МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Алгоритмы и структуры данных»

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

Тема «Проверка корректности скобочной последовательность»

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Мухаметзянов Т. А. /

подпись инициалы, фамилия

Курс 2 Группа ИВТАПбд-21

Направление/ специальность Информатика и вычислительная техника

Руководитель старший преподаватель кафедры «Вычислительная техника»

должность, ученая степень, ученое звание

Горшков Даниил Александрович

фамилия, имя, отчество

Дата сдачи:

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Дата защиты:

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2023 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Алгоритмы и структуры данных»

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

студенту ИВТАПбд-21 Мухаметзянову Т. А.

группа фамилия, инициалы

Тема проекта (работы) Поиск корректности скобочной последовательности

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

Срок сдачи законченного проекта (работы) «22» Мая 2023г.

Исходные данные к проекту (работе)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(базовое предприятие, характер курсового проекта (работы):

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

задание кафедры, инициативная НИР, рекомендуемая литература, материалы практики)

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

Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Поясните заданный в индивидуальном задании на курсовую работу алгоритм и его место в пространстве типовых задач автоматизированных систем. Каковы теоретические оценки времени исполнения заданного в индивидуальном задании алгоритма. Какие вариации использовались в экспериментальном исследовании различных реализаций алгоритма. Какие варианты рабочей нагрузки на программы реализации алгоритма в ее экспериментальном исследовании вы использовали и на чем основан выбор. Какими средствами оцениваются затраты времени во время экспериментирования с реализациями алгоритма. Каков характер изменения времени исполнения алгоритма в зависимости от параметров рабочей нагрузки.

В какой мере экспериментальные оценки затрат времени исполнения алгоритма близки к теоретическим. Какие реализации алгоритма и при каких условиях вы бы рекомендовали на основе результатов экспериментального исследования.

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

Перечень графического материала (с точным указанием обязательных чертежей)

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

Руководитель старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ /Д. А. Горшков/

должность подпись инициалы, фамилия

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **/** Т. А. Мухаметзянов **/**

подпись инициалы, фамилия

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ  
руководителя на курсовой проект (работу)**

студента Мухаметзянова Тимура Александровича

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

фамилия, имя и отчество

Факультет Информационных Систем и технологий

группа ИВТАПбд-21

курс 2

Дисциплина Алгоритмы и структуры данных

Тема проекта (работы) Проверка корректности скобочной последовательность

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

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

Руководитель Старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ /Д. А. Горшков /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.

**Оглавление**

[Введение 5](#_Toc134893718)

[Проверка скобочной последовательности и ее Алгоритм 6](#_Toc134893719)

[Объяснение построенных алгоритмов 8](#_Toc134893720)

[Экспериментальная оценка времени работы программы 11](#_Toc134893721)

[Заключение 18](#_Toc134893722)

[Список литературы 19](#_Toc134893723)

[Листинг кода 20](#_Toc134893724)

# Введение

Проверка скобочной последовательности — это алгоритм, который определяет, является ли данная последовательность скобок корректной. В корректной последовательности каждая открывающая скобка имеет соответствующую закрывающую скобку того же типа, что является корректным выражением. В программировании этот алгоритм используется для проверки правильности расстановки скобок в коде.

Цель курсовой работы - реализация алгоритма проверки скобочной последовательности и реализация двух способов варьирования реализации по языку программирования и реализация 5 вариаций скобочных последовательностей.

Для достижения данной цели необходимо решить следующие задачи:

1. Изучить учебные, научные источники информации и статьи, посвященные данному алгоритму;
2. Реализовать алгоритм на двух языках программирования;
3. Провести анализ работы алгоритмов;
4. Сформировать заключение по окончанию выполнения курсовой работы.

# Проверка скобочной последовательности и ее Алгоритм

Основная идея алгоритма заключается в использовании стека для хранения открывающих скобок. Когда встречается открывающая скобка, она добавляется в стек, а когда встречается закрывающая скобка, алгоритм проверяет верхний элемент стека. Если верхний элемент стека является соответствующей открывающей скобкой, то он удаляется из стека. В противном случае, последовательность считается некорректной.

После прохода по всем символам последовательности, если стек оказывается пустым, это означает, что последовательность корректна. Если в стеке остаются элементы, последовательность считается некорректной.

Алгоритм проверки скобочной последовательности обычно применяется для следующих типов скобок:

1. Круглые скобки: ()
2. Квадратные скобки: []
3. Фигурные скобки: {}
4. Угловые скобки: <>
5. Угловые верхние скобки: /\

Время работы алгоритма проверки скобочной последовательности зависит от количества символов в последовательности. Алгоритм проверки скобочной последовательности работает за линейное время относительно длины входной последовательности. Теоретическая оценка времени исполнения алгоритма составляет O(n), где n - длина входной последовательности.

Алгоритм проходит по всем символам входной последовательности только один раз. Для каждого символа алгоритм выполняет проверки и операции добавления или удаления элементов из стека.

Операции добавления и удаления элементов из стека выполняются за константное время O (1).

В худшем случае, алгоритму придется обработать все символы последовательности, причем каждый из них будет добавлен или удален из стека.

Таким образом, суммарное время работы алгоритма пропорционально длине входной последовательности, что соответствует оценке O(n).

Алгоритм проверки скобочной последовательности является популярным и широко используется в различных областях IT и программирования. Некоторые из применений этого алгоритма включают:

1. Проверка корректности синтаксиса кода: компиляторы и интерпретаторы часто используют этот алгоритм для проверки правильности расстановки скобок в коде программы.
2. Обработка математических выражений: алгоритм может быть использован для проверки корректности скобочной структуры в математических выражениях и формулах.
3. Валидация и обработка данных: алгоритм может быть использован для проверки корректности структуры данных, таких как JSON, XML или других форматов, которые используют скобки для обозначения вложенности.

Популярность алгоритма проверки скобочной последовательности обусловлена его простотой, эффективностью и широким применением в различных задачах и областях.

# Объяснение построенных алгоритмов

**Алгоритм на языке C++ для обработки скобок одного типа**

Функции isOpeningBracket и isClosingBracket проверяют является ли символ открывающей или закрывающей скобкой.

Функция isMatchingBracket проверяет наличие закрывающей скобки для открывающей скобки такого же вида, например если открывающая скобка - '(', то соответствующая закрывающая скобка должна быть ')'

В функции isBalanced, мы создаем стек для хранения открывающих скобок и переменную для хранения типа скобки, который мы обнаруживаем первым в последовательности.

Мы проходимся по каждому символу в последовательности. Если это открывающая скобка, мы проверяем, соответствует ли она типу скобки bracketType. Если bracketType еще не определен, мы присваиваем ему значение текущей скобки.

Если текущая скобка не соответствует bracketType, мы сразу возвращаем false, указывая на то, что последовательность некорректна.

Если это закрывающая скобка, мы проверяем, что стек не пуст и что верхний элемент стека соответствует этой закрывающей скобке. Если это так, мы удаляем верхний элемент стека. В противном случае, мы возвращаем false, указывая на то, что последовательность некорректна.

Если символ не является скобкой, мы возвращаем false, указывая на то, что последовательность содержит недопустимый символ.

После прохода по всем символам последовательности, если стек пуст, это означает, что последовательность корректна, и мы возвращаем true. В противном случае, последовательность некорректна, и мы возвращаем false.

Для хранения элементов типа char был использован stack – это стандартный контейнер-адаптер в C++, который предоставляет структуру данных "стек" (LIFO, или "последний вошел — первый вышел").

**Алгоритм на языке C++ для обработки скобок различного типа и количества**

Функции isOpeningBracket и isClosingBracket проверяют является ли символ открывающей или закрывающей скобкой.

Функция isMatchingBracket проверяет наличие закрывающей скобки для открывающей скобки такого же вида, например если открывающая скобка - '(', то соответствующая закрывающая скобка должна быть ')'

Мы проходимся по каждому символу в последовательности. Если это открывающая скобка, то добавляем в стек. Если это закрывающая скобка, мы проверяем, что стек не пуст и что верхний элемент стека соответствует этой закрывающей скобке при помощи функции isMatchingBracket. Если это так, мы удаляем верхний элемент стека. В противном случае, мы возвращаем false, указывая на то, что последовательность некорректна. Если символ не является скобкой, мы возвращаем false, указывая на то, что последовательность содержит недопустимый символ.

После прохода по всем символам последовательности, если стек пуст, это означает, что последовательность корректна, и мы возвращаем true. В противном случае, последовательность некорректна, и мы возвращаем false.

Для хранения элементов типа char был использован stack – это стандартный контейнер-адаптер в C++, который предоставляет структуру данных "стек" (LIFO, или "последний вошел — первый вышел").

**Алгоритм на языке Python для обработки скобок одного типа**

Функции is\_opening\_bracket и is\_closing\_bracket проверяют является ли символ открывающей или закрывающей скобкой.

Функция is\_matching\_bracket проверяет наличие закрывающей скобки для открывающей скобки такого же вида, например если открывающая скобка - '(', то соответствующая закрывающая скобка должна быть ')'

В функции is\_balanced используется список brackets в качестве стека для хранения открывающих скобок.

Для каждого символа c в последовательности выполняются следующие действия:

Если c — открывающая скобка, то проверяется тип этой скобки, и, если bracket\_type еще не определена, ей присваивается данный символ. Если следующий символ не совпадает с типом символа в bracket\_type, то возвращается False.

Если c — закрывающая скобка, проверяется, не пуст ли стек и соответствует ли верхняя скобка в стеке текущей закрывающей скобке. Если условие выполняется, верхний элемент стека удаляется. В противном случае, последовательность считается некорректной, и функция возвращает False.

После обработки всех символов последовательности, если стек пуст, то последовательность считается корректной, и функция возвращает True. В противном случае, функция возвращает False.

Для хранения элементов последовательности был использован тип данных – [] – список в языке Python

**Алгоритм на языке Python для обработки скобок различного типа и количества**

Функции is\_opening\_bracket и is\_closing\_bracket проверяют является ли символ открывающей или закрывающей скобкой.

Функция is\_matching\_bracket проверяет наличие закрывающей скобки для открывающей скобки такого же вида, например если открывающая скобка - '(', то соответствующая закрывающая скобка должна быть ')'

В функции is\_balanced используется список brackets в качестве стека для хранения открывающих скобок.

Для каждого символа c в последовательности выполняются следующие действия:

Если c — открывающая скобка, то она добавляется в стек (список brackets).

Если c — закрывающая скобка, проверяется, не пуст ли стек и соответствует ли верхняя скобка в стеке текущей закрывающей скобке. Если условие выполняется, верхний элемент стека удаляется. В противном случае, последовательность считается некорректной, и функция возвращает False.

После обработки всех символов последовательности, если стек пуст, то последовательность считается корректной, и функция возвращает True. В противном случае, функция возвращает False.

Для хранения элементов последовательности был использован тип данных – [] – список в языке Python.

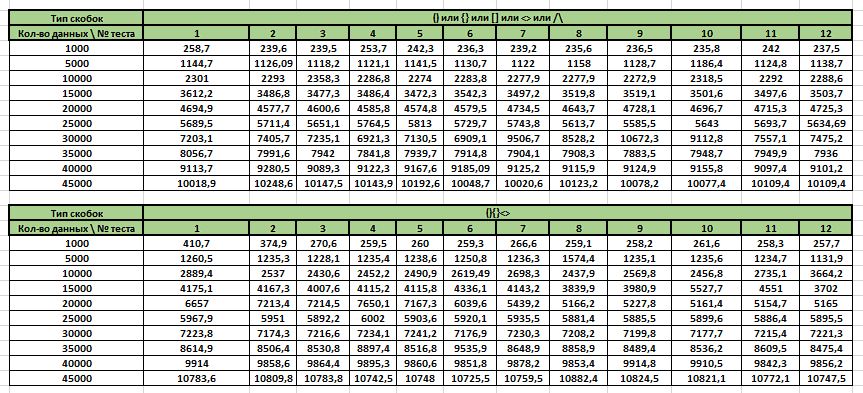
# Экспериментальная оценка времени работы программы

Для измерения времени выполнения алгоритма в C++ была использована библиотека <chrono>. Библиотека <chrono> предоставляет функции и классы для работы с временем. Все замеры времени производятся в микросекундах (1 секунда = 1000000 микросекунд). В С++ существует метод chrono::duration\_cast<microseconds> для нахождения разницы в микросекундах между началом работы алгоритма и его концом.

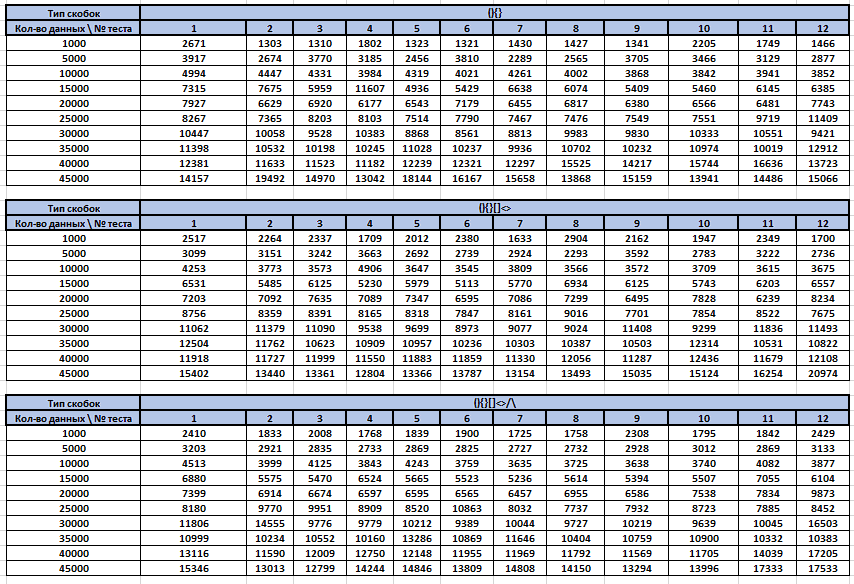
Для измерения времени выполнения алгоритма в Python был использован модуль time. Модуль time, используется для решения задач, связанных со временем. Был использован метод time. Измерения производились уже в секундах, но был использован цикл с 1000 повторений данного алгоритма из-за того, что программа отрабатывала очень быстро, и время было равно 0. Программа выполняется 1000 раз, затем производится замер среднего времени, в итоге мы получаем усредненное значение времени работы алгоритма.

Для тестов была использована программа на языке Python, которая генерирует скобочную последовательность той длины, которую укажет пользователь. Были подготовлены тесты различной длины, чтобы увидеть, как алгоритм справляется с разными размерами входных данных и сколько времени тратит на обработку. Были подготовлены 10 тестов, начиная от 1000 и заканчивая 45000 данных, с шагом в 5000. Длина последовательности обязательно должна быть четной, чтобы была сформирована корректная скобочная последовательность.

Нагрузка программы была произведена следующим образом — для каждого типа скобок по отдельности было произведено по 12 тестов, а также 12 тестов для таких последовательностей: (){}, (){}<>, (){}[]<>, (){}[]<>/\ (рисунки 1-2).



**Рисунок 1. Таблицы измерений времени работы программы на Python**



**Рисунок 2. Таблицы измерений времени работы программы на C++**

Среднее значение времени работы можно получить, найдя максимально и минимальное значения каждого теста (таблица 1-5)

***Таблица 1. Среднее время, максимальное и минимальное тестов на Python***

|  |  |  |
| --- | --- | --- |
| **() или {} или [] или <> или /\** | | |
| **макс** | **мин** | **ср** |
| **258,7** | **235,6** | **240,24** |
| **1186,4** | **1118,2** | **1133,629** |
| **2358,3** | **2272,9** | **2289,35** |
| **3612,2** | **3472,3** | **3503,18** |
| **4734,5** | **4574,8** | **4654,76** |
| **5813** | **5585,5** | **5687,509** |
| **10672,3** | **6909,1** | **7807,57** |
| **8056,7** | **7841,8** | **7931,86** |
| **9280,5** | **9089,3** | **9130,909** |
| **10248,6** | **10018,9** | **10105,09** |
|  |  |  |
| **(){}<>** | | |
| **макс** | **мин** | **ср** |
| **410,7** | **257,7** | **272,81** |
| **1574,4** | **1131,9** | **1239,04** |
| **3664,2** | **2430,6** | **2588,689** |
| **5527,7** | **3702** | **4143,21** |
| **7650,1** | **5154,7** | **6045,14** |
| **6002** | **5881,4** | **5913,73** |
| **7241,2** | **7174,3** | **7210,41** |
| **9535,9** | **8475,4** | **8620,92** |
| **9914,8** | **9842,3** | **9874,3** |
| **10882,4** | **10725,5** | **10779,24** |

