Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Рязанский государственный радиотехнический университет  
имени В.Ф. Уткина

Кафедра САПР ВС

К защите

Руководитель работы:

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

дата, подпись

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

по дисциплине

**«Структуры и алгоритмы обработки данных»**

Тема:

«Синтаксический анализ скобочных структур»

Выполнил студент группы 045

Анохин В.А.

дата сдачи на проверку, подпись

Руководитель работы

д.т.н., профессор кафедры САПР ВС

Скворцов С.В.

оценка дата защиты, подпись

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

#### **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

Кафедра САПР ВС

**ЗАДАНИЕ**

на курсовой проект по дисциплине

**«Структуры и алгоритмы обработки данных»**

Студенту Анохину Вячеславу Алексеевичу группа 045 \_

Тема: Синтаксический анализ скобочных структур\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Вариант № 36

**Примерное содержание пояснительной записки**

Введение

1. Постановка задачи

2. Описание и анализ алгоритмов решения задачи

2.1. Описание обобщенного алгоритма

2.2. Анализ особенностей, достоинств и недостатков

2.3. Описание используемых структур данных

2.4. Детализация алгоритма

3. Разработка программного обеспечения

3.1. Структура программы

3.2. Основные переменные, массивы и структуры данных

3.3. Основные подпрограммы

4. Экспериментальная часть

4.1. Ручное решение тестовых задач

4.2. Машинное решение тестовых задач

4.3. Результаты экспериментальных исследований программы

5. Программная документация

Заключение

Список использованной литературы

Приложения:

- текст программы;

-листинг с результатами машинного решения

Дата выдачи задания\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Срок сдачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Скворцов С.В.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Анохин В.А.

**Вариант 36**

Синтаксический анализ скобочных структур

Выполнить разработку программы, которая считывает текст прикладной

программы на языке высокого уровня (например, Паскаль) и выполняет синтаксический анализ скобочных структур, т.е. корректность использования скобок следующих видов:

( ), [ ], { }, (\* \*)

с использованием алгоритма, описанного в разделе 2.3 работы [1]. При разработке программы необходимо обеспечить диагностику возможных ошибок с визуализацией их расположения в тексте программы. Исходный текст должен считываться из файла, список обнаруженных ошибок также должен сохраняться в файле для анализа.

При оформлении программной документации учитывать требования, изложенные в работе [2].

Литература

1. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 368 с.

2. Структуры и алгоритмы обработки данных: методические указания к

курсовому проектированию / Рязан. гос. радиотехн. ун-т; сост. С.В. Скворцов,

В.И. Хрюкин. Рязань, 2021. 16 с. (номер в каталоге 5982)

**Содержание**

[Введение 5](#_Toc154146616)

[1. Постановка задачи 6](#_Toc154146617)

[2. Описание и анализ алгоритмов решения задачи 7](#_Toc154146618)

[2.1. Описание обобщенного алгоритма 7](#_Toc154146619)

[2.2. Анализ особенностей, достоинств и недостатков 8](#_Toc154146620)

[2.3. Описание используемых структур данных 9](#_Toc154146621)

[3. Разработка программного обеспечения 11](#_Toc154146622)

[3.1. Структура программы 11](#_Toc154146623)

[3.2. Основные переменные, массивы и структуры данных 15](#_Toc154146624)

[3.3. Основные подпрограммы 16](#_Toc154146625)

[4. Экспериментальная часть 17](#_Toc154146626)

[4.1. Ручное решение тестовых задач 17](#_Toc154146627)

[4.2. Машинное решение тестовых задач 19](#_Toc154146628)

[5. Программная документация 20](#_Toc154146629)

[Заключение 22](#_Toc154146630)

[Список использованной литературы 23](#_Toc154146631)

[Приложение А – текст программы 24](#_Toc154146632)

[Приложение Б – листинг с результатами машинного решения 28](#_Toc154146633)

# Введение

Синтаксический анализ является важной составной частью процесса разработки программного обеспечения, особенно при работе с языками высокого уровня. Он направлен на обнаружение и корректное понимание структуры кода, что включает в себя проверку соответствия скобочных структур. В рамках данной курсовой работы рассматривается разработка программы, способной проводить синтаксический анализ скобочных структур в тексте прикладной программы на языке высокого уровня, таком как С++.

Данная курсовая работа имеет практическое значение для разработчиков программного обеспечения, поскольку предоставляет инструмент для проверки синтаксической правильности расстановки скобок в исходном коде, что способствует повышению качества программ и облегчает процесс их разработки и отладки. Поэтому разработка программного инструмента, способного автоматически обнаруживать и анализировать такие ошибки, является актуальной и востребованной задачей в области разработки программного обеспечения.

Целью данной работы является создание программы, способной проводить синтаксический анализ скобочных структур в тексте программы на языке высокого уровня. Программа должна предоставлять возможность обнаружения и визуализации ошибок в расстановке скобок, а также осуществлять чтение и сохранение информации из файла для последующего анализа.

# 1. Постановка задачи

Данная курсовая работа нацелена на разработку программы, способной проводить синтаксический анализ скобочных структур в тексте программы на языке высокого уровня. Задача формулируется следующим образом:

1. Разработать модуль программы, который будет осуществлять чтение исходного кода из файла для последующего анализа скобочных структур.
2. Разработать и реализовать алгоритм программы для обнаружения ошибок в расстановке скобок. Для этого программа должна выявлять некорректные скобочные структуры и фиксировать их положение в тексте.
3. Реализовать функционал, который позволит визуализировать расположение обнаруженных ошибок в скобочных структурах.
4. Разработать механизм сохранения списка обнаруженных ошибок в отдельный файл для дальнейшего анализа и исправления.
5. Провести тестирование разработанной программы на различных примерах, включая сценарии с корректными и некорректными скобочными структурами, с целью проверки ее корректной работы и эффективности обнаружения ошибок.
6. Составить программную документацию

# 2. Описание и анализ алгоритмов решения задачи

## 2.1. Описание обобщенного алгоритма

Определим критерии правильно построенной последовательности:

1. Последовательности ( ), [ ], { } и (\* \*) являются правильно построенными.
2. Если последовательность х правильно построенная, то правильно построены и последовательности (х), [х], {х} и (\*х\*).
3. Если последовательности x и y правильно построенные, то такова же и последовательность xy.
4. Правильно построенными являются лишь те последовательности, правильность которых следует из конечной последовательности применений правил 1, 2 и 3.

Для установления правильности выражения используется стековая память. Алгоритм синтаксический анализ скобочных структур сводится к последовательному чтению текстовой строки и работе со стеком.

Алгоритм имеет следующий вид:

1. Создать пустой стек для хранения скобок
2. Прочитать проверяемый текст
3. Для каждого символа в тексте выполнить следующие шаги:
   1. Считать текущий символ
   2. Если символ является открывающей скобкой, то поместить её в стек
   3. Если символ является закрывающей скобкой, то:
      1. Если стек пустой, то вывести «Отсутствует открывающая скобка», иначе извлечь вершину стека
      2. Если извлеченная скобка не соответствует закрывающей скобке, то вывести «Неверная закрывающая скобка»
4. Проверить стек на наличие элементов:

если стек пустой, то вывести: «Ошибки не обнаружены»,

иначе вывести «Не все скобки закрыты»

## 2.2. Анализ особенностей, достоинств и недостатков

Определим вычислительную сложность алгоритма синтаксического анализа скобочных структур. Вычислительная сложность данного алгоритма состоит из двух частей:

1. Необходимо проверить каждый символ текста длиной n
2. Каждый символ необходимо сравнивать с каждым видом скобок, пока не будет определен тип скобки или подтвердится, что символ не является скобкой.

Максимальное количество сравнений m зависит от количества описанных видов скобок и символов в составных скобках, поскольку проверка на соответствие скобке прерывается после первого несовпадения. При проверке корректности расстановки скобок ( ), [ ], { } и (\* \*) получим m = 9, т.к. первые символы скобок (\* и ( совпадают.

Работа со стеком происходит за конечное время, поэтому на вычислительную сложность алгоритма не влияет.

Таким образом, время выполнения программы зависит от длины текста, его содержимого и количества описанных с скобочных структур. В худшем случае, если передать текст без скобок, получим сложность O(n\*m).

Программа может завершить выполнение на первом шаге, если первым символом окажется закрывающая скобка.

Недостатком данного алгоритма является то, что он не анализирует иные синтаксические структуры. Например, следующий корректный код программы будет ошибочным для данного анализатора:

char ch = '('

if (ch == '(') { }

В данном примере имеется три открывающих скобки ( и только одна закрывающая, что приведет к ошибке.

## 2.3. Описание используемых структур данных

Для реализации данного алгоритма используется стек. Стек – это абстрактный тип данных, представляющий собой список элементов, организованный по принципу LIFO (last in – first out, «последним пришел – первым вышел»).

Возможные операции со стеком: добавление элемента в вершину(push), удаление вершины стека с получением элемента (pop), чтение вершины стека, проверка на наличие элементов в стеке.

В данной работе был использован стек на указателях, разработанный в ходе выполнения лабораторных работ.

Для хранения видов обрабатываемых скобок были использованы массивы. Массив – это последовательность однотипных данных в памяти, имеющее общее имя, доступ к элементам которой осуществляется по смещению относительно первого (нулевого) элемента.

Для чтения и записи файлов использовались потоки ввода-вывода ifstream и ofstream из стандартной библиотеки C++ fstream.

**2.4. Детализация алгоритма**

Чтобы обеспечить универсальность алгоритма и простоту изменения видов скобочных структур, видов обрабатываемых скобок были сохранены в массивах. Было принято решения использовать отдельные массивы для открывающих и закрывающих скобок для упрощения определения вида скобки (открывающая или закрывающая) и для быстрого поиска пары скобки. Соответствующие друг другу пары скобок расположены под одинаковыми индексами в массиве. Текущий символ проверяется поочередно с каждым элементом массивов.

Также добавлена возможность проверки скобок, состоящих из нескольких символов. Для этого нужно проверять каждый символ такой скобочной структуры с текстом. В связи с этим виды скобок, у которых первые символы совпадают друг с другом, должны располагаться в порядке убывания длины записи.

Например, если скобу ( расположить в массиве раньше (\*, то последняя никогда не будет определена.

# 3. Разработка программного обеспечения

## 3.1. Структура программы

На рисунке 1 показана функция проверки текущего символа на принадлежность к скобочной структуре.

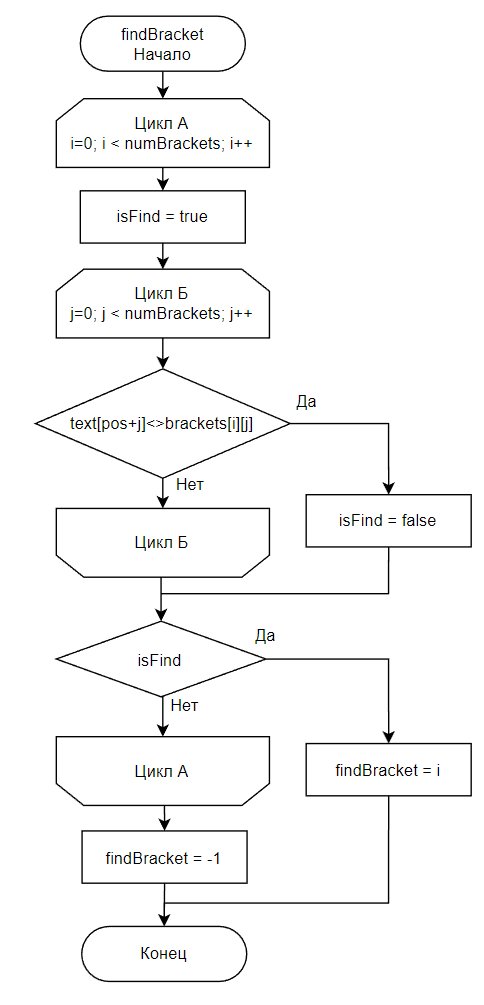


Рисунок 1 – Функция findBraket

На рисунках 2 – 4 показана схема основной программы.

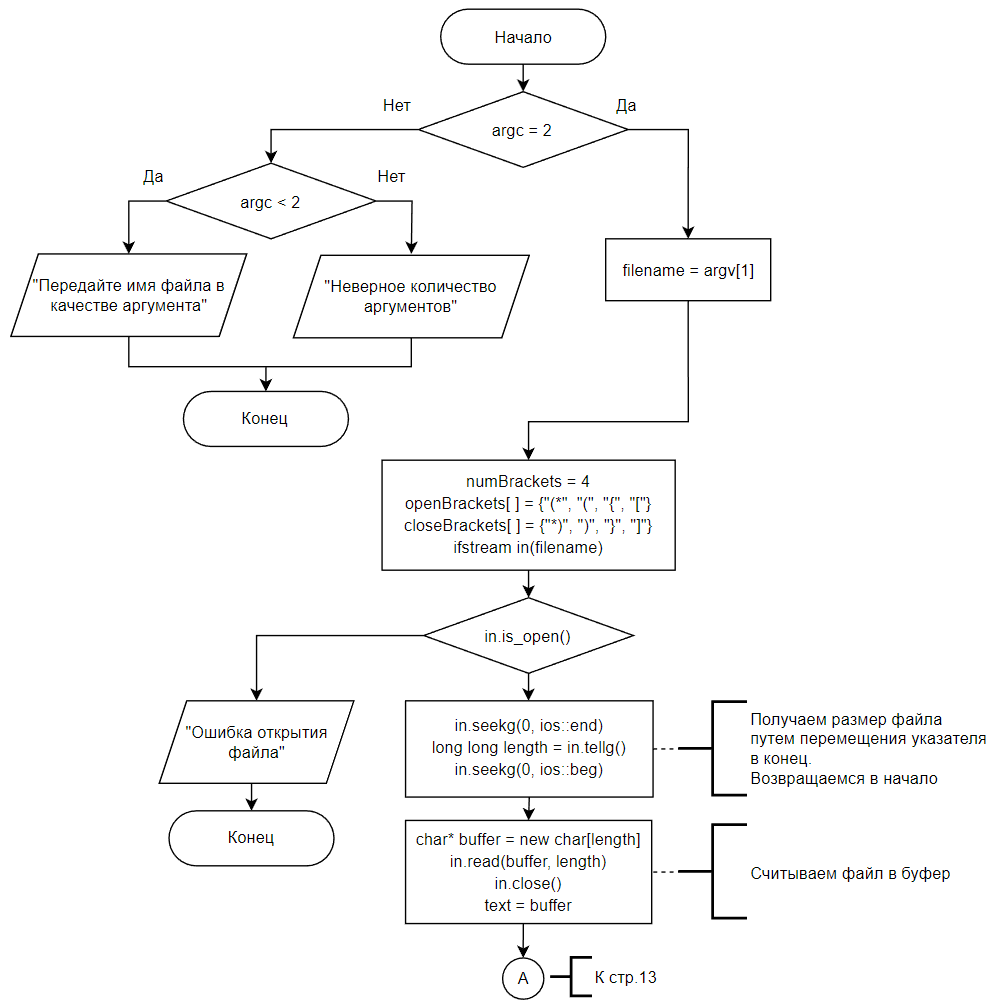


Рисунок 2 – Чтение текстового файла

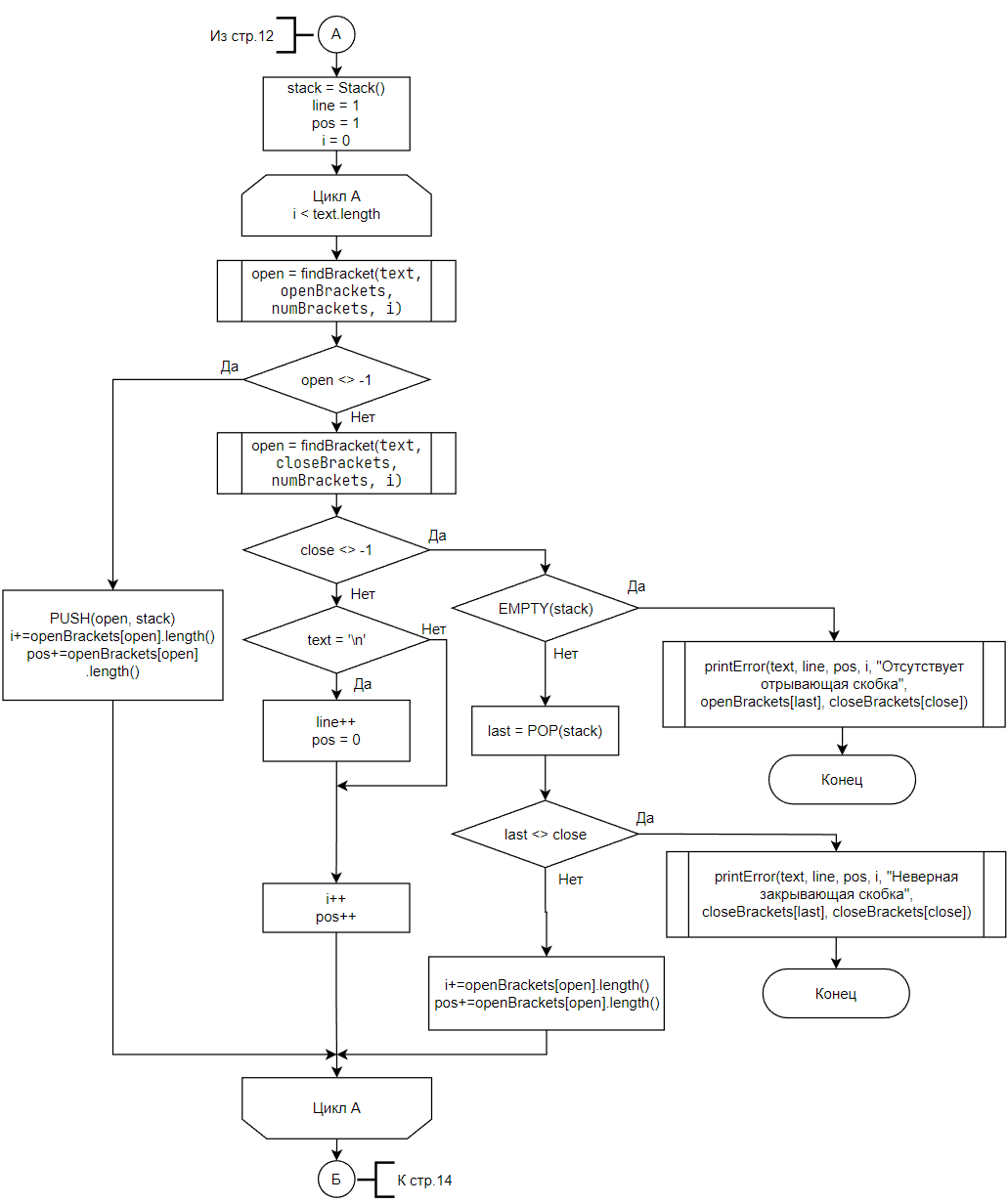


Рисунок 3 – Основной алгоритм

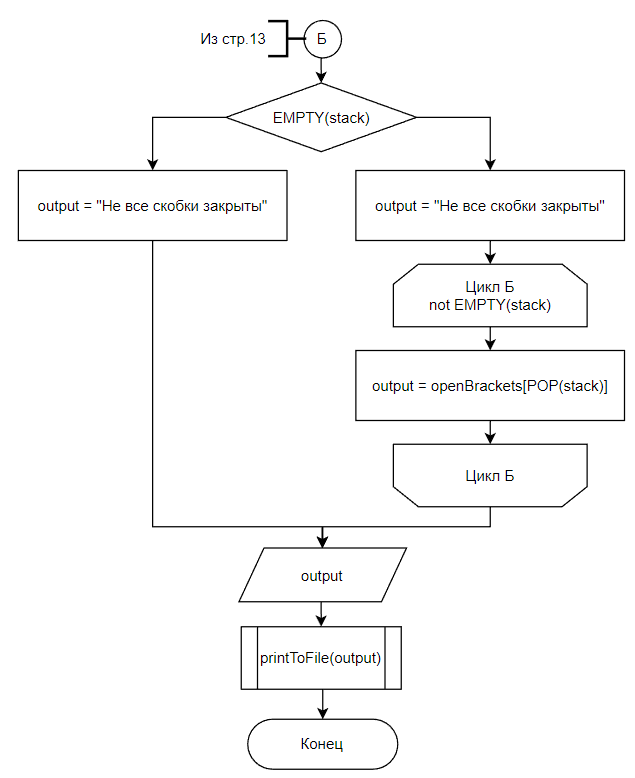


Рисунок 4 – Вывод результатов

## 3.2. Основные переменные, массивы и структуры данных

В реализации алгоритма были использованы следующие переменные, массивы и структуры данных:

* argv – массив, хранящий аргументы командной строки;
* argc– количество принятых аргументов командной строки;
* filename– имя проверяемого файла;
* in – поток для чтения файла;
* out – поток для записи в файл;
* text– считанный из файла текст;
* openBrackets– массив, в котором описаны открывающие скобки;
* closeBrackets– массив, в котором описаны закрывающие скобки;
* numBrackets– количество видов скобок;
* stack– хранит найденные открывающие скобки в виде индексов элементов массива openBrackets;
* line*,* pos *–* номер текущей строки и позиция в ней;
* i – текущая позиция в тексте
* open*,* close *–* индекс найденной скобки в массиве openBrackets *и* closeBrackets*.* Имеют значение -1, если текущий символ не является скобкой;
* last *–* индекс найденной скобки извлеченной из стека;
* output – строка, в которую записывается результат работы программы.

## 3.3. Основные подпрограммы

В реализации алгоритма были использованы следующие подпрограммы:

* void printToFile(const string& output) – сохраняет в текстовый файл результат выполнения программы output.
* void printError(const string& text, int line, int line\_pos, int pos, const string& message, const string& expect, const string& found) – формирует сообщение об ошибке, выводит её на экран и вызывает printToFile() для сохранения результата в файл. Описание принимаемых параметров:

text – ссылка на проверяемый текст

line – номер строки, в которой обнаружена ошибка

line\_pos – позиция в строке, в которой обнаружена ошибка

pos – позиция ошибки в тексте

message – сообщение об ошибке

expect – требуемая скобка

found – найденная скобка

* int findBracket(const string& text, const string brackets[], int numBrackets, int pos) – проверяет наличие скобки в тексте text на позиции pos. Виды проверяемых скобок передаются в виде массива brackets длиной numBrackets.
* void PUSH(my\_type val, Stack\* stack) – добавляет в стек stack элемент val.
* my\_type POP(Stack\* stack) – извлекает вершину стека stack.
* bool EMPTY(Stack\* stack) – проверяет стек на наличие элемента. Если стек пустой вернет true, иначе false
* seekg(off\_type, \_Ios\_Seekdir) – перемещает указатель в файле
* long long tellg() – возвращает позицию указателя в файле
* read(char\_type \*\_\_s, streamsize \_\_n) – считывает содержимое файла длиной \_\_n в буфер \_\_s.

# 4. Экспериментальная часть

## 4.1. Ручное решение тестовых задач

Рассмотрим синтаксический анализ скобочных структур на следующем примере:

(\*[Comment]\*)

{y\*(х[]+())}

{(\*}\*)

Таблица 1 – Проверка корректности расстановки скобок

|  |  |  |  |
| --- | --- | --- | --- |
| № | Текущий символ | Действие | Состояние стека |
| 1 | ( | Проверка следующего символа |  |
| 2 | (\* | Добавляем в стек | (\* |
| 3 | [ | Добавляем в стек | (\*, [ |
| 4-11 | Комментарий | – | (\*, [ |
| 12 | ] | Извлекаем из стека | (\* |
| 13 | \* | Проверка следующего символа | (\* |
| 14 | \*) | Извлекаем из стека |  |
| 15 | \n | – |  |
| 16 | { | Добавляем в стек | { |
| 17 | y | – | { |
| 18 | \* | Проверка следующего символа.  – | { |
| 19 | ( | Проверка следующего символа.  Добавляем в стек | {, ( |
| 20 | x | – | {, ( |
| 21 | [ | Добавляем в стек | {, (, [ |
| 22 | ] | Извлекаем из стека | {, ( |
| 23 | + | – | {, ( |

Таблица 1 – Проверка корректности расстановки скобок(продолжение)

|  |  |  |  |
| --- | --- | --- | --- |
| 24 | ( | Проверка следующего символа.  Добавляем в стек | {, (, ( |
| 25 | ) | Извлекаем из стека | {, ( |
| 26 | ) | Извлекаем из стека | { |
| 27 | } | Извлекаем из стека |  |
| 28 | \n | – |  |
| 29 | [ | Добавляем в стек | [ |
| 30 | ( | Проверка следующего символа | [ |
| 31 | (\* | Добавляем в стек | [, (\* |
| 32 | } | Извлекаем из стека | [ |
| Ошибка: неверная закрывающая скобка | | | |

Результатом решения данного примера является ошибка в третьей строке в четвертом символе: при открывающей скобке (\* встретилась закрывающая скобка }. Чтобы исправить ошибку необходимо поменять местами } и \*).

## 4.2. Машинное решение тестовых задач

Введем в текстовый файл пример из пункта 4.1 и запустим программу. Результат выполнения программы:

Ошибка в строке 3 на позиции 4

Неверная закрывающая скобка

Ожидается: '\*)', найдено: '}'

{(\*}\*)

\_\_\_^\_\_

Исправим ошибку в третьей строке:

(\*[Comment]\*)

{y\*(х[]+())}

{(\*\*)}

Результат выполнения программы:

Ошибки не обнаружены

Результаты ручного и машинного решения тестовой задачи совпадают.

Тестирование всех вариантов ошибок приводится в листинге приложения Б.

# 5. Программная документация

Руководство оператора.

1. Назначение программы

Данная программа предназначена для проверки правильности расстановки скобок в текстовых файлах. Проверка корректности использования затрагивает только заданные виды скобок: ( ), [ ], { }, (\* \*). Иные символы, входящие в текстовый файл, игнорируются.

Программа выполняется в консольном режиме. Результат проверки отображается в консоли и сохраняется в текстовом файле "brackets\_analyze\_result.txt" в каталоге с исполняемым файлом программы.

1. Условия выполнения программы

Для выполнения программы должны быть выполнены следующие технические требования:

Операционная система: Windows

Оперативная память: <10 МБ

Свободное место на диске: <3 МБ

1. Выполнение программы

Для проверки текстового файла необходимо указать в консоли путь к исполняемому файлу программы, а затем через пробел указать путь к текстовому файлу.

Формат пути зависит от текущего каталога в консоли. Если текстовый файл и\или исполняемый файл находятся в текущем каталоге, то можно использовать относительный путь.

Основные варианты команды запуска:

1. Файл и программа находятся не в текущем каталоге

>C:\folfer1\CourseworkAlgs.exe C:\folfer2\file.txt

1. Файл находятся в текущем каталоге

>C:\folfer1\CourseworkAlgs.exe file.txt

1. Программа находятся в текущем каталоге

>.\CourseworkAlgs.exe C:\folfer2\file.txt

1. Файл и программа находятся в текущем каталоге

>.\CourseworkAlgs.exe file.txt

1. Сообщения оператору

После запуска программы получим один из возможных результатов:

1. Программа не смогла открыть указанный файл. Наиболее вероятная причина ошибки – указанный файл не существует.

Ошибка открытия файла

1. Программа успешно завершила выполнение

Ошибки не обнаружены

1. Сообщение об ошибке. Указывается местоположение ошибки, текст ошибки, выводится полностью строка с ошибкой и указатель на некорректную скобку.

Ошибка в строке 2 на позиции 14

Неверная закрывающая скобка

Ожидается: ')' ,найдено: '}'

(2+arr[2]})/4(

\_\_\_\_\_\_\_\_\_\_\_\_\_^\_\_\_\_

Возможны три варианта ошибок: «Отсутствует открывающая скобка», «Неверная закрывающая скобка» и «Не все скобки закрыты».

# Заключение

В ходе курсовой работы был изучен синтаксический анализ скобочных структур. Данный алгоритм был реализован в виде консольного приложения на языке C++.

Разработанное приложение позволяет проверять корректность расстановки скобок в текстовых файлах и выводит в консоль и текстовый файл результаты проверки в удобном для пользователя формате.

Также было проведено тестирование разработанной программы на различных примерах, охватывающие различные ситуации. Приложение корректно работает в основных случаях, но было найдено ограничение, связанное с проверкой кода программ: алгоритм не позволяет игнорировать скобочные структуры внутри кавычек.

# Список использованной литературы

1. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 368 с.

2. Структуры и алгоритмы обработки данных: методические указания к

курсовому проектированию / Рязан. гос. радиотехн. ун-т; сост. С.В. Скворцов,

В.И. Хрюкин. Рязань, 2021. 16 с. (номер в каталоге 5982)

3. Списочные структуры данных: методические указания к лабораторным работам / Рязан. гос. радиотехн. ун-т; сост.: С.В. Скворцов, В.И. Хрюкин. Рязань, 2023. 32 с.

# Приложение А – текст программы

#include <iostream>  
#include <fstream>  
#include <windows.h>  
#include "StackList.h"  
using namespace std;  
using namespace stackList;  
  
void printToFile(const string& output) {  
 ofstream out("brackets\_analyze\_result.txt");  
 if(out.is\_open()){  
 out<<output;  
 }  
 else{  
 cout<<"Ошибка сохранения в файл"<<endl;  
 }  
 out.close();  
}  
  
void printError(const string& text, int line, int line\_pos, int pos, const string& message, const string& expect, const string& found){  
 string output = "Ошибка в строке " + to\_string(line) + " на позиции " + to\_string(line\_pos) + "\n"+  
 message + "\n" +  
 "Ожидается: '" + expect + "', найдено: '" + found + "'\n";  
 string str;  
 str += text[pos];  
 string pointer;  
 for (int i=pos-1; i>=0; i--){  
 char c = text[i];  
 if (c=='\n') break;  
 str = c + str;  
 pointer+='\_';  
 }  
 pointer+='^';  
 for (int i=pos+1; i<text.length(); i++){  
 char c = text[i];  
 if (c=='\n') break;  
 str+=c;  
 pointer+='\_';  
 }  
 output += str+"\n"+pointer+"\n";  
 cout<<output;  
 printToFile(output);  
}  
  
