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**

**ОТЧЕТ   
по учебной практике**

бакалавра направления 09.03.04 "Программная инженерия"

|  |  |
| --- | --- |
|  | Выполнил:\_\_\_\_\_\_\_\_\_\_\_\_ студент группы КЭ-143  Д.С. Чистяков  Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_  преподаватель кафедры СП  A.A. Сахарова  Дата:\_\_\_\_\_\_\_\_, Оценка:\_\_\_\_\_\_\_\_\_\_ |

Челябинск, 2023 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

системного программирования

\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

**ЗАДАНИЕ**

**по** **учебной практике**

1. **Цель работы**

Необходимо разработать распознаватель заданной символьной цепочки. Символьная цепочка задается с помощью формул Бэкуса-Наура.

1. **Исходные данные к работе**
2. *Йенсен К., Вирт Н.* Паскаль. Руководство пользователя и описание языка. М.: Компьютер, 1995.
3. *Льюис Ф., Розенкранц Д., Стирнз Р.* Теоретические основы проектирования компиляторов. М.: Мир, 1979.
4. **Перечень подлежащих разработке вопросов**
5. Выполнить анализ требований и разработать спецификации программы.
6. Провести проектирование программы.
7. Реализовать спроектированные модули.
8. Провести тестирование и отладку реализованных модулей.
9. **Сроки**

Дата выдачи задания: "26" июня 2023 г.

Срок сдачи законченной работы: "21" июля 2023 г.

**Руководитель:**

преподаватель кафедры СП \_\_\_\_\_\_\_\_\_\_\_\_\_ Сахарова A.A.

должность, ученая степень подпись ФИО преподавателя

**Задание принял к исполнению:**

\_\_\_\_\_\_\_\_\_\_\_\_\_ Чистяков Д.С.

подпись ФИО студента

**ОГЛАВЛЕНИЕ**

[1. СПЕЦИФИКАЦИЯ 4](#_Toc139651612)

[2. ПРОЕКТИРОВАНИЕ 6](#_Toc139651613)

[2.1. Модульная структура 6](#_Toc139651614)

[2.2. Интерфейсы модулей 6](#_Toc139651615)

[3. КОДИРОВАНИЕ 10](#_Toc139651616)

[3.1. Структура текста программы 10](#_Toc139651617)

[3.2. Алгоритмы реализации модулей 11](#_Toc139651618)

[3.2.1. Блок транслитерации 11](#_Toc139651619)

[3.2.2. Лексический блок 12](#_Toc139651620)

[3.2.3. Синтаксический блок 16](#_Toc139651621)

[3.2.4. Блок идентификации ключевых слов 16](#_Toc139651622)

[3.3. Размер текста программы 17](#_Toc139651623)

[4. ТЕСТИРОВАНИЕ 18](#_Toc139651624)

[4.1. Автономное тестирование 18](#_Toc139651625)

[4.2. Комплексное тестирование 21](#_Toc139651626)

[ЗАКЛЮЧЕНИЕ 22](#_Toc139651627)

[ЛИТЕРАТУРА 23](#_Toc139651628)

# СПЕЦИФИКАЦИЯ

Необходимо разработать распознаватель заданной символьной цепочки. Символьная цепочка задается с помощью формул Бэкуса-Наура, представленная в листинге 1.

Листинг 1 – Формула цепочки

<цепочка>::=<описание константы>;

<значение>::=<целая константа> | <16-ричная константа> | <вещественная константа>| <строковая константа>

<строковая константа>::='<цифра> | <идентификатор> |

<цифра><идентификатор>'

<целая константа>::=<целое со знаком> | <целое без знака>

<целое со знаком>::=<знак><целое без знака>

<целое без знака>::=<цифра> | <цифра><целое без знака>

<цифра>::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<знак>::=+ | -

<16-ричная константа>::= $<список 16-ричных букв и цифр>

<вещественная константа>::=<целое со знаком> | <вещественное со знаком> | <вещественное без знака>

<идентификатор>::= <буква> | <идентификатор><буква> | <идентификатор><цифра>

<буква>::=A | B | C | D | E | F | ... | Z

**Описание входных данных**

Цепочка записана в текстовом файле INPUT.TXT, который состоит из одной строки. Длина цепочки не превышает 80 символов.

**Описание выходных данных**

Результат распознавания необходимо записать в текстовый файл OUTPUT.TXT в одно из следующих сообщений: ACCEPT, если цепочка допустима, и REJECT, если цепочка недопустима.

**Примеры входных и выходных данных**

В таблице 1 приведены примеры входных и соответствующих им выходных данных.

Таблица 1 – Примеры входных и выходных данных

|  |  |
| --- | --- |
| **INPUT.TXT** | **OUTPUT.TXT** |
| const Error = $FFFF; | ACCEPT |
| const Eps = 1E-3; | ACCEPT |
| const OneHndNeg = -100; | ACCEPT |
| const Str = '0123456789abcdefgh'; | ACCEPT |

Продолжение таблицы 1

|  |  |
| --- | --- |
| const Hex = $GHIJ; | REJECT |
| const expression = -True; | REJECT |
| const abs: (-$3F9EA); | REJECT |
| const Real = 1.23.45; | REJECT |

# ПРОЕКТИРОВАНИЕ

В данном параграфе представлена информация о модульной структуре программы, содержащая описание интерфейсов между модулями. Кроме того, здесь представлена семантика модулей программы.

## Модульная структура

На рисунке 1 представлена модульная структура.

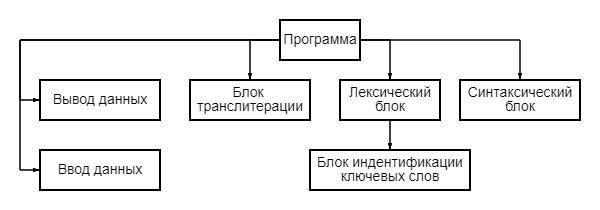


Рисунок 1 - Модульная структура

## Интерфейсы модулей

**Программа:** главный модуль, из которого происходит запуск программы и обработка входной цепочки.

Входные данные: -

Выходные данные: -

Модуль вызывает остальные подпрограммы.

**Ввод данных:** считывает данные из файла INPUT.txt для дальнейшей работы с ними.

Входные данные: -

Выходные данные: строка, считанная из файла INPUT.txt.

**Вывод данных:** записывает итоговый результат в файл OUTPUT.txt.

Входные данные: result – переменная, в зависимости от которой в файл output.txt выводится ACCEPT или REJECT.

Выходные данные: –

В зависимости от значения переменной result в файл OUTPUT.txt

записывается результат.

**Блок транслитерации:** подпрограмма, преобразующая исходную символьную цепочку в цепочку лексем вида («символ цепочки», «класс символа цепочки»).

Входные данные: string – строка, считанная из INPUT.txt.

Выходные данные: возвращает двумерный массив размера length \* 2 вида [«символ цепочки»] [«класс символа цепочки»], где length – длина исходной строки.

Например, символьную цепочку «const Error = $FFFF;» блок транслитерации должен преобразовать в двумерный массив лексем, пример представлен в таблице 2.

Таблица 2 – Пример массива блока транслитерации

|  |  |
| --- | --- |
| **Символ** | **Класