_file);
219     config << "operations:\n"
220         << "    test_avg:\n"
221             << "        func: average\n"
222                 << "        column: 2\n"; // Столбец с числами 1-9
223     config.close();
224
225     config::load(config_file);
226     table::read(csv_file);
227
228     Node *test_node = createNode("test_avg");
229     logger::openLog(result_file);
230     procedure(test_node);
231     logger::close();
232
233     EXPECT_EQ("test_avg >> 5", readTxt(result_file)); // Среднее 1-9 = 5
234
235     config::clear();
236     table::clear();
237     clearGraph();
238
239     fs::remove(setup_file);
240     fs::remove(config_file);
241     fs::remove(result_file);
242     fs::remove(csv_file);
243 }
244
245 // Тест процедуры с неизвестным типом столбца
246 TEST(UTILS, ProcedureUnknownType) {
247     string setup_file = "setup.conf";
248     string config_file = "test_unknown_config.yaml";
249     string result_file = "test_unknown_result.txt";
250
251     createSetupFile(setup_file, config_file, result_file);
252
253     std::ifstream config(config_file);
254     config << "operations:\n"
255         << "    test_unknown:\n"
256             << "        func: sum\n"
257                 << "        column: 999\n"; // Несуществующий столбец
258     config.close();
259
260     table::clear(); // Очищаем таблицу
261
262     config::load(config_file);
263
264     Node *test_node = createNode("test_unknown");
265     logger::openLog(result_file);
266     procedure(test_node);
267     logger::close();

```

```

268
269     string result = readTxt(result_file);
270     EXPECT_TRUE(result.find("Unknown type found") != string::npos);
271     EXPECT_TRUE(result.find("Skipping test_unknown") != string::npos);
272
273     config::clear();
274     clearGraph();
275
276     fs::remove(setup_file);
277     fs::remove(config_file);
278     fs::remove(result_file);
279 }
```

5.5. Скрипты

5.5.1. buildTests.sh

```

1  #!/bin/bash
2
3  set -e
4
5  echo "Building all tests..."
6  echo "Current directory: $(pwd)"
7
8  # Конфигурация путей для разных ОС
9  if [[ "$OSTYPE" == "darwin"* ]]; then
10    echo "Building on macOS"
11    INCLUDE_FLAGS="-I../../include -I../../libs/Tiny_Yaml/yaml -I/opt/homebrew/include"
12    LIB_FLAGS="-L/opt/homebrew/lib -lgtest -lgtest_main -lpthread"
13    COMPILER="g++ -std=c++17"
14  else
15    echo "Building on Linux"
16    INCLUDE_FLAGS="-I../../include -I../../libs/Tiny_Yaml/yaml"
17    LIB_FLAGS="-lgtest -lgtest_main -lpthread"
18    COMPILER="g++ -std=c++17"
19  fi
20
21 # Функция сборки одного теста
22 build_test() {
23   local test_name="$1"
24   local source_files="$2"
25   local test_dir="$3"
26
27   echo "Building $test_name..."
28
29   cd "tests/$test_dir"
30   $COMPILER $source_files $INCLUDE_FLAGS $LIB_FLAGS -o "${test_name}.out"
31   cd ../..
32
33   if [ -f "tests/$test_dir/${test_name}.out" ]; then
```

```

34     echo "□ $test_name - SUCCESS"
35   else
36     echo "✗ $test_name - FAILED"
37     exit 1
38   fi
39 }
40
41 # Сборка всех тестов
42 echo "Starting test builds..."
43
44 build_test "test_operations" \
45   "test_operations.cpp ../../src/operations.cpp" \
46   "operations"
47
48 build_test "test_graph" \
49   "test_graph.cpp ../../src/graph.cpp" \
50   "graph"
51
52 build_test "test_parser" \
53   "test_parser.cpp ../../src/graph.cpp ../../src/parser.cpp ../../src/
54   operations.cpp ../../libs/Tiny_Yaml/yaml/yaml.cpp" \
55   "parser"
56
57 build_test "test_utils" \
58   "test_utils.cpp ../../src/graph.cpp ../../src/operations.cpp ../../src/
59   utils.cpp ../../src/parser.cpp ../../libs/Tiny_Yaml/yaml/yaml.cpp" \
60   "utils"
61
62 echo "All tests built successfully!"

```

5.5.2. runTests.sh

```

1 #!/bin/bash
2
3 cd tests
4
5 cd operations
6 ./test_operations.out
7 cd ..
8
9 cd graph
10 ./test_graph.out
11 cd ..
12
13 cd parser
14 ./test_parser.out
15 cd ..
16
17 cd utils
18 ./test_utils.out
19 cd ../../..

```

bash

5.5.3. .github/workflows/ci.yml

```
1  name: CI Tests
2
3  on:
4    push:
5      branches: [ main, dev ]
6    pull_request:
7      branches: [ main ]
8
9  jobs:
10   test:
11     name: Run Tests on ${{ matrix.os }}
12     runs-on: ${{ matrix.os }}
13     strategy:
14       matrix:
15         os: [ubuntu-latest, macos-latest]
16
17     steps:
18       - name: Checkout code
19         uses: actions/checkout@v4
20
21       - name: Install dependencies (Ubuntu)
22         if: matrix.os == 'ubuntu-latest'
23         run: |
24           sudo apt-get update
25           sudo apt-get install -y build-essential libgtest-dev
26
27       - name: Install dependencies (macOS)
28         if: matrix.os == 'macos-latest'
29         run: |
30           brew update
31           brew install googletest
32
33       - name: Build GoogleTest (Ubuntu)
34         if: matrix.os == 'ubuntu-latest'
35         run: |
36           cd /usr/src/gtest
37           sudo cmake CMakeLists.txt
38           sudo make
39           sudo cp lib/*.a /usr/lib
40
41       - name: Build tests
42         run: |
43           chmod +x buildTests.sh
44           ./buildTests.sh
45
46       - name: Run tests
47         run: |
48           chmod +x runTests.sh
49           ./runTests.sh
```

```
50
51   - name: Upload test artifacts
52     uses: actions/upload-artifact@v4
53     if: always()
54     with:
55       name: test-executables-${{ matrix.os }}
56       path: |
57         tests/**/*.out
58       retention-days: 7
```

6. Тестирование

6.1. Среда тестирования

- Операционная система: [Arch Linux/Ubuntu-latest/macOS-latest]
- Используемые инструменты:
 - компилятор: gcc version 15.1.1 20250729 (GCC)
 - стандарт языка: C++17
 - CI: GitHub Actions
- Входные данные:
 - YAML файлы конфигурации
 - CSV файлы с тестовыми данными
- Выходные данные:
 - Текстовые файлы с результатами выполнения операций
 - Вывод в консоль

6.2. Ручное тестирование

6.2.1. Проверка директории data на пустоту

- **Цель:** Проверить корректную обработку отсутствия конфигурационных файлов.
- **Предусловие:** Директория data пуста.
- **Шаги выполнения:**
 1. Запустить программу ./SimpleDAG.out.
 2. Наблюдать за выводом программы.
- **Ожидаемый результат:** Программа должна вывести сообщение «Не найдены конфигурационные файлы в директории data».
- **Фактический результат:** Сообщение выводится корректно.
- **Изображение:**

```
› ./SimpleDAG.out
SIMPLEDAG
Не найдены конфигурационные файлы в директории data
› tree
.
└── data
    └── SimpleDAG.out
2 directories, 1 file
A  ↵ ~/programming/cc++/dag/test_app git 2 dev !4
```

Рис. 1.

6.2.2. Выбор файла конфигурации

6.2.2.1. Некорректный ввод (буквы, спецсимволы)

- **Цель:** Проверить обработку некорректного ввода при выборе файла.
- **Предусловие:** В директории data есть конфигурационные файлы.
- **Шаги выполнения:**
 1. Запустить программу.
 2. Ввести буквы или спецсимволы при запросе выбора файла.
 3. Наблюдать за реакцией программы.
- **Ожидаемый результат:** Программа должна вывести сообщение «Ошибка: введите число!».
- **Фактический результат:** Ошибка обрабатывается корректно.
- **Изображение:**

```
> у
> ./SimpleDAG.out

SIMPLE DAG

===== Доступные файлы =====
[1] data/test_conf.yaml
[2] data/test.yaml
[3] data/unknown_func.yaml
[4] data/col_out_of_range.yaml
[0] Выход

===== Выберите файл (0-4): a
✗ Ошибка: введите число!
===== Выберите файл (0-4): %
✗ Ошибка: введите число!
===== Выберите файл (0-4): 2d
✗ Ошибка: введите число!
===== Выберите файл (0-4): 6$ 
✗ Ошибка: введите число!
===== Выберите файл (0-4): []
```

Рис. 2.

6.2.2.2. Введенное число вне диапазона

- **Цель:** Проверить валидацию диапазона вводимых чисел.
- **Предусловие:** В директории data есть конфигурационные файлы.
- **Шаги выполнения:**
 1. Запустить программу.
 2. Ввести число, превышающее количество доступных файлов.
 3. Наблюдать за реакцией программы.
- **Ожидаемый результат:** Программа должна вывести «Ошибка: число должно быть от 0 до N!».
- **Фактический результат:** Ошибка обрабатывается корректно.
- **Изображение:**

Рис. 3.

6.2.2.3. Пустой ввод

- **Цель:** Проверить обработку пустого ввода.
 - **Предусловие:** В директории data есть конфигурационные файлы.
 - **Шаги выполнения:**
 1. Запустить программу.
 2. Нажать Enter без ввода числа.
 3. Наблюдать за реакцией программы.
 - **Ожидаемый результат:** Программа должна повторно запросить ввод.
 - **Фактический результат:** Обработка работает корректно.

6.2.2.4. Ввод «0» (выход)

- **Цель:** Проверить корректный выход из программы.
 - **Предусловие:** В директории data есть конфигурационные файлы.
 - **Шаги выполнения:**
 1. Запустить программу.
 2. Ввести «0» при выборе файла.

3. Наблюдать за поведением программы.

 - **Ожидаемый результат:** Программа должна завершиться с сообщением «Выход из программы. Файл конфигурации не выбран».
 - **Фактический результат:** Выход выполняется корректно.
 - **Изображение:**

Рис. 4.

6.2.3. Ввод схемы графа

6.2.3.1. Недопустимые символы (буквы, спецсимволы)

- **Цель:** Проверить валидацию символов при вводе схемы графа.
 - **Предусловие:** Успешно выбран файл конфигурации test.yaml.
 - **Шаги выполнения:**
 1. Ввести схему с недопустимыми символами: a→2→#.
 2. Нажать Enter, затем ввести end.
 3. Наблюдать за реакцией программы.
 - **Ожидаемый результат:** Программа должна вывести ошибку: «Строка 1 a→2→#: недопустимые символы».
 - **Фактический результат:** Ошибка выводится корректно.
 - **Изображение:**

```
Ввод
a→2→#
end

✗ Обнаружены ошибки в схеме:
Строка 1 "a→2→#": недопустимые символы;

⚠ Пожалуйста, исправьте ошибки и введите схему заново:
Ввод
[]
```

Рис. 5.

6.2.3.2. Разделитель в начале/конце строки

- **Цель:** Проверить обработку некорректного использования разделителей.
- **Предусловие:** Успешно выбран файл конфигурации.
- **Шаги выполнения:**
 1. Ввести схему: →2→3.
 2. Нажать Enter, затем ввести end.
 3. Повторить с 2→3→.
 4. Наблюдать за реакцией программы.
- **Ожидаемый результат:** Программа должна вывести ошибку: «неожиданный разделитель».
- **Фактический результат:** Ошибка обрабатывается корректно.
- **Изображение:**