int findBracket(const string& text, const string brackets[], int numBrackets, int pos){  
 for(int i=0; i<numBrackets;i++){  
 bool isFind = true;  
 for(int j=0; j<brackets[i].length();j++){  
 if(text[pos+j]!=brackets[i][j]){  
 isFind = false;  
 break;  
 }  
 }  
 if(isFind) return i;  
 }  
 return -1;  
}  
  
int main(int argc, char\* argv[]) {  
 SetConsoleCP(65001);  
 SetConsoleOutputCP(65001);  
 string filename;  
 if(argc == 2){  
 filename = argv[1];  
 }  
 else{  
 if(argc < 2){  
 cout<<"Передайте имя файла в качесте аргумента"<<endl;  
 }  
 else cout<<"Неверное количество аргументов"<<endl;  
 return 0;  
 }  
 int numBrackets = 4;  
 string openBrackets[] = {"(\*", "(", "{", "["};  
 string closeBrackets[] = {"\*)", ")", "}", "]"};  
 ifstream in(filename);  
 string text;  
 if(in.is\_open()){  
 in.seekg(0, ios::end);  
 long long length = in.tellg();  
 in.seekg(0, ios::beg);  
 char\* buffer = new char[length];  
 in.read(buffer, length);  
 in.close();  
 text = buffer;  
 }  
 else {  
 cout << "Ошибка открытия файла" << endl;  
 return 0;  
 }  
 Stack stack = Stack();  
 int line = 1, pos = 1, i = 0;  
 while(i<text.length()){  
 int open = findBracket(text, openBrackets, numBrackets, i);  
 if(open != -1){  
 PUSH(open, &stack);  
 i+=openBrackets[open].length();  
 pos+=openBrackets[open].length();  
 }  
 else{  
 int close = findBracket(text, closeBrackets, numBrackets, i);  
 if(close != -1){  
 if(EMPTY(&stack)){  
 printError(text, line, pos, i, "Отсутствует отрывающая скобка", openBrackets[close], closeBrackets[close]);  
 return 0;  
 }  
 int last = POP(&stack);  
 if (last != close){  
 printError(text, line, pos, i, "Неверная закрывающая скобка", closeBrackets[last], closeBrackets[close]);  
 return 0;  
 }  
 i+=closeBrackets[close].length();  
 pos+=closeBrackets[close].length();  
 }  
 else{  
 if(text[i]=='\n'){  
 line++;  
 pos = 0;  
 }  
 i++;  
 pos++;  
 }  
 }  
 }  
 string output;  
 if(EMPTY(&stack)){  
 output = "Ошибки не обнаружены\n";  
 } else {  
 output = "Не все скобки закрыты: ";  
 while(!EMPTY(&stack)){  
 output+=openBrackets[POP(&stack)]+" ";  
 }  
 output+="\n";  
 }  
 cout << output;  
 printToFile(output);  
 return 0;  
}

# Приложение Б – листинг с результатами машинного решения

Содержимое файла file.txt

(\*[Comment]\*)  
{y\*(х[]+())}  
{(\*}\*)

Содержимое файла file2.txt

(\*[Comment]\*)  
{y\*(х[]+())}  
{(\*\*)}

Содержимое файла file3.txt

{y\*(х[]+())}[  
{(\*\*)}

Содержимое файла file4.txt

[Comment])

**Окно консоли:**

PS > .\CourseworkAlgs.exe file.txt

Ошибка в строке 3 на позиции 4

Неверная закрывающая скобка

Ожидается: '\*)', найдено: '}'

{(\*}\*)

\_\_\_^\_\_

PS > .\CourseworkAlgs.exe file1.txt

Ошибка открытия файла

PS > .\CourseworkAlgs.exe file2.txt

Ошибки не обнаружены

PS > .\CourseworkAlgs.exe file3.txt

Не все скобки закрыты: [

PS > .\CourseworkAlgs.exe file4.txt

Ошибка в строке 1 на позиции 10

Отсутствует отрывающая скобка

Ожидается: '(', найдено: ')'

[Comment])

\_\_\_\_\_\_\_\_\_^

PS >