МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

**Разработка программного обеспечения для верификации блок-схем алгоритмов в формате JSON**

КУРСОВАЯ РАБОТА

по дисциплине «Программная инженерия»

ЮУрГУ – 02.03.02.20241.308-0372.КР

|  |  |
| --- | --- |
| Нормоконтролер3,  доцент кафедры СП, к.ф.-м.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Алеева  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | Научный руководитель:  доцент кафедры СП, к.ф.-м.н.  \_\_\_\_\_\_\_\_\_\_В.Н. Алеева  Автор работы:  студент группы КЭ-302  \_\_\_\_\_\_\_\_\_\_Д.Д. Цветков  Работа защищена  с оценкой: \_\_\_\_\_\_\_\_\_\_\_  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

15.02.2024 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине «Программная инженерия»

студенту группы КЭ-302

Цветкову Данилу Дмитриевичу,

обучающемуся по направлению

02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы**

Разработка программного обеспечения для верификации блок-схем алгоритмов в формате JSON.

1. **Срок сдачи студентом законченной работы:** 31.05.2024 г.
2. **Исходные данные к работе2**
3. V. Aleeva и R. Aleev, «Investigation and Implementation of Parallelism Resources of Numerical Algorithms», ACM Transactions on Parallel Computing, т. 10, вып. 2, 2023, doi: 10.1145/3583755.
4. Побегайло А. П. C/C++ для студентов. – БХВ-Петербург, 2006.
5. Song K. SMT-Based Bounded Model Checking for Solidity Smart Contracts, с. 64-65.
6. **Перечень подлежащих разработке вопросов3**
7. Анализ предметной области.
8. Изучить использование блок-схем алгоритмов для генерации Q-детерминантов алгоритмов.
9. Провести анализ требований к программной системе.
10. Разработать программное обеспечение, проверяющее на корректность блок-схемы алгоритмов в формате JSON.
11. Протестировать программное обеспечение, проверяющее на корректность блок-схемы алгоритмов в формате JSON.
12. **Дата выдачи задания:** 15.02.2024 г.

**Научный руководитель,**

доцент кафедры СП, к.ф.-м.н. В.Н. Алеева

**Задание принял к исполнению** Д.Д. Цветков

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 5](#_Toc168858322)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc168858323)

[1.1. Описание предметной области 7](#_Toc168858324)

[1.2. Сравнительный анализ аналогов 8](#_Toc168858325)

[1.3. Анализ существующих решений для реализации проекта 9](#_Toc168858326)

[2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ 10](#_Toc168858327)

[2.1. Анализ требований 10](#_Toc168858328)

[2.2. Варианты использования системы 10](#_Toc168858329)

[2.3. Классы анализа системы 11](#_Toc168858330)

[3. РЕАЛИЗАЦИЯ ПРОГРАММНОЙ СИСТЕМЫ 13](#_Toc168858331)

[3.1. Реализация основного функционала программной системы 13](#_Toc168858332)

[3.2. Реализация интерфейса программы 19](#_Toc168858333)

[4. ТЕСТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ 20](#_Toc168858334)

[ЗАКЛЮЧЕНИЕ 21](#_Toc168858335)

[ЛИТЕРАТУРА 22](#_Toc168858336)

[ПРИЛОЖЕНИЯ 23](#_Toc168858337)

[Приложение А. Спецификация вариантов использования 23](#_Toc168858338)

[Приложение Б. Скриншоты интерфейса программного обеспечения 27](#_Toc168858339)

# ВВЕДЕНИЕ

**Актуальность**

На данный момент процессоры с многоядерной архитектурой широко используются в нашей жизни. Для их эффективного использования нужны программы, предполагающие распараллеливание задач. Решением данной проблемы является концепция Q-детерминанта, чтобы упростить его генерацию требуется программное обеспечение для верификации блок-схем алгоритмов в формате JSON.

**Постановка задачи**

Целью курсовой работы является разработка программного обеспечения для верификации блок-схем алгоритмов в формате JSON. Для достижения поставленной цели необходимо решить следующие задачи:

1. провести анализ предметной области;
2. изучить использование блок-схем алгоритмов для генерации Q-детерминантов алгоритмов;
3. провести анализ требований к программной системе;
4. разработать программное обеспечение, проверяющее на корректность блок-схемы алгоритмов в формате JSON;
5. протестировать программное обеспечение, проверяющее на корректность блок-схемы алгоритмов в формате JSON.

**Структура и содержание работы**

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 30 страниц, объем списка литературы – 10 источников.

В первой главе описываются правила построения блок-схем алгоритмов в формате JSON, а также проводится сравнительный анализ аналогов и анализ существующих решений для анализа проекта.

Вторая глава посвящена проектированию разрабатываемой системы на основе выявленных функциональных и нефункциональных требований. Также проводится описание диаграмм вариантов использования и классов анализа.

В третьей главе описана программная реализация основного функционала программной системы и интерфейса.

Четвертая глава содержит результаты проведенных тестов.

В приложениях содержится спецификация вариантов использования разрабатываемого программного обеспечения, которая описывает процесс взаимодействия пользователя с системой и скриншоты интерфейса программного обеспечения.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

Основная задача заключается в разработке программного обеспечения для верификации блок-схем алгоритмов, которые описаны в формате JSON. Блок-схемы должны быть составлены правильно (рисунок 1), так как они используются для генерации Q-детерминантов численных алгоритмов. Создание данной программы необходимо для облегчения процесса разработки параллельных программ на основе концепции Q-детерминанта.

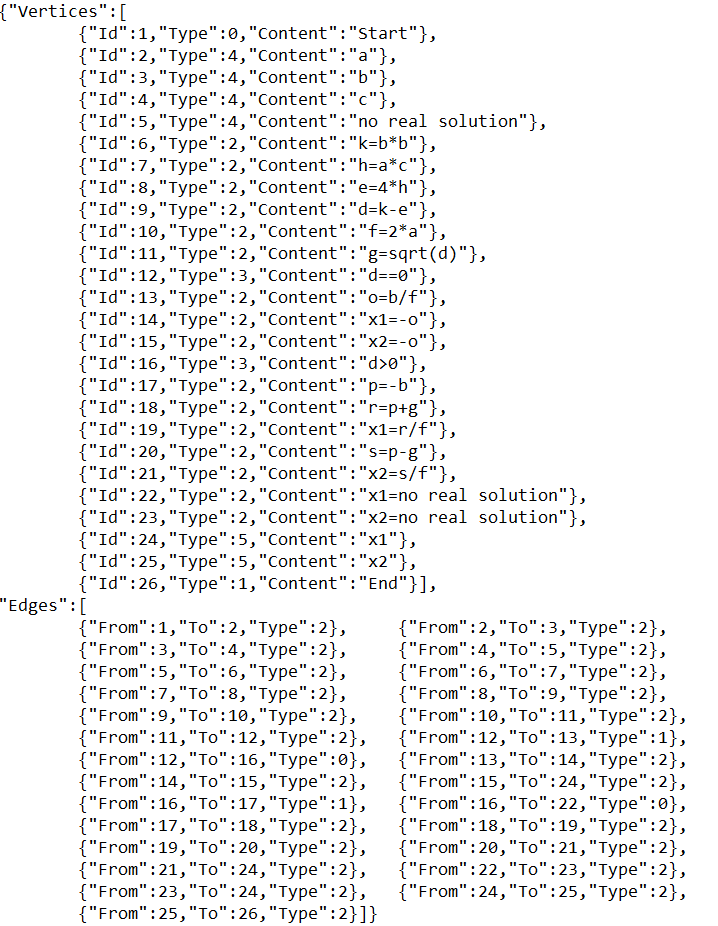


Рисунок 1 – Блок-схема в формате JSON алгоритма решения квадратного уравнения с унарным минусом

В соответствии с форматом JSON блоксхема представляет собой описание блоков «Vertices» и соединений «Edges». Блоки «Vertices» идентифицируются номером «Id», типом «Type» и текстовым содержимым «Content». Значения «Type» блоков «Vertices»: 0 ‒ блок «Start», 1 ‒ блок «End», 2 ‒ блок «Процесс»», 3 ‒ блок «Решение», 4 ‒ блок «Входные данные», 5 ‒ блок «Выходные данные». Соединения «Edges» определяются номерами начального блока «From» и конечного блока «To» и типом соединения «Type». Значения «Type» соединений «Edges» следующие: 0 ‒ переход по условию false, 1 ‒ переход по условию true, 2 ‒ обычное соединение.

Генерация Q-детерминантов является одной из подсистем Q-системы. Она используется для исследования ресурсов параллелизма численных алгоритмов, которые можно вычислять и сравнивать. Q-система вычисляет и сравнивает высоту и ширину численных алгоритмов, и, таким образом, позволяет количественно оценить ресурс параллелизма алгоритма.

В рамках текущей работы предлагается следующее решение поставленной задачи: идет посимвольная проверка блок-схемы для уточнения верного расположения различных знаков, далее программа, используя несколько циклов и условных операторов, проверяет по заданным правилам блок-схему формата JSON.

1.2. Сравнительный анализ аналогов

Сегодня существует большое количество инструментов для параллельной обработки информации, например, такие технологии как OpenMP или MPI. В настоящее время уже существуют системы, которые создают Q-эффективные программы. Для них требуется программное обеспечение для верификации блок-схем алгоритмов в формате JSON, которое еще не реализовано, но существуют проекты, которые осуществляют проверку блок-схем в формате JSON на правильную конструкцию (рисунок 2).