```
Ввод
→2→3
end

✗ Обнаружены ошибки в схеме:
Строка 1 "->2→3": неожиданный разделитель;

⚠ Пожалуйста, исправьте ошибки и введите схему заново:
Ввод
2→3→
end

✗ Обнаружены ошибки в схеме:
Строка 1 "2→3→": неожиданный разделитель;

⚠ Пожалуйста, исправьте ошибки и введите схему заново:
Ввод
[]
```

Рис. 6.

6.2.3.3. ID операции вне диапазона

- **Цель:** Проверить валидацию номеров операций.
- **Предусловие:** Успешно выбран файл конфигурации test.yaml (5 операций).
- **Шаги выполнения:**
 1. Ввести схему: 1→3→2 и 1→6.
 2. Нажать Enter, затем ввести end.
 3. Наблюдать за реакцией программы.
- **Ожидаемый результат:** Программа должна вывести ошибку: «Строка 2 «1→6»: индекс вне диапазона».
- **Фактический результат:** Ошибка выводится корректно.
- **Изображение:**

Ввод
1→3→2
1→6
end

✗ Обнаружены ошибки в схеме:
Строка 2 "1→6": индекс вне диапазона;

⚠ Пожалуйста, исправьте ошибки и введите схему заново:
Ввод
[]

Рис. 7.

6.2.3.4. Пустая строка

- **Цель:** Проверить обработку пустой строки в схеме.
- **Предусловие:** Успешно выбран файл конфигурации.
- **Шаги выполнения:**
 1. Ввести пустую строку.
 2. Нажать Enter, затем ввести end.
 3. Наблюдать за реакцией программы.
- **Ожидаемый результат:** Программа должна вывести ошибку: «пустая строка».
- **Фактический результат:** Ошибка обрабатывается корректно.
- **Изображение:**

```
Ввод
end
✗ Обнаружены ошибки в схеме:
Строка 1 "": пустая строка;

⚠ Пожалуйста, исправьте ошибки и введите схему заново:
Ввод
[]
```

Рис. 8.

6.2.3.5. Корректный ввод схемы

- **Цель:** Проверить успешный ввод корректной схемы.
- **Предусловие:** Успешно выбран файл конфигурации test.yaml.
- **Шаги выполнения:**
 1. Ввести корректную схему: 1→3→2.
 2. Нажать Enter, затем ввести end.
 3. Наблюдать за реакцией программы.
- **Ожидаемый результат:** Программа должна вывести «Схема графа корректна!».
- **Фактический результат:** Схема принимается корректно.
- **Изображение:**

```
Ввод
1→3→2
end
✓ Схема графа корректна!

▲ ➔ ~/programming/cc++/dag/test_app git 🌐 dev !4 ?1 []
```

Рис. 9.

6.2.4. Проверка конфигурации

6.2.4.1. Недоступная операция в YAML

- **Цель:** Проверить обработку неизвестных операций в конфигурационном файле.

- **Предусловие:** В директории data есть файл unknown_func.yaml.
- **Шаги выполнения:**
 1. Выбрать файл unknown_func.yaml.
 2. Наблюдать за выводом программы.
- **Ожидаемый результат:** Программа должна вывести сообщение о неизвестной функции и список доступных функций.
- **Фактический результат:** Ошибка обрабатывается корректно.
- **Изображение:**

```

> ./SimpleDAG.out
SIMPLEDAG

===== Доступные файлы =====
[1] data/test_conf.yaml
[2] data/test.yaml
[3] data/unknown_func.yaml
[4] data/col_out_of_range.yaml
[0] Выход
=====
Выберите файл (0-4): 3
✓ Выбран файл: data/unknown_func.yaml
Найдены неизвестные функции в файле конфигурации:
id: concat, функция: max
В файле конфигурации могут быть указаны только функции, которые есть в этом списке:
- average
- concatenate
- sum

```

Рис. 10.

- **YAML файл:**

```

1 path: test.csv
2
3 operations:
4
5   sum_elements_of_column_2:

```

```

6     func: sum
7     column: 2
8
9     find_average_of_column_2:
10    func: average
11    column: 2
12
13 concat:
14   func: max
15   column: 3

```

6.2.4.2. Номер столбца вне диапазона

- **Цель:** Проверить обработку неверного номера столбца в CSV.
- **Шаги выполнения:**
 1. Выбрать файл col_out_of_range.yaml.
 2. Ввести корректную схему графа.
 3. Наблюдать за выполнением операций.
- **Ожидаемый результат:** Программа должна обработать ошибку доступа к несуществующему столбцу.
- **Фактический результат:** Программа выводит предупреждение: «Unknown type found in column 9. Skipping concat».
- **Изображение:**

```

~/programming/cc++/dag/test_app/data
  □ data
  □ SimpleDAG.out
  ○ col_out_of_range.yaml
  □ Sun Dec 7 14:27:13 2025
  ○ test.csv
  ○ test.yaml
  ○ test_conf.yaml
  ○ unknown_func.yaml

```

sum_elements_of_column_2 >> 45
Unknown type found in column 9. Skipping concat
find_average_of_column_2 >> 5

Рис. 11.

- **YAML файл:**

```

1   path: test.csv
2
3   operations:
4
5     sum_elements_of_column_2:

```

yaml

```
6     func: sum
7     column: 2
8
9     find_average_of_column_2:
10    func: average
11    column: 2
12
13 concat:
14   func: concatenate
15   column: 9
```

- **CSV файл:**

```
1 1,! ,1,a,1
2 1,@,2,b,abc
3 1,#,3,c,
4 1,$,4,d,
5 1,%,5,e,
6 1,^,6,f,
7 1,&,7,g,
8 1,*,8,h,
9 1,?,9,i,
```

CSV

6.2.5. Создание и выполнение графа

6.2.5.1. Граф без ветвлений

- **Цель:** Проверить выполнение простого линейного графа.
- **Предусловие:** Успешно выбран файл test.yaml.
- **Шаги выполнения:**
 1. Ввести схему: 1→3→2.
 2. Завершить ввод end.
 3. Наблюдать за выполнением операций.
- **Ожидаемый результат:** Все операции должны выполниться последовательно.
- **Фактический результат:** Граф выполняется корректно.
- **Результаты выполнения:**

```
1 sum_elements_of_column_0 >> 9
```

```
2 find_average_of_column_0 >> 1
3 sum_elements_of_column_2 >> 45
```

- **Изображение:**

```
~/programming/cc++/dag/test_app/data
  └── data
      ├── col_out_of_range.yaml
      └── Sun Dec 7 14:39:11 2025
          ├── test.csv
          ├── test.yaml
          ├── test_conf.yaml
          └── unknown_func.yaml

sum_elements_of_column_0 >> 9
find_average_of_column_0 >> 1
sum_elements_of_column_2 >> 45
```

Рис. 12.

6.2.5.2. Граф с ветвленими

- **Цель:** Проверить выполнение графа с параллельными ветвями.
- **Предусловие:** Успешно выбран файл test.yaml.
- **Шаги выполнения:**
 1. Ввести схему:

```
1 1→2→5
2 1→3→4
```
 2. Завершить ввод end.
 3. Наблюдать за выполнением операций.
- **Ожидаемый результат:** Операции должны выполняться согласно графу.
- **Фактический результат:** Граф выполняется корректно.
- **Изображение корректного ввода:**

```

> ./SimpleDAG.out
SIMPLEDAG

===== Доступные файлы =====
[1] data/test_conf.yaml
[2] data/test.yaml
[3] data/unknown_func.yaml
[4] data/col_out_of_range.yaml
[0] Выход
_____
Выберите файл (0-4): 2
✓ Выбран файл: data/test.yaml

===== Доступные операции =====
[1] sum_elements_of_column_0
[2] sum_elements_of_column_2
[3] find_average_of_column_0
[4] find_average_of_column_2
[5] concat
_____

ВВОД СХЕМЫ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ

ФОРМАТ: индекс1→индекс2→индекс3
ПРИМЕР: 1→2→3 или 1→3→5→2

ИНСТРУКЦИЯ:
1. Используйте номера операций из списка выше
2. Разделяйте номера стрелками '→'
3. Каждая строка – отдельная цепочка выполнения
4. Операции выполняются в порядке слева направо
5. Для завершения ввода введите 'end'

ВИЗУАЛИЗАЦИЯ ГРАФА:

Ввод: 1→2→4
      1→3→5
      2→4

Строит граф:

    graph LR
        1[1] --> 2[2]
        2[2] --> 4[4]
        1[1] --> 3[3]
        3[3] --> 5[5]
        5[5] --> 4[4]

ПОРЯДОК ВЫПОЛНЕНИЯ:
1 → 2 → 4
1 → 3 → 5
2 → 4 (уже выполнена в первой цепочке)

⚠ ВАЖНО:
• Нумерация операций начинается с 1
• Убедитесь, что все номера существуют в списке операций
• Граф будет построен автоматически на основе связей
• Повторяющиеся связи игнорируются

Ввод
1→2→5
1→3→4
end
✓ Схема графа корректна!

```

Рис. 13.

- Результаты выполнения:

```

1 sum_elements_of_column_0 >> 9
2 sum_elements_of_column_2 >> 45
3 concat >> !@#$%^&*?
4 find_average_of_column_0 >> 1
5 find_average_of_column_2 >> 5

```

- Изображение результатов:

```

~/programming/cc++/dag/test_app/data
  └── data
    ├── col_out_of_range.yaml
    ├── Sun Dec  7 14:41:18 2025
    └── SimpleDAG.out

test.csv
test.yaml
test_conf.yaml
unknown_func.yaml

sum_elements_of_column_0 >> 9
sum_elements_of_column_2 >> 45
concat >> !@#$%^&*
find_average_of_column_0 >> 1
find_average_of_column_2 >> 5

```

Рис. 14.

6.3. Автоматическое тестирование

- Инструменты:

- Компилятор: GCC 15.1.1 / Clang 18.1.6
- Стандарт языка: C++17
- Фреймворк для написания тестов: Google Test

6.3.1. Среда разработки

```

> ./runTests.sh
Running main() from /usr/src/debug/gtest/googletest-1.17.0/googletest/src/gtest_main.cc
[=====] Running 6 tests from 1 test suite.
[=====] Global test environment set-up.
[=====] 6 tests from OPERATIONS
[RUN    ] OPERATIONS.Sum
[OK     ] OPERATIONS.Sum (0 ms)
[RUN    ] OPERATIONS.Average
[OK     ] OPERATIONS.Average (0 ms)
[RUN    ] OPERATIONS.Concatinate
[OK     ] OPERATIONS.Concatinate (0 ms)
[RUN    ] OPERATIONS.SumFromMap
[OK     ] OPERATIONS.SumFromMap (0 ms)
[RUN    ] OPERATIONS.AverageFromMap
[OK     ] OPERATIONS.AverageFromMap (0 ms)
[RUN    ] OPERATIONS.ConcatinateFromMap
[OK     ] OPERATIONS.ConcatinateFromMap (0 ms)
[=====] 6 tests from OPERATIONS (0 ms total)

[=====] Global test environment tear-down
[=====] 6 tests from 1 test suite ran. (0 ms total)
[PASSED ] 6 tests.

```

Рис. 15.

```
Running main() from /usr/src/debug/gtest/gtest-1.17.0/gtest/src/gtest_main.cc
[=====] Running 18 tests from 1 test suite.
[=====] Global test environment set-up.
[=====] 18 tests from GRAPH
[RUN    ] GRAPH.CreateNode
[OK     ] GRAPH.CreateNode (0 ms)
[RUN    ] GRAPH.AddNode
[OK     ] GRAPH.AddNode (0 ms)
[RUN    ] GRAPH.ClearGraphWithoutConnections
[OK     ] GRAPH.ClearGraphWithoutConnections (0 ms)
[RUN    ] GRAPH.ConnectNodes
[OK     ] GRAPH.ConnectNodes (0 ms)
[RUN    ] GRAPH.ConnectNodesById
[OK     ] GRAPH.ConnectNodesById (0 ms)
[RUN    ] GRAPH.GetAdjacentNodes
[OK     ] GRAPH.GetAdjacentNodes (0 ms)
[RUN    ] GRAPH.ClearGraphWithConnections
[OK     ] GRAPH.ClearGraphWithConnections (0 ms)
[RUN    ] GRAPH.AlreadyInGraph
[OK     ] GRAPH.AlreadyInGraph (0 ms)
[RUN    ] GRAPH.AlreadyInGraphById
[OK     ] GRAPH.AlreadyInGraphById (0 ms)
[RUN    ] GRAPH.GetById
[OK     ] GRAPH.GetById (0 ms)
[RUN    ] GRAPH.GraphIsEmpty
[OK     ] GRAPH.GraphIsEmpty (0 ms)
[RUN    ] GRAPH.DeleteNodeWithoutConnections
[OK     ] GRAPH.DeleteNodeWithoutConnections (0 ms)
[RUN    ] GRAPH.DeleteNodeWithConnections
[OK     ] GRAPH.DeleteNodeWithConnections (0 ms)
[RUN    ] GRAPH.DeepFirstSearch_VisitsAllNodes
[OK     ] GRAPH.DeepFirstSearch_VisitsAllNodes (0 ms)
[RUN    ] GRAPH.DeepFirstSearch_StartsFromRoot
[OK     ] GRAPH.DeepFirstSearch_StartsFromRoot (0 ms)
[RUN    ] GRAPH.DeepFirstSearch_EmptyGraph
[OK     ] GRAPH.DeepFirstSearch_EmptyGraph (0 ms)
[RUN    ] GRAPH.DeepFirstSearch_SingleNode
[OK     ] GRAPH.DeepFirstSearch_SingleNode (0 ms)
[RUN    ] GRAPH.DeepFirstSearch_ComplexGraph
[OK     ] GRAPH.DeepFirstSearch_ComplexGraph (0 ms)
[=====] 18 tests from GRAPH (0 ms total)

[=====] Global test environment tear-down
[=====] 18 tests from 1 test suite ran. (0 ms total)
[PASSED ] 18 tests.
```

Рис. 16.

```
Running main() from /usr/src/debug/gtest/googletest-1.17.0/googletest/src/gtest_main.cc
[=====] Running 13 tests from 1 test suite.
[=====] Global test environment set-up.
[=====] 13 tests from PARSER
[ RUN   ] PARSER.GetScheme
[ OK    ] PARSER.GetScheme (0 ms)
[ RUN   ] PARSER.Split
[ OK    ] PARSER.Split (0 ms)
[ RUN   ] PARSER.CreateGraphFromScheme
[ OK    ] PARSER.CreateGraphFromScheme (0 ms)
[ RUN   ] PARSER.CreateGraphFromSchemeWithIndexes
[ OK    ] PARSER.CreateGraphFromSchemeWithIndexes (0 ms)
[ RUN   ] PARSER.GetTypeOfColumn
[ OK    ] PARSER.GetTypeOfColumn (0 ms)
[ RUN   ] PARSER.ReadNumericColumn
[ OK    ] PARSER.ReadNumericColumn (0 ms)
[ RUN   ] PARSER.ReadStringColumn
[ OK    ] PARSER.ReadStringColumn (0 ms)
[ RUN   ] PARSER.GetIds
[ OK    ] PARSER.GetIds (0 ms)
[ RUN   ] PARSER.GetColumnById
[ OK    ] PARSER.GetColumnById (0 ms)
[ RUN   ] PARSER.GetOpTypeById
[ OK    ] PARSER.GetOpTypeById (0 ms)
[ RUN   ] PARSER.CheckFunctions
[ OK    ] PARSER.CheckFunctions (0 ms)
[ RUN   ] PARSER.CheckScheme
[ OK    ] PARSER.CheckScheme (0 ms)
[ RUN   ] PARSER.GetCSV
[ OK    ] PARSER.GetCSV (0 ms)
[=====] 13 tests from PARSER (1 ms total)

[=====] Global test environment tear-down
[=====] 13 tests from 1 test suite ran. (1 ms total)
[ PASSED ] 13 tests.
```

Рис. 17.

```
Running main() from /usr/src/debug/gtest/googletest-1.17.0/googletest/src/gtest_main.cc
[=====] Running 8 tests from 1 test suite.
[=====] Global test environment set-up.
[=====] 8 tests from UTILS
[ RUN   ] UTILS.GetSourceFiles
[ OK    ] UTILS.GetSourceFiles (0 ms)
[ RUN   ] UTILS.WriteNumericResult
[ OK    ] UTILS.WriteNumericResult (0 ms)
[ RUN   ] UTILS.WriteStringResult
[ OK    ] UTILS.WriteStringResult (0 ms)
[ RUN   ] UTILS.Warning
[ OK    ] UTILS.Warning (0 ms)
[ RUN   ] UTILS.ProcedureNumeric
[ OK    ] UTILS.ProcedureNumeric (0 ms)
[ RUN   ] UTILS.ProcedureString
[ OK    ] UTILS.ProcedureString (0 ms)
[ RUN   ] UTILS.ProcedureAverage
[ OK    ] UTILS.ProcedureAverage (0 ms)
[ RUN   ] UTILS.ProcedureUnknownType
[ OK    ] UTILS.ProcedureUnknownType (0 ms)
[=====] 8 tests from UTILS (1 ms total)

[=====] Global test environment tear-down
[=====] 8 tests from 1 test suite ran. (1 ms total)
[ PASSED ] 8 tests.
```

Рис. 18.

6.3.2. GitHub Actions (CI)

- **Цель:** Обеспечить непрерывную интеграцию и проверку кода при обновлении веток: dev и main.

The screenshot shows a GitHub Actions test run titled "Run Tests on ubuntu-latest". The status bar indicates it succeeded 11 hours ago in 45s. The terminal output shows the execution of a script named "runTests.sh". The script runs Google Test cases for two main components: "OPERATIONS" and "GRAPH".

```
Run Tests on ubuntu-latest
succeeded 11 hours ago in 45s

Run tests

1 ► Run chmod +x runTests.sh
5 Running main() from ./googletest/src/gtest_main.cc
6 [=====] Running 6 tests from 1 test suite.
7 [-----] Global test environment set-up.
8 [-----] 6 tests from OPERATIONS
9 [ RUN      ] OPERATIONS.Sum
10 [ OK ] OPERATIONS.Sum (0 ms)
11 [ RUN      ] OPERATIONS.Average
12 [ OK ] OPERATIONS.Average (0 ms)
13 [ RUN      ] OPERATIONS.Concatenate
14 [ OK ] OPERATIONS.Concatenate (0 ms)
15 [ RUN      ] OPERATIONS.SumFromMap
16 [ OK ] OPERATIONS.SumFromMap (0 ms)
17 [ RUN      ] OPERATIONS.AverageFromMap
18 [ OK ] OPERATIONS.AverageFromMap (0 ms)
19 [ RUN      ] OPERATIONS.ConcatenateFromMap
20 [ OK ] OPERATIONS.ConcatenateFromMap (0 ms)
21 [-----] 6 tests from OPERATIONS (0 ms total)
22
23 [-----] Global test environment tear-down
24 [=====] 6 tests from 1 test suite ran. (0 ms total)
25 [ PASSED ] 6 tests.
26 Running main() from ./googletest/src/gtest_main.cc
27 [=====] Running 18 tests from 1 test suite.
28 [-----] Global test environment set-up.
29 [-----] 18 tests from GRAPH
30 [ RUN      ] GRAPH.CreateNode
31 [ OK ] GRAPH.CreateNode (0 ms)
32 [ RUN      ] GRAPH.AddNode
33 [ OK ] GRAPH.AddNode (0 ms)
34 [ RUN      ] GRAPH.ClearGraphWithoutConnections
35 [ OK ] GRAPH.ClearGraphWithoutConnections (0 ms)
36 [ RUN      ] GRAPH.ConnectNodes
37 [ OK ] GRAPH.ConnectNodes (0 ms)
38 [ RUN      ] GRAPH.ConnectNodesById
39 [ OK ] GRAPH.ConnectNodesById (0 ms)
40 [ RUN      ] GRAPH.GetAdjacentNodes
41 [ OK ] GRAPH.GetAdjacentNodes (0 ms)
42 [ RUN      ] GRAPH.ClearGraphWithConnections
43 [ OK ] GRAPH.ClearGraphWithConnections (0 ms)
44 [ RUN      ] GRAPH.AlreadyInGraph
45 [ OK ] GRAPH.AlreadyInGraph (0 ms)
46 [ RUN      ] GRAPH.AlreadyInGraphById
47 [ OK ] GRAPH.AlreadyInGraphById (0 ms)
48 [ RUN      ] GRAPH.GetById
49 [ OK ] GRAPH.GetById (0 ms)
50 [ RUN      ] GRAPH.GraphIsEmpty
51 [ OK ] GRAPH.GraphIsEmpty (0 ms)
52 [ RUN      ] GRAPH.DeleteNodeWithoutConnections
53 [ OK ] GRAPH.DeleteNodeWithoutConnections (0 ms)
54 [ RUN      ] GRAPH.DeleteNodeWithConnections
55 [ OK ] GRAPH.DeleteNodeWithConnections (0 ms)
56 [ RUN      ] GRAPH.DeepFirstSearch_VisitsAllNodes
57 [ OK ] GRAPH.DeepFirstSearch_VisitsAllNodes (0 ms)
58 [ RUN      ] GRAPH.DeepFirstSearch_StartsFromRoot
59 [ OK ] GRAPH.DeepFirstSearch_StartsFromRoot (0 ms)
60 [ RUN      ] GRAPH.DeepFirstSearch_EmptyGraph
61 [ OK ] GRAPH.DeepFirstSearch_EmptyGraph (0 ms)
62 [ RUN      ] GRAPH.DeepFirstSearch_SingleNode
63 [ OK ] GRAPH.DeepFirstSearch_SingleNode (0 ms)
64 [ RUN      ] GRAPH.DeepFirstSearch_ComplexGraph
65 [ OK ] GRAPH.DeepFirstSearch_ComplexGraph (0 ms)
66 [-----] 18 tests from GRAPH (0 ms total)
67
68 [-----] Global test environment tear-down
69 [=====] 18 tests from 1 test suite ran. (0 ms total)
70 [ PASSED ] 18 tests.
```

Рис. 19.

Run Tests on ubuntu-latest
succeeded 11 hours ago in 45s

Run tests