***Таблица 2. Среднее время, максимальное и минимальное тестов на C++***

|  |  |  |
| --- | --- | --- |
| **(){}** | | |
| **макс** | **мин** | **ср** |
| **2671** | **1303** | **1537,4** |
| **3917** | **2289** | **3163,7** |
| **4994** | **3842** | **4102,6** |
| **11607** | **4936** | **6248,9** |
| **7927** | **6177** | **6771,3** |
| **11409** | **7365** | **7963,9** |
| **10551** | **8561** | **9766,4** |
| **12912** | **9936** | **10556,5** |
| **16636** | **11182** | **13160,3** |
| **19492** | **13042** | **15161,6** |
|  |  |  |
| **(){}[]<>** | | |
| **макс** | **мин** | **ср** |
| **2904** | **1633** | **2137,7** |
| **3663** | **2293** | **3018** |
| **4906** | **3545** | **3719,2** |
| **6934** | **5113** | **5974,8** |
| **8234** | **6239** | **7166,9** |
| **9016** | **7675** | **8207,4** |
| **11836** | **8973** | **10306,9** |
| **12504** | **10236** | **10911,1** |
| **12436** | **11287** | **11810,9** |
| **20974** | **12804** | **14241,6** |
|  |  |  |
| **(){}<>[]/\** | | |
| **макс** | **мин** | **ср** |
| **2429** | **1725** | **1946,1** |
| **3203** | **2727** | **2885,7** |
| **4513** | **3635** | **3903,1** |
| **7055** | **5236** | **5825,6** |
| **9873** | **6457** | **6965,7** |
| **10863** | **7737** | **8635,4** |
| **16503** | **9389** | **10580,2** |
| **13286** | **10160** | **10707,8** |
| **17205** | **11569** | **12307,3** |
| **17533** | **12799** | **14483,9** |

Экспериментальным путем были получены теоретические (ожидаемые) замеры времени при помощи метода наименьших квадратов (для линейной сложности). Получена разница между исходным и ожидаемым средними временами по всем тестам (Таблицы 3-7).

***Таблица 3. Среднее, ожидаемое и их разница на Python***

|  |  |  |  |
| --- | --- | --- | --- |
| **() или {} или [] или <> или /\** | **Среднее** | **Ожидаемое** | **Разница** |
| **1000** | **240,24** | 301,0901397 | -60,85014 |
| **5000** | **1133,629** | 1217,260429 | -83,63143 |
| **10000** | **2289,35** | 2362,47329 | -73,12329 |
| **15000** | **3503,18** | 3507,686151 | -4,506151 |
| **20000** | **4654,76** | 4652,899012 | 1,8609878 |
| **25000** | **5687,509** | 5798,111873 | -110,6029 |
| **30000** | **7807,57** | 6943,324735 | 864,24527 |
| **35000** | **7931,86** | 8088,537596 | -156,6776 |
| **40000** | **9130,909** | 9233,750457 | -102,8415 |
| **45000** | **10105,09** | 10378,96332 | -273,8733 |

***Таблица 4. Среднее, ожидаемое и их разница на Python***

|  |  |  |  |
| --- | --- | --- | --- |
| **(){}<>** | **Среднее** | **Ожидаемое** | **Разница** |
| **1000** | **272,81** | 529,929236 | -257,1192 |
| **5000** | **1239,04** | 1481,562507 | -242,5225 |
| **10000** | **2588,689** | 2671,104096 | -82,4151 |
| **15000** | **4143,21** | 3860,645685 | 282,56432 |
| **20000** | **6045,14** | 5050,187274 | 994,95273 |
| **25000** | **5913,73** | 6239,728863 | -325,9989 |
| **30000** | **7210,41** | 7429,270452 | -218,8605 |
| **35000** | **8620,92** | 8618,81204 | 2,1079596 |
| **40000** | **9874,3** | 9808,353629 | 65,946371 |
| **45000** | **10779,24** | 10997,89522 | -218,6552 |

***Таблица 5. Среднее, ожидаемое и их разница на C++***

|  |  |  |  |
| --- | --- | --- | --- |
| **(){}** | **Среднее** | **Ожидаемое** | **Разница** |
| **1000** | **1505,429** | 1580,548038 | -75,11947 |
| **5000** | **3054,714** | 2740,309512 | 314,40477 |
| **10000** | **3989,857** | 4190,011356 | -200,1542 |
| **15000** | **5934,286** | 5639,713199 | 294,57252 |
| **20000** | **6631,571** | 7089,415042 | -457,8436 |
| **25000** | **7957,429** | 8539,116885 | -581,6883 |
| **30000** | **9661,571** | 9988,818728 | -327,2473 |
| **35000** | **10491** | 11438,52057 | -947,5206 |
| **40000** | **13723,71** | 12888,22241 | 835,49187 |
| **45000** | **14906,43** | 14337,92426 | 568,50431 |

***Таблица 6. Среднее, ожидаемое и их разница на C++***

|  |  |  |  |
| --- | --- | --- | --- |
| **(){}[]<>** | **Среднее** | **Ожидаемое** | **Разница** |
| **1000** | **2137,7** | 1910,766885 | 226,93312 |
| **5000** | **3018** | 2992,004499 | 25,995501 |
| **10000** | **3719,2** | 4343,551516 | -624,3515 |
| **15000** | **5974,8** | 5695,098533 | 279,70147 |
| **20000** | **7166,9** | 7046,645551 | 120,25445 |
| **25000** | **8207,4** | 8398,192568 | -190,7926 |
| **30000** | **10306,9** | 9749,739586 | 557,16041 |
| **35000** | **10911,1** | 11101,2866 | -190,1866 |
| **40000** | **11810,9** | 12452,83362 | -641,9336 |
| **45000** | **14241,6** | 13804,38064 | 437,21936 |

***Таблица 7. Среднее, ожидаемое и их разница на C++***

|  |  |  |  |
| --- | --- | --- | --- |
| **(){}[]<>/\** | **Среднее** | **Ожидаемое** | **Разница** |
| **1000** | **1946,1** | 1767,079786 | 179,02021 |
| **5000** | **2885,7** | 2888,746492 | -3,046492 |
| **10000** | **3903,1** | 4290,829875 | -387,7299 |
| **15000** | **5825,6** | 5692,913258 | 132,68674 |
| **20000** | **6965,7** | 7094,996641 | -129,2966 |
| **25000** | **8635,4** | 8497,080024 | 138,31998 |
| **30000** | **10580,2** | 9899,163407 | 681,03659 |
| **35000** | **10707,8** | 11301,24679 | -593,4468 |
| **40000** | **12307,3** | 12703,33017 | -396,0302 |
| **45000** | **14483,9** | 14105,41356 | 378,48644 |

**Рисунок 3. Разница исходного времени работы алгоритма и ожидаемого**

Графики для тестов со сравнением экспериментальных замеров времени и теоретических на графиках 1 – 5.

**График 1. Сравнение среднего времени работы с ожидаемым средним временем работы для любого одного типа скобок**

**График 2. Сравнение среднего времени работы с ожидаемым средним временем работы для скобок 2 типов (){}**

**График 3. Сравнение среднего времени работы с ожидаемым средним временем работы для скобок 3 типов (){}[]**

**График 4. Сравнение среднего времени работы с ожидаемым средним временем работы для скобок 4 типов (){}[]<>**

**График 5. Сравнение среднего времени работы с ожидаемым средним временем работы для скобок всех типов (){}[]<>/\**

Характер изменения времени исполнения алгоритма в зависимости от параметров рабочей нагрузки имеет линейную зависимость, время работы зависит от количества входных данных. Путем замеров времени был сделан вывод о том, что алгоритм на языке С++ работает быстрее на больших размерах данных, чем алгоритма на языке Python на таких же, но на маленьких размерах данных программа на Python справлялась быстрее чем на языке C++. В результате построения графиков сравнения времени работы алгоритма было выявлено, что измерения отличаются от друг друга не сильно, также было определенно, что некоторые данные показали экспериментальное время быстрее, чем теоретическое.

# Заключение

В ходе курсовой работы был выполнен, рассмотрен и проанализирован алгоритм проверки скобочной последовательности, реализованный на двух языках программирования — С++ и Python для 5 видов скобок – (), {}, [], <>, /\. Были использованы стандартные средства языков программирования для измерения времени работы алгоритмов. Была проведена экспериментальная оценка алгоритмов, с получением времени работы за каждый тест, зависящие от количества входных данных или параметров рабочей нагрузки, были получены теоретические оценки затрат времени, задающие асимптотическую сложность алгоритма с помощью метода наименьших квадратов (МНК) и были построены графики по полученным данным, показывающие отличие теоретического или ожидаемого времени работы алгоритма от экспериментального. В результате выполнения данной курсовой работы можно сделать вывод о том, что алгоритм проверки скобочной последовательности можно использовать для скобочных последовательностей разной длины и типа скобок. Использование данного алгоритма показало эффективность после экспериментальных замеров при маленьких, средних и больших объемах вводимых данных.