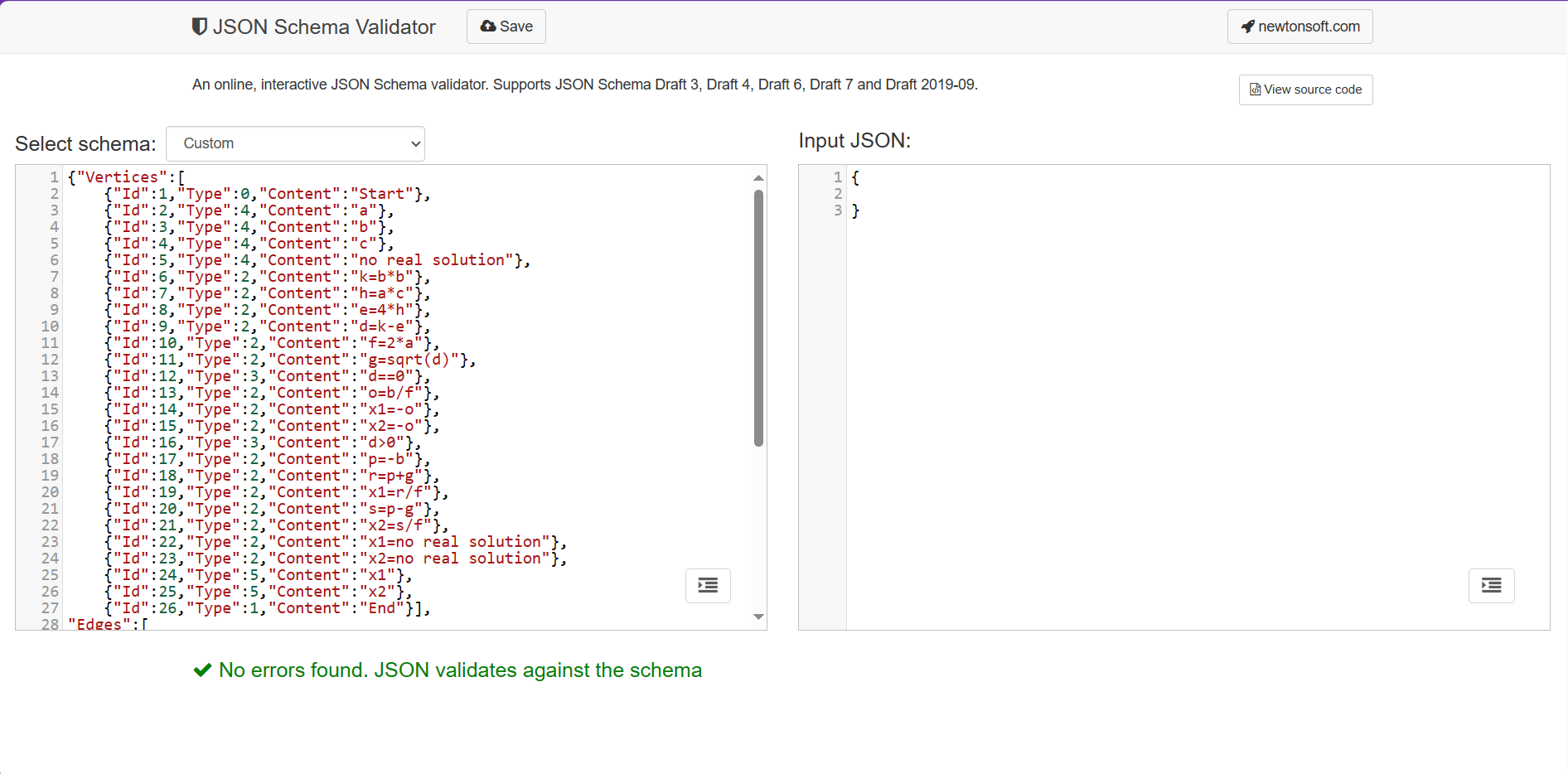
****

Рисунок 2 – Пример работы проекта, осуществляющего проверку блок-схемы JSON

1.3. Анализ существующих решений для реализации проекта

Для того чтобы произвести посимвольную проверку JSON файла можно использовать JSON-парсер из библиотеки valijson.

Для работы с файлом JSON следует использовать библиотеку nlohmann::json, так как здесь можно удобно использовать объект JSON и обращение по индексу.

Другим способом посимвольной проверки JSON файла является конвертация файла JSON в формат .txt и считывание всех данных в одну строку. Далее, по определенным правилам построения блок-схемы, проверять нахождение нужного символа в правильном месте.

**Выводы по первой главе**

В результате анализа источников, был выбран набор инструментальных средств для реализации поставленной задачи. Было принято решение использовать конвертацию файла в формат .txt, а также библиотеку nlohmann/json. На основе изученного материала, можно сформулировать функциональные и нефункциональные требования.

# 2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ

2.1. Анализ требований

Для разрабатываемой программы выявлены следующие функциональные требования:

1) верификатор блок-схем должен осуществлять проверку на нарушение структуры блок-схемы;

2) верификатор блок-схем должен запрашивать ввод названия файла;

3) верификатор блок-схем должен проверять логику блок-схемы;

4) верификатор блок-схем должен осуществлять проверку терминальных блоков;

5) верификатор блок-схем должен осуществлять проверку блоков процессов;

6) верификатор блок-схем должен осуществлять проверку блоков решений;

7) верификатор блок-схем должен осуществлять проверку блоков данных.

Также были выявлены следующие нефункциональные требования:

1) верификатор блок-схем должен быть написан на языке программирования С++;

2) верификатор блок-схем должен предоставлять консольный интерфейс.

2.2. Варианты использования системы

Чтобы получить представление о функциональных возможностях разрабатываемой программы, с помощью языка графического описания для объектного моделирования UML была построена модель взаимодействия актера «Пользователь» с верификатором блок-схем. На основе определенных функциональных требований была построена диаграмма вариантов использования (рисунок 3). Спецификация вариантов использования представлена в приложении в таблицах 1–7 приложения А.

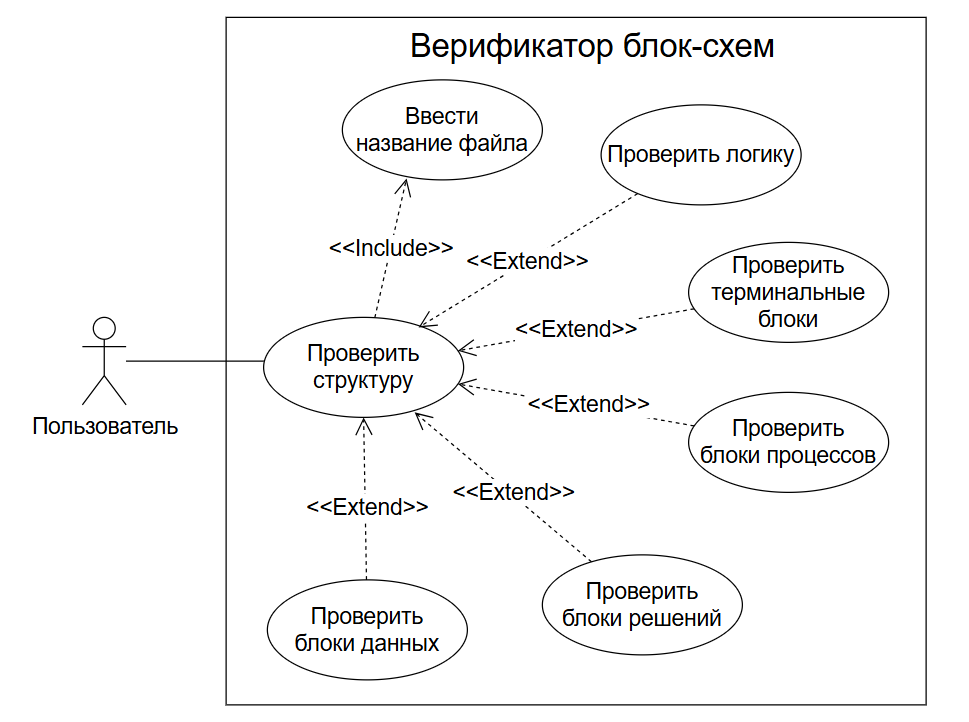


Рисунок 3 – Диаграмма вариантов использования верификатора блок-схем

2.3. Классы анализа системы

Диаграмма представлена на рисунке 4 и состоит из следующих классов: User\_Interface, Converter, Checker.

Класс User\_Interface можно обозначить стереотипом «boundary», так как он является посредником между системой и пользователем. Было создано два метода и четыре атрибута. Атрибуты name, filename\_json, filename\_txt и filename\_check имеют тип string.

Enter\_name(), который ничего не принимает в качестве параметра и имеет возвращаемый тип string.