```

68 [-----] Global test environment tear-down
69 [=====] 18 tests from 1 test suite ran. (0 ms total)
70 [ PASSED ] 18 tests.
71 Running main() from ./googletest/src/gtest_main.cc
72 [=====] Running 13 tests from 1 test suite.
73 [-----] Global test environment set-up.
74 [-----] 13 tests from PARSER
75 [ RUN ] PARSER.GetScheme
76 [ OK ] PARSER.GetScheme (0 ms)
77 [ RUN ] PARSER.Split
78 [ OK ] PARSER.Split (0 ms)
79 [ RUN ] PARSER.CreateGraphFromScheme
80 [ OK ] PARSER.CreateGraphFromScheme (0 ms)
81 [ RUN ] PARSER.CreateGraphFromSchemeWithIndexes
82 [ OK ] PARSER.CreateGraphFromSchemeWithIndexes (0 ms)
83 [ RUN ] PARSER.GetTypeOfColumn
84 [ OK ] PARSER.GetTypeOfColumn (21 ms)
85 [ RUN ] PARSER.ReadNumericColumn
86 [ OK ] PARSER.ReadNumericColumn (0 ms)
87 [ RUN ] PARSER.ReadStringColumn
88 [ OK ] PARSER.ReadStringColumn (0 ms)
89 [ RUN ] PARSER.GetIds
90 [ OK ] PARSER.GetIds (0 ms)
91 [ RUN ] PARSER.GetColumnById
92 [ OK ] PARSER.GetColumnById (0 ms)
93 [ RUN ] PARSER.GetOpTypeById
94 [ OK ] PARSER.GetOpTypeById (0 ms)
95 [ RUN ] PARSER.CheckFunctions
96 [ OK ] PARSER.CheckFunctions (0 ms)
97 [ RUN ] PARSER.CheckScheme
98 [ OK ] PARSER.CheckScheme (0 ms)
99 [ RUN ] PARSER.GetCSV
100 [ OK ] PARSER.GetCSV (0 ms)
101 [-----] 13 tests from PARSER (23 ms total)
102
103 [-----] Global test environment tear-down
104 [=====] 13 tests from 1 test suite ran. (23 ms total)
105 [ PASSED ] 13 tests.
106 Running main() from ./googletest/src/gtest_main.cc
107 [=====] Running 8 tests from 1 test suite.
108 [-----] Global test environment set-up.
109 [-----] 8 tests from UTILS
110 [ RUN ] UTILS.GetSourceFiles
111 [ OK ] UTILS.GetSourceFiles (0 ms)
112 [ RUN ] UTILS.WriteNumericResult
113 [ OK ] UTILS.WriteNumericResult (0 ms)
114 [ RUN ] UTILS.WriteStringResult
115 [ OK ] UTILS.WriteStringResult (0 ms)
116 [ RUN ] UTILS.Warning
117 [ OK ] UTILS.Warning (0 ms)
118 [ RUN ] UTILS.ProcedureNumeric
119 [ OK ] UTILS.ProcedureNumeric (0 ms)
120 [ RUN ] UTILS.ProcedureString
121 [ OK ] UTILS.ProcedureString (0 ms)
122 [ RUN ] UTILS.ProcedureAverage
123 [ OK ] UTILS.ProcedureAverage (0 ms)
124 [ RUN ] UTILS.ProcedureUnknownType
125 [ OK ] UTILS.ProcedureUnknownType (0 ms)
126 [-----] 8 tests from UTILS (1 ms total)
127
128 [-----] Global test environment tear-down
129 [=====] 8 tests from 1 test suite ran. (1 ms total)
130 [ PASSED ] 8 tests.

```

> Upload test artifacts

> Post Checkout code

> Complete job

Рис. 20.

Run Tests on macos-latest
succeeded 11 hours ago in 35s

Run tests

```

1 ► Run chmod +x runTests.sh
5 Running main() from /private/tmp/googletest-20250808-4780-aa0gxy/googletest-1.17.0/googletest/src/gtest_main.cc
6 [=====] Running 6 tests from 1 test suite.
7 [-----] Global test environment set-up.
8 [-----] 6 tests from OPERATIONS
9 [ RUN ] OPERATIONS.Sum
10 [ OK ] OPERATIONS.Sum (0 ms)
11 [ RUN ] OPERATIONS.Average
12 [ OK ] OPERATIONS.Average (0 ms)
13 [ RUN ] OPERATIONS.Concatenate
14 [ OK ] OPERATIONS.Concatenate (0 ms)
15 [ RUN ] OPERATIONS.SumFromMap
16 [ OK ] OPERATIONS.SumFromMap (0 ms)
17 [ RUN ] OPERATIONS.AverageFromMap
18 [ OK ] OPERATIONS.AverageFromMap (0 ms)
19 [ RUN ] OPERATIONS.ConcatenateFromMap
20 [ OK ] OPERATIONS.ConcatenateFromMap (0 ms)
21 [-----] 6 tests from OPERATIONS (0 ms total)
22
23 [-----] Global test environment tear-down
24 [=====] 6 tests from 1 test suite ran. (0 ms total)
25 [ PASSED ] 6 tests.
26 Running main() from /private/tmp/googletest-20250808-4780-aa0gxy/googletest-1.17.0/googletest/src/gtest_main.cc
27 [=====] Running 18 tests from 1 test suite.
28 [-----] Global test environment set-up.
29 [-----] 18 tests from GRAPH
30 [ RUN ] GRAPH.CreateNode
31 [ OK ] GRAPH.CreateNode (0 ms)
32 [ RUN ] GRAPH.AddNode
33 [ OK ] GRAPH.AddNode (0 ms)
34 [ RUN ] GRAPH.ClearGraphWithoutConnections
35 [ OK ] GRAPH.ClearGraphWithoutConnections (0 ms)
36 [ RUN ] GRAPH.ConnectNodes
37 [ OK ] GRAPH.ConnectNodes (0 ms)
38 [ RUN ] GRAPH.ConnectNodesById
39 [ OK ] GRAPH.ConnectNodesById (0 ms)
40 [ RUN ] GRAPH.GetAdjentNodes
41 [ OK ] GRAPH.GetAdjentNodes (0 ms)
42 [ RUN ] GRAPH.ClearGraphWithConnections
43 [ OK ] GRAPH.ClearGraphWithConnections (0 ms)
44 [ RUN ] GRAPH.AlreadyInGraph
45 [ OK ] GRAPH.AlreadyInGraph (0 ms)
46 [ RUN ] GRAPH.AlreadyInGraphById
47 [ OK ] GRAPH.AlreadyInGraphById (0 ms)
48 [ RUN ] GRAPH.GetById
49 [ OK ] GRAPH.GetById (0 ms)
50 [ RUN ] GRAPH.GraphIsEmpty
51 [ OK ] GRAPH.GraphIsEmpty (0 ms)
52 [ RUN ] GRAPH.DeleteNodeWithoutConnections
53 [ OK ] GRAPH.DeleteNodeWithoutConnections (0 ms)
54 [ RUN ] GRAPH.DeleteNodeWithConnections
55 [ OK ] GRAPH.DeleteNodeWithConnections (0 ms)
56 [ RUN ] GRAPH.DeepFirstSearch_VisitsAllNodes
57 [ OK ] GRAPH.DeepFirstSearch_VisitsAllNodes (0 ms)
58 [ RUN ] GRAPH.DeepFirstSearch_StartsFromRoot
59 [ OK ] GRAPH.DeepFirstSearch_StartsFromRoot (0 ms)
60 [ RUN ] GRAPH.DeepFirstSearch_EmptyGraph
61 [ OK ] GRAPH.DeepFirstSearch_EmptyGraph (0 ms)
62 [ RUN ] GRAPH.DeepFirstSearch_SingleNode
63 [ OK ] GRAPH.DeepFirstSearch_SingleNode (0 ms)
64 [ RUN ] GRAPH.DeepFirstSearch_ComplexGraph
65 [ OK ] GRAPH.DeepFirstSearch_ComplexGraph (0 ms)
66 [-----] 18 tests from GRAPH (0 ms total)
67
68 [-----] Global test environment tear-down
69 [=====] 18 tests from 1 test suite ran. (0 ms total)
70 [ PASSED ] 18 tests.

```

Рис. 21.

Run Tests on macos-latest
succeeded 11 hours ago in 35s

Run tests

```

71  Running main() from /private/tmp/googletest-20250808-4780-aa0gxy/googletest-1.17.0/googletest/src/gtest_main.cc
72  [=====] Running 13 tests from 1 test suite.
73  [-----] Global test environment set-up.
74  [-----] 13 tests from PARSER
75  [ RUN   ] PARSER.GetScheme
76  [ OK    ] PARSER.GetScheme (0 ms)
77  [ RUN   ] PARSER.Split
78  [ OK    ] PARSER.Split (0 ms)
79  [ RUN   ] PARSER.CreateGraphFromScheme
80  [ OK    ] PARSER.CreateGraphFromScheme (0 ms)
81  [ RUN   ] PARSER.CreateGraphFromSchemeWithIndexes
82  [ OK    ] PARSER.CreateGraphFromSchemeWithIndexes (0 ms)
83  [ RUN   ] PARSER.GetTypeOfColumn
84  [ OK    ] PARSER.GetTypeOfColumn (0 ms)
85  [ RUN   ] PARSER.ReadNumericColumn
86  [ OK    ] PARSER.ReadNumericColumn (0 ms)
87  [ RUN   ] PARSER.ReadStringColumn
88  [ OK    ] PARSER.ReadStringColumn (0 ms)
89  [ RUN   ] PARSER.GetIds
90  test_parser.cpp:96: Failure
91  Expected equality of these values:
92  ids
93  Which is: { "sum_elements_of_column_1", "sum_elements_of_column_2", "find_average_of_column_1", "find_average_of_column_2", "concat" }
94  config::getIds()
95  Which is: { "sum_elements_of_column_1", "find_average_of_column_2", "sum_elements_of_column_2", "find_average_of_column_1", "concat" }
96
97  [ FAILED ] PARSER.GetIds (0 ms)
98  [ RUN   ] PARSER.GetColumnById
99  [ OK    ] PARSER.GetColumnById (0 ms)
100 [ RUN   ] PARSER.GetOpTypeById
101 [ OK    ] PARSER.GetOpTypeById (0 ms)
102 [ RUN   ] PARSER.CheckFunctions
103 [ OK    ] PARSER.CheckFunctions (0 ms)
104 [ RUN   ] PARSER.CheckScheme
105 [ OK    ] PARSER.CheckScheme (0 ms)
106 [ RUN   ] PARSER.GetCSV
107 [ OK    ] PARSER.GetCSV (0 ms)
108 [-----] 13 tests from PARSER (3 ms total)
109
110 [-----] Global test environment tear-down
111 [=====] 13 tests from 1 test suite ran. (3 ms total)
112 [ PASSED ] 12 tests.
113 [ FAILED  ] 1 test, listed below:
114 [ FAILED ] PARSER.GetIds
115
116 1 FAILED TEST
117 Running main() from /private/tmp/googletest-20250808-4780-aa0gxy/googletest-1.17.0/googletest/src/gtest_main.cc
118 [=====] Running 8 tests from 1 test suite.
119 [-----] Global test environment set-up.
120 [-----] 8 tests from UTILS
121 [ RUN   ] UTILS.GetSourceFiles
122 [ OK    ] UTILS.GetSourceFiles (0 ms)
123 [ RUN   ] UTILS.WriteNumericResult
124 [ OK    ] UTILS.WriteNumericResult (0 ms)
125 [ RUN   ] UTILS.WriteStringResult
126 [ OK    ] UTILS.WriteStringResult (0 ms)
127 [ RUN   ] UTILS.Warning
128 [ OK    ] UTILS.Warning (0 ms)
129 [ RUN   ] UTILS.ProcedureNumeric
130 [ OK    ] UTILS.ProcedureNumeric (1 ms)
131 [ RUN   ] UTILS.ProcedureString
132 [ OK    ] UTILS.ProcedureString (0 ms)
133 [ RUN   ] UTILS.ProcedureAverage
134 [ OK    ] UTILS.ProcedureAverage (0 ms)
135 [ RUN   ] UTILS.ProcedureUnknownType
136 [ OK    ] UTILS.ProcedureUnknownType (0 ms)
137 [-----] 8 tests from UTILS (3 ms total)
138
139 [-----] Global test environment tear-down
140 [=====] 8 tests from 1 test suite ran. (3 ms total)
141 [ PASSED ] 8 tests.

```

Рис. 22.