# Список литературы

1. Правильные скобочные последовательности [электронный ресурс] – URL: https://neerc.ifmo.ru/wiki/index.php?title=Правильные\_скобочные\_последовательности
2. Правильные скобочные последовательности. Проверка на правильность. [электронный ресурс] – URL: https://foxford.ru/wiki/informatika/pravilnye-skobochnye-posledovatelnosti-proverka-na-pravilnost
3. Основы теории алгоритмов и анализа их сложности: курс лекций / Носов В.А. – М., 1992.
4. Geoffrey Viola's Blog Chrono Basics [электронный ресурс] – URL: https://geoffviola.github.io/2019/02/18/chrono-basics.html
5. Python time module [электронный ресурс] – URL: https://www.geeksforgeeks.org/python-time-module/

# Листинг кода

C++

Реализация для всех видов скобок

#include <iostream>

#include <stack>

#include <string>

#include <fstream>

#include <chrono>

using namespace std;

bool isOpeningBracket(char c) {

return c == '(' || c == '{' || c == '[' || c == '<' || c == '/';

}

bool isClosingBracket(char c) {

return c == ')' || c == '}' || c == ']' || c == '>' || c == '\\';

}

bool isMatchingBracket(char opening, char closing) {

return (opening == '(' && closing == ')') || (opening == '{' && closing == '}') || (opening == '[' && closing == ']') || (opening == '<' && closing == '>') || (opening == '/' && closing == '\\');

}

bool isBalanced(const string& sequence) {

stack<char> brackets;

for (char c : sequence) {

if (isOpeningBracket(c)) {

brackets.push(c);

}

else if (isClosingBracket(c)) {

if (brackets.empty() || !isMatchingBracket(brackets.top(), c)) {

return false;

}

brackets.pop();

}

else {

return false;

}

}

return brackets.empty();

}

int main() {

setlocale(LC\_ALL, "Russian");

string sequence;

string filename = "C:/Users/messi/PycharmProjects/CourseWorkAlgorithms/correct\_nested\_brackets\_sequence.txt";

ifstream input\_file(filename);

if (!input\_file) {

cerr << "Не удалось открыть файл " << filename << endl;

return 1;

}

input\_file >> sequence;

input\_file.close();

int num\_tests = 12;

for (int test = 1; test <= num\_tests; ++test) {

auto start\_time = chrono::high\_resolution\_clock::now();

if (isBalanced(sequence)) {

cout << "Тест " << test << ": Последовательность корректна." << endl;

}

else {

cout << "Тест " << test << ": Последовательность некорректна." << endl;

}

auto end\_time = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count();

cout << "Тест " << test << ": Время выполнения алгоритма: " << duration << " микросекунд" << endl;

cout << endl;

}

return 0;

}

Реализация для одного любого вида скобок

#include <iostream>

#include <stack>

#include <string>

#include <fstream>

#include <chrono>

using namespace std;

bool isOpeningBracket(char c) {

return c == '(' || c == '{' || c == '[' || c == '<' || c == '/';

}

bool isClosingBracket(char c) {

return c == ')' || c == '}' || c == ']' || c == '>' || c == '\\';

}

bool isMatchingBracket(char opening, char closing) {

return (opening == '(' && closing == ')')

|| (opening == '{' && closing == '}')

|| (opening == '[' && closing == ']')

|| (opening == '<' && closing == '>')

|| (opening == '/' && closing == '\\');

}

bool isBalanced(const string& sequence) {

stack<char> brackets;

char bracketType = '\0';

for (char c : sequence) {

if (isOpeningBracket(c)) {

if (bracketType == '\0') {

bracketType = c;

}

else if (c != bracketType) {

return false;

}

brackets.push(c);

}

else if (isClosingBracket(c)) {

if (brackets.empty() || !isMatchingBracket(brackets.top(), c)) {

return false;

}

brackets.pop();

}

else {

return false;

}

}

return brackets.empty();

}

int main() {

setlocale(LC\_ALL, "Russian");

string sequence;

string filename = "C:/Users/messi/PycharmProjects/CourseWorkAlgorithms/correct\_nested\_brackets\_sequence.txt";

ifstream input\_file(filename);

if (!input\_file) {

cerr << "Не удалось открыть файл " << filename << endl;

return 1;

}

input\_file >> sequence;

input\_file.close();

int num\_tests = 12;

for (int test = 1; test <= num\_tests; ++test) {

auto start\_time = chrono::high\_resolution\_clock::now();

if (isBalanced(sequence)) {

cout << "Тест " << test << ": Последовательность корректна." << endl;

}

else {

cout << "Последовательность некорректна или содержит недопустимый символ." << endl;

}

auto end\_time = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count();

cout << "Тест " << test << ": Время выполнения алгоритма: " << duration << " микросекунд" << endl;

cout << endl;

}

return 0;

}

Python

Реализация для всех видов скобок

*import* time  
*def* is\_opening\_bracket(c):  
 *return* c *in* "({[</"  
  
  
*def* is\_closing\_bracket(c):  
 *return* c *in* ")}]>\\"  
  
  
*def* is\_matching\_bracket(opening, closing):  
 *return* (opening == '(' *and* closing == ')') *or* (opening == '{' *and* closing == '}') \  
 *or* (opening == '[' *and* closing == ']') *or* (opening == '<' *and* closing == '>') \  
 *or* (opening == '/' *and* closing == '\\')  
  
  
*def* is\_balanced(sequence):  
 brackets = []  
  
 *for* c *in* sequence:  
 *if* is\_opening\_bracket(c):  
 brackets.append(c)  
 *elif* is\_closing\_bracket(c):  
 *if not* brackets *or not* is\_matching\_bracket(brackets[-1], c):  
 *return False* brackets.pop()  
  
 *return not* brackets  
  
  
*def* read\_from\_txt\_file(filename):  
 *with open*(filename, 'r') *as* f:  
 *return* f.read()  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 filename = "correct\_nested\_brackets\_sequence.txt"  
 sequence = read\_from\_txt\_file(filename)  
 *print*(f"Считанная последовательность: {sequence}")  
  
 num\_tests = 12  
  
 *for* test *in range*(1, num\_tests + 1):  
 start\_time = time.perf\_counter()  
  
 result = is\_balanced(sequence)  
  
 end\_time = time.perf\_counter()  
  
 *if* result:  
 *print*(f"Тест {test}: Последовательность корректна.")  
 *else*:  
 *print*(f"Тест {test}: Последовательность некорректна.")  
  
 duration = (end\_time - start\_time) \* 1\_000\_000 *# миллисекунды  
  
 print*(f"Тест {test}: Время выполнения алгоритма: {duration} миллисекунд")

Реализация для одного вида скобок

*import* time  
  
*def* is\_opening\_bracket(c):  
 *return* c *in* "({[</"  
  
  
*def* is\_closing\_bracket(c):  
 *return* c *in* ")}]>\\"  
  
  
*def* is\_matching\_bracket(opening, closing):  
 *return* (opening == '(' *and* closing == ')') *or* (opening == '{' *and* closing == '}') \  
 *or* (opening == '[' *and* closing == ']') *or* (opening == '<' *and* closing == '>') \  
 *or* (opening == '/' *and* closing == '\\')  
  
  
*def* is\_balanced(sequence):  
 brackets = []  
 bracket\_type = '\0'  
 *for* c *in* sequence:  
 *if* is\_opening\_bracket(c):  
 *if* bracket\_type == '\0':  
 bracket\_type = c  
 *elif* c != bracket\_type:  
 *return False* brackets.append(c)  
 *elif* is\_closing\_bracket(c):  
 *if not* brackets *or not* is\_matching\_bracket(brackets[-1], c):  
 *return False* brackets.pop()  
  
 *return not* brackets  
*def* read\_from\_txt\_file(filename):  
 *with open*(filename, 'r') *as* f:  
 *return* f.read()  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 filename = "correct\_nested\_brackets\_sequence.txt"  
 sequence = read\_from\_txt\_file(filename)  
 *print*(f"Считанная последовательность: {sequence}")  
  
 num\_tests = 12  
  
 *for* test *in range*(1, num\_tests + 1):  
 start\_time = time.perf\_counter()  
  
 result = is\_balanced(sequence)  
  
 end\_time = time.perf\_counter()  
  
 *if* result:  
 *print*(f"Тест {test}: Последовательность корректна.")  
 *else*:  
 *print*(f"Тест {test}: Последовательность некорректна.")  
  
 duration = (end\_time - start\_time) \* 1\_000\_000 *# миллисекунды  
  
 print*(f"Тест {test}: Время выполнения алгоритма: {duration} миллисекунд")