Show(objJson: nlohmann::json) является основным методом, с помощью которого идет взаимодействие с другими классами.

Также было создано два класса, которые являются сущностями – Converter и Checker.

В классе Converter имеется два метода To\_JSON(filename::string) и To\_Txt(filename::string), которые принимают имя файла и конвертируют его в определенный тип.

В классе Checker методы представляют собой проверку блок-схемы. Также здесь было создано несколько атрибутов, которые используются в ходе выполнения работы программы.

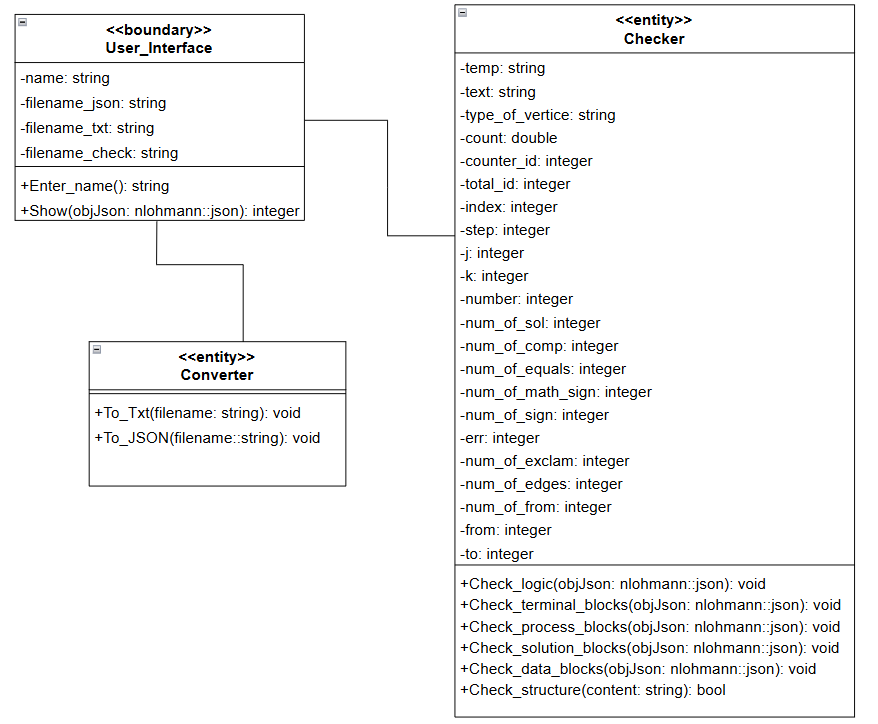


Рисунок 4 – Диаграмма классов анализа верификатора блок-схем

**Выводы по второй главе.**

В ходе проектирования разрабатываемого программного обеспечения были выявлены функциональные и нефункциональные требования, а также построены диаграммы вариантов использования и классов анализа.

# 3. РЕАЛИЗАЦИЯ ПРОГРАММНОЙ СИСТЕМЫ

3.1. Реализация основного функционала программной системы

Для разработки программного обеспечения был выбран язык C++, так как в нем присутствует необходимая библиотека. Реализация программы осуществлялась при помощи среды разработки Microsoft Visual Studio, потому что в ней возможна одновременная работа с файлами форматов .json и .cpp. Все исходные коды размещены на GitHub по ссылке URL: https://github.com/dwe11er/Course\_Work.

Разрабатываемая программная система использует библиотеку nlohmann для взаимодействия с объектами JSON.

Все основные методы для верификации блок-схем алгоритмов были реализованы в классе Checker.

В методе для проверки терминальных блоков (листинг 1) используется условный оператор и цикл. В условном операторе проверяется содержимое первой вершины, если оно не равно «Start», то выводится сообщение, что допущена ошибка в терминальном блоке Start. В цикле содержится несколько условных операторов. В первом, если содержимое вершины равняется «Start», а тип не равен нулю, то выводится сообщение, что допущена ошибка в значении типа. Во втором происходит то же самое, но для терминального блока «End». В третьем и четвертом условных операторах проверяется обратная ситуация, когда уже содержимое не соответствует типу вершины. При нахождении ошибки в любом из условных операторов увеличивается значение переменной err на единицу. В последнем условном операторе проверяется наличие ошибок, если их нет, то выводится сообщение об этом.

Листинг 1 – Метод для проверки терминальных блоков

void Check\_terminal\_blocks(nlohmann::json objJson)

{

if (objJson["Vertices"][0]["Content"] != "Start")

{

std::cout << "Ошибка в терминальном блоке Start\n" << std::endl;

err++;

}

for (int i = 0; i < size(objJson["Vertices"]); i++) {

if (objJson["Vertices"][i]["Content"] == "Start")

{

if (objJson["Vertices"][i]["Type"] != 0)

{

std::cout << "Ошибка в значении Type блока 'Start'" << "\n" << std::endl;

err++;

}

}

if (objJson["Vertices"][i]["Content"] == "End")

{

if (objJson["Vertices"][i]["Type"] != 1)

{

std::cout << "Ошибка в значении Type блока 'End'" << "\n" << std::endl;

err++;

}

}

if (objJson["Vertices"][i]["Type"] == 0)

{

if (objJson["Vertices"][i]["Content"] != "Start")

{

std::cout << "Ошибка, получен 0 в значении Type, Id " << objJson["Vertices"][i]["Id"] << "\n" << std::endl;

err++;

}

}

if (objJson["Vertices"][i]["Type"] == 1)

{

if (objJson["Vertices"][i]["Content"] != "End")

{

std::cout << "Ошибка, получена 1 в значении Type, Id " << objJson["Vertices"][i]["Id"] << "\n" << std::endl;

err++;

}

}

}

if (err == 0) {

std::cout << "Ошибок в терминальных блоках нет" << std::endl;

}

};

В методе для проверки блоков процессов (листинг 2) используется цикл. С помощью этого цикла программа проходит по всем вершинам, если тип равен 2, то идет дальнейшая проверка, которая состоит вложенного цикла for, там проверяется содержимое вершины, если находится знак равно или матчематические знаки, такие как +, \*, -, /, sqrt, то увеличиваются их счетчики. Если счетчик математических знаков больше единицы, то выводится сообщение с этой ошибкой, тоже самое происходит и с счетчиком знаков равно. В последнем условном операторе проверяется наличие ошибок, если их нет, то выводится сообщение об этом.

Листинг 2 – Метод для проверки блоков процессов

void Check\_process\_blocks(nlohmann::json objJson)

{

for (int i = 0; i < size(objJson["Vertices"]); i++) {

if (objJson["Vertices"][i]["Type"] == 2) // проверка блока процесса

{

std::string str = objJson["Vertices"][i]["Content"];

for (int i = 0; i < str.size(); i++)

{

if (str[i] == \*"=")

{

num\_of\_equals += 1;

}

if (str[i] == \*"+" or str[i] == \*"\*" or str[i] == \*"-" or str[i] == \*"/" or (str[i] == \*"s" and str[i + 1] == \*"q" and str[i + 2] == \*"r" and str[i + 3] == \*"t"))

{

num\_of\_math\_sign += 1;

}

}

if (num\_of\_math\_sign > 1)

{

std::cout << "Ошибка в количестве математических действий, получено: " << num\_of\_math\_sign << std::endl;

std::cout << "Ошибка в Id: " << i + 1 << "\n" << std::endl;

err++;

}

if (num\_of\_equals > 1)

{

std::cout << "Ошибка в количестве знаков равно, получено: " << num\_of\_equals << std::endl;

std::cout << "Ошибка в Id: " << i + 1 << "\n" << std::endl;

err++;

}

num\_of\_math\_sign = 0; // обнуляем счетчики

num\_of\_equals = 0;

}

}

if (err == 0) {

std::cout << "Ошибок в блоках процессов нет" << std::endl;

}

}

В методе для проверки блоков решений используется два цикла. В первом программа проходит по всем соединениям, если встречает тип соединения равный 0, а тип вершины не равный 3, то выводит сообщение с ошибкой об этом. Во втором программа проходит по всем вершинам, если встречает тип равный 3, то записывает Id этой вершины и увеличивает счетчик решений на единицу, также записывает тип этого соединения. Во вложенном цикле программа проходится по всем соединениям, если значение From совпадает с записанным Id и тип соединения равен нулю или единице, то увеличивает счетчик такого соединения. Если таких соединений не приходится по две штуки на один блок решения, то выводится сообщение с этой ошибкой. Кроме того, если сложение двух типов соединений одного блока решения не дает в сумме единицу, то выводится сообщение об этом. Далее идет цикл для проверки содержимого блока решения, если обнаруживается два и более знаков сравнения или два и более восклицательных знаков, или три знака равно, то выводится сообщение с соответствующей ошибкой. В последнем условном операторе проверяется наличие ошибок, если их нет, то выводится сообщение об этом.

В методе проверки блоков данных (листинг 3) используется цикл, который позволяет проходить по вершинам, если тип вершины равняется 4 или 5, то происходит проверка содержимого, в случае нахождения математических действий, описанных ранее, кроме знака минус, или знаков равно выводится соответствующая ошибка. Также не допускаются такие символы, как точка с запятой, двоеточие и одинарная кавычка. В последнем условном операторе проверяется наличие ошибок, если их нет, то выводится сообщение об этом.

Листинг 3 – Метод проверки блоков данных

void Check\_data\_blocks(nlohmann::json objJson)

{

for (int i = 0; i < size(objJson["Vertices"]); i++) {

if (objJson["Vertices"][i]["Type"] == 4 or objJson["Vertices"][i]["Type"] == 5) //проверка блока данных

{

std::string str = objJson["Vertices"][i]["Content"];

for (int i = 0; i < str.size(); i++)

{

if (str[i] == \*"=")

{

num\_of\_equals += 1;

}

if (str[i] == \*"+" or str[i] == \*"\*" or str[i] == \*"/" or (str[i] == \*"s" and str[i + 1] == \*"q" and str[i + 2] == \*"r" and str[i + 3] == \*"t"))

{

num\_of\_math\_sign += 1;

}

if (str[i] == \*";" or str[i] == \*"'" or str[i] == \*":")

{

num\_of\_sign += 1;

}

}

if (num\_of\_math\_sign > 0)

{

std::cout << "Ошибка, найдены математические действия в блоке данных" << std::endl;

std::cout << "Ошибка в Id: " << i + 1 << "\n" << std::endl;

err++;

}

if (num\_of\_equals > 0)

{

std::cout << "Ошибка, найдены знаки равно в блоке данных" << std::endl;

std::cout << "Ошибка в Id: " << i + 1 << "\n" << std::endl;

err++;

}

if (num\_of\_sign > 0)

{

std::cout << "Ошибка, найдены недопустимые знаки в блоке данных" << std::endl;

std::cout << "Ошибка в Id: " << i + 1 << "\n" << std::endl;

err++;

}

num\_of\_math\_sign = 0; // обнуляем счетчики

num\_of\_equals = 0;

num\_of\_sign = 0;

}

}

if (err == 0) {

std::cout << "Ошибок в блоках данных нет" << std::endl;

}

}

В методе проверки логики блок-схемы используются циклы и условные операторы из предыдущих методов, но также имеется цикл, который проверяет, что элемент «To» не может иметь значения начальной вершины, а элемент «From» значения последней.

Метод проверки структуры блок-схемы возвращает true или false. True, если ошибок в структуре нет, false в обратном случае. На вход поступает строка, которая получается путем считывания из текстового файла. Для проверки используется четыре цикла. Первый нужен для подсчета количества вершин, их число получается при нахождении в строке элемента «Id». Далее идет цикл, который проверяет структуру блока «Vertices». Все пробелы пропускаются, так как они не влияют на структуру. Каждому символу выдается индекс в порядке его встречи, если обнаруживается символ с неправильным индексом, то выводится соответствующее сообщение. После обработки одной вершины переменная, которая отвечает за разницу индексов, увеличивается на число символов, отвечающих за структуру. В конце блока «Vertices», когда встречается символ закрывающей квадратной скобки, то записывается номер символа и идет переход к следующему циклу. В нем подсчитывается количество соединений, путем нахождения числа элементов «From». Далее идет проверка структуры блока «Edges», здесь используется аналогичный способ.

В классе Converter реализовано два метода. Первый (листинг 4) конвертирует файл из формата .json в формат .txt, путем удаления прошлого окончания в наименовании файла и добавления нового.

Листинг 4 – Метод преобразования в формат .txt

void To\_Txt(std::string filename)

{

std::filesystem::path p(filename);

filename.erase(filename.size() - 5, 5); // удаляем .json

rename(p, std::string(filename) + ".txt");

}

Второй (листинг 5) метод конвертирует файл из формата .txt в формат .json тем же способом.

Листинг 5 – Метод преобразования в формат .json

void To\_JSON(std::string filename)

{

std::filesystem::path p(filename);

filename.erase(filename.size() - 4, 4); // удаляем .txt

rename(p, std::string(filename) + ".json");

}

3.2. Реализация интерфейса программы

Интерфейс программы является консольным. В нем реализовано меню выбора основных возможностей программного обеспечения. Скриншоты разработанного интерфейса программного обеспечения приведены на рисунках 1–10 приложения Б. Все методы связанные с интерфейсом программы реализованы в классе User\_Interface.

Метод Enter\_name() запрашивает ввод названия файла и возвращает его. В методе Show(nlohmann::json objJson) вызывается прошлый метод и идет проверка существования введенного файла, в случае отсутствия или ошибки при вводе, выводится сообщение с ошибкой. В ином случае, идет конвертация файла в формат .txt и считывание информации в одну строку, далее файл закрывается и конвертируется обратно в JSON. Если проверка структуры блок-схемы пройдена успешно, то выводится меню с вариантами выбора проверки блок-схемы.

**Выводы по третьей главе**

На основе спроектированной ранее модели с помощью выбранных методов и технологий было разработано программное обеспечение для верификации блок-схем алгоритмов в формате JSON. Необходимо провести функциональное тестирование системы для выявления возможных ошибок.

# 4. ТЕСТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ

Для тестирования программного обеспечения использовались следующие методы тестирования:

* функциональное тестирование;
* тестирование интерфейса пользователя.

Таблица 1 – Тестирование системы

| **№** | **Название**  **теста** | **Шаги** | **Ожидаемый**  **результат** | **Тест пройден?** |
| --- | --- | --- | --- | --- |
|  | Ввод названия файла | 1. Запустить программу. 2. Ввести название файла без расширения. | Появление меню и права выбора варианта. | Да |
|  | Проверка структуры блок-схемы | 1. Запустить программу. 2. Ввести название файла без расширения. | Появление меню и права выбора варианта. | Да |
|  | Проверка логики блок-схемы | 1. Запустить программу. 2. Ввести название файла без расширения. 3. Ввести цифру 1. | Появление сообщения «Ошибок нет». | Да |
|  | Проверка терминальных блоков | 1. Запустить программу. 2. Ввести название файла без расширения. 3. Ввести цифру 2. | Появление сообщения «Ошибок в терминальных блоках нет». | Да |
|  | Проверка блоков процессов | 1. Запустить программу. 2. Ввести название файла без расширения. 3. Ввести цифру 3. | Появление сообщения «Ошибок в блоках процессов нет». | Да |
|  | Проверка блоков решений | 1. Запустить программу. 2. Ввести название файла без расширения. 3. Ввести цифру 4. | Появление сообщения «Ошибок в блоках решений нет». | Да |
|  | Проверка блоков данных | 1. Запустить программу. 2. Ввести название файла без расширения. 3. Ввести цифру 5. | Появление сообщения «Ошибок в блоках данных нет». | Да |

**Выводы по четвертой главе**

В рамках данной главы было проведено функциональное тестирование программной системы. Также было выполнено тестирование интерфейса пользователя. С помощью данных методов тестирования было выявлено, что программное обеспечение выполняет свои основные функции.

# ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано программное обеспечение для верификации блок-схем алгоритмов в формате JSON. При этом были решены следующие задачи.

1. проведен анализ предметной области;
2. изучено использование блок-схем алгоритмов для генерации Q-детерминантов алгоритмов;
3. проведен анализ требований к программной системе;
4. разработано программное обеспечение, проверяющее на корректность блок-схемы алгоритмов в формате JSON;
5. протестировано программное обеспечение, проверяющее на корректность блок-схемы алгоритмов в формате JSON.

# ЛИТЕРАТУРА

1. Алеева В.Н. Анализ параллельных численных алгоритмов. Новосибирск, 1985. 23 с.
2. Алеева В.Н., Шатов М.Б. Применение концепции Q-детерминанта для эффективной реализации численных алгоритмов на примере метода сопряженных градиентов для решения систем линейных уравнений. // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2021. – Т. 10, № 3. – С. 56–71.
3. Официальный сайт проекта JSON Schema. URL: <https://json-schema.org/> (дата обращения 06.03.2024 г.).
4. Aleeva V., Bogatyreva E., Skleznev A., Sokolov M., Shuppa A. (2019) Software Q-system for the Research of the Resource of Numerical Algorithms Parallelism. In: Voevodin V., Sobolev S. (eds) Supercomputing. RuSCDays 2019. Communications in Computer and Information Science, vol 1129, pp. 641-652, 2019. Springer, Cham. DOI: 10.1007/978-3-030-36592-9\_52 [https://link.springer.com/chapter/10.1007%2F978-3-030-36592-9\_52].
5. JSON for Modern C++. [Электронный ресурс] URL: <https://json.nlohmann.me> (дата обращения 12.05.2024 г.).
6. Meyers S. Effective Modern C++ 42 Specific Ways to Improve Your Use of C++11 and C++14 // O’Reilly Media. 2014.
7. OMG. Unified modeling language // Object Manag. Gr. 2017.
8. Song K. SMT-Based Bounded Model Checking for Solidity Smart Contracts, с. 64-65.
9. Stroustrup B. the C++ Programming Language 4Th Edition // The C++ Programming. 2013.
10. V. Aleeva и R. Aleev, «Investigation and Implementation of Parallelism Resources of Numerical Algorithms», ACM Transactions on Parallel Computing, т. 10, вып. 2, 2023, doi: 10.1145/3583755.

# ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–7.

Таблица 1 – Спецификация ВИ «Проверить логику»

|  |
| --- |
| Прецедент: Проверить структуру |
| ID: 1 |
| Краткое описание:  Проверка структуры блок схемы на правильное расположение определенных символов и на правильное название основных элементов блок-схемы |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия: блок-схема добавлена в папку JSON и введено название файла |
| Основной поток:  1. Прецедент начинается, когда пользователь запускает программу и вводит название файла 2. Программа проверяет структуру блок-схемы |
| Постусловия: 1. Проверенная структура блок-схемы. |
| Альтернативные потоки: I. Входной файл пуст  1. Программа выводит сообщение об ошибке. |

Продолжение приложения А

Таблица 2 – Спецификация ВИ «Ввести название файла»

|  |
| --- |
| Прецедент: Ввести название файла |
| ID: 2 |
| Краткое описание:  Ввод названия файла. |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия:  Нет |
| Основной поток:  1. Прецедент начинается, когда пользователь запускает программу. 2. Пользователь вводит название файла. |
| Постусловия: 1. Введено название файла в программу. |
| Альтернативные потоки: I. Введено неверное название файла.  1. Приложение выводит сообщение об ошибке.  II. Файл отсутствует в папке JSON.  1. Приложение выводит сообщение об ошибке. |

Таблица 3 – Спецификация ВИ «Проверить логику»

|  |
| --- |
| Прецедент: Проверить логику |
| ID: 3 |
| Краткое описание:  Проверка основных блоков блок-схемы на верную логику и элементов «From» и «To» |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия: 1. Проверена структура блок-схемы |
| Основной поток:  1. Прецедент начинается, когда пользователь выбирает пункт в меню «Проверить логику». 2. Программа проверяет логику блок-схемы. |
| Постусловия: 1. Проверена логика блок-схемы |
| Альтернативные потоки: I. Неправильный ввод при выборе в меню.  1. Приложение выводит сообщение об ошибке. |

Продолжение приложения А

Таблица 4 – Спецификация ВИ «Проверить терминальные блоки»

|  |
| --- |
| Прецедент: Проверить терминальные блоки |
| ID: 4 |
| Краткое описание:  Проверка блоков «Start» и «End». |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия: 1. Проверена структура блок-схемы |
| Основной поток:  1. Прецедент начинается, когда пользователь выбирает пункт в меню «Проверить терминальные блоки». 2. Программа проверяет терминальные блоки. |
| Постусловия: 1. Проверены терминальные блоки. |
| Альтернативные потоки: I. Неправильный ввод при выборе в меню.  1. Приложение выводит сообщение об ошибке. |

Таблица 5 – Спецификация ВИ «Проверить блоки процессов»

|  |
| --- |
| Прецедент: Проверить блоки процессов |
| ID: 5 |
| Краткое описание:  Проверка блоков с типом 2. |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия: 1. Проверена структура блок-схемы |
| Основной поток:  1. Прецедент начинается, когда пользователь выбирает пункт в меню «Проверить блоки процессов». 2. Программа проверяет блоки процессов. |
| Постусловия: 1. Проверены блоки процессов. |
| Альтернативные потоки: I. Неправильный ввод при выборе в меню.  1. Приложение выводит сообщение об ошибке. |

Окончание приложения А

Таблица 6 – Спецификация ВИ «Проверить блоки процессов»

|  |
| --- |
| Прецедент: Проверить блоки процессов |
| ID: 6 |
| Краткое описание:  Проверка блоков с типом 3. |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия: 1. Проверена структура блок-схемы |
| Основной поток:  1. Прецедент начинается, когда пользователь выбирает пункт в меню «Проверить блоки решений». 2. Программа проверяет блоки решений. |
| Постусловия: 1. Проверены блоки решений. |
| Альтернативные потоки: I. Неправильный ввод при выборе в меню.  1. Приложение выводит сообщение об ошибке. |

Таблица 7 – Спецификация ВИ «Проверить блоки данных»

|  |
| --- |
| Прецедент: Проверить блоки данных |
| ID: 7 |
| Краткое описание:  Проверка блоков с типом 4 или 5. |
| Главные актеры:  Пользователь |
| Второстепенные актеры: Нет |
| Предусловия: 1. Проверена структура блок-схемы |
| Основной поток:  1. Прецедент начинается, когда пользователь выбирает пункт в меню «Проверить блоки данных». 2. Программа проверяет блоки данных. |
| Постусловия: 1. Проверены блоки данных. |
| Альтернативные потоки: I. Неправильный ввод при выборе в меню.  1. Приложение выводит сообщение об ошибке. |

# Приложение Б. Скриншоты интерфейса программного обеспечения

Скриншоты разработанного интерфейса программного обеспечения приведены на рисунках 1–10.

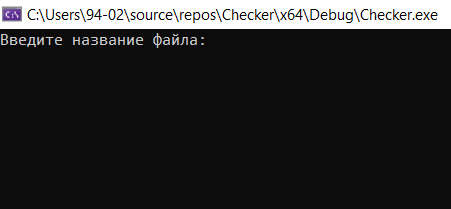


Рисунок 1 – Запуск программы

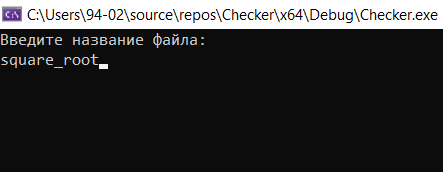


Рисунок 2 – Ввод названия файла

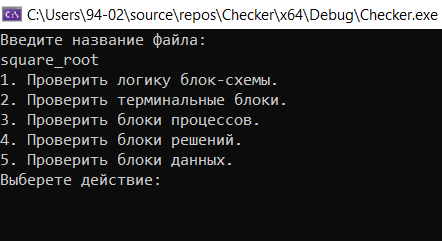


Рисунок 3 – Меню программы с выбором действия

Продолжение приложения Б

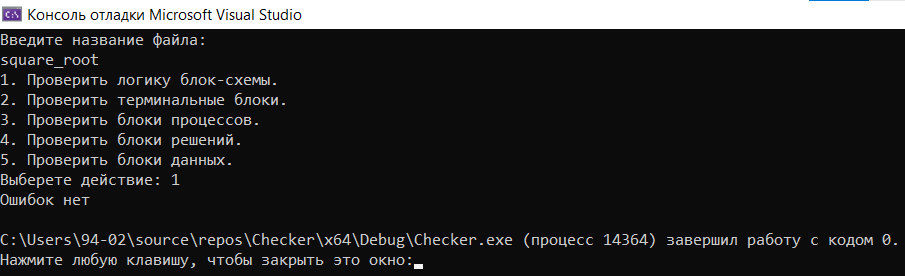


Рисунок 4 – Результат проверки логики блок-схемы

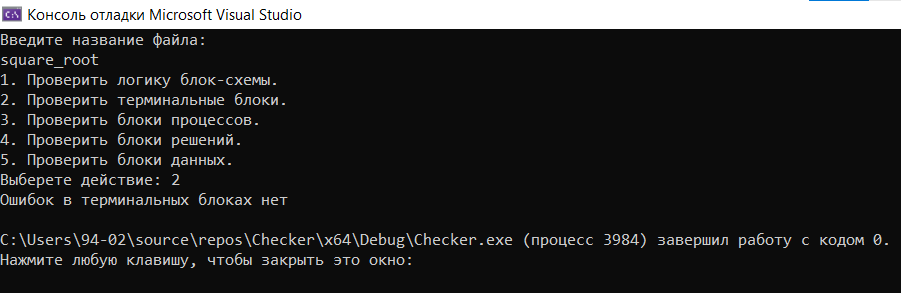


Рисунок 5 – Результат проверки терминальных блоков

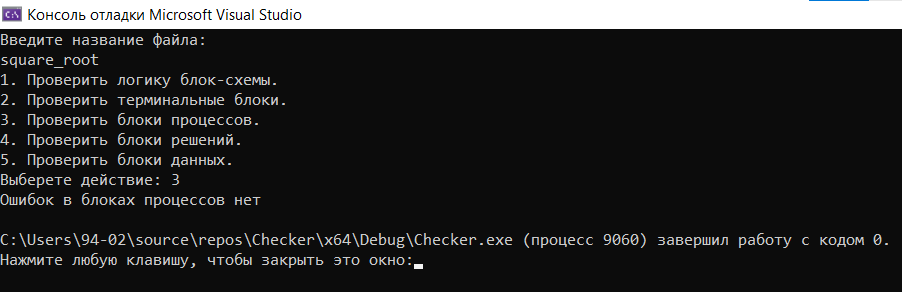


Рисунок 6 – Результат проверки блоков процессов

Продолжение приложения Б

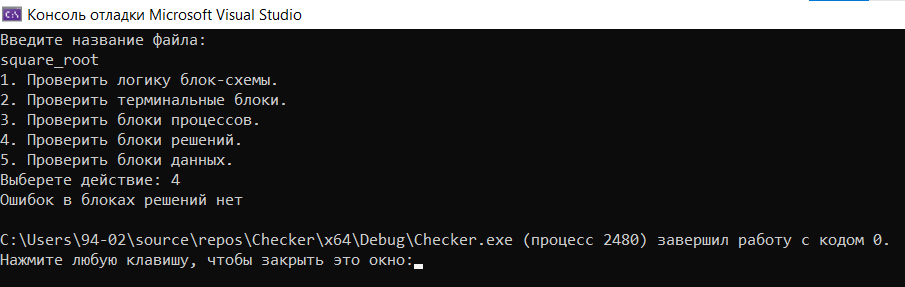


Рисунок 7 – Результат проверки блоков решений

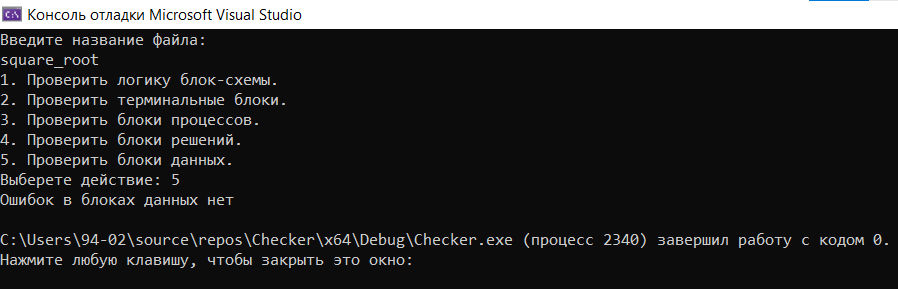


Рисунок 8 – Результат проверки блоков данных

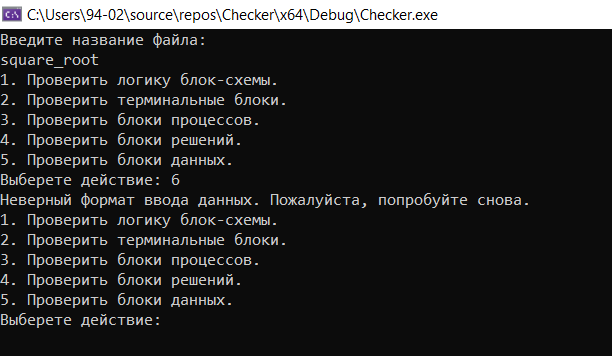


Рисунок 9 – Неправильный формат ввода данных

Окончание приложения Б

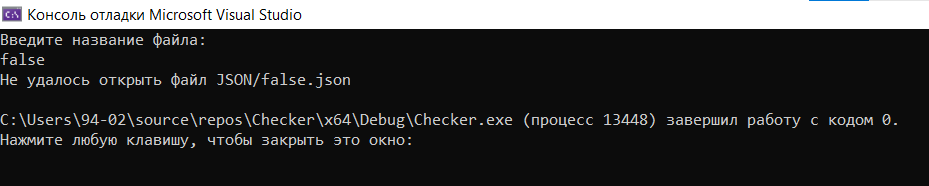


Рисунок 10 – Неправильный ввод названия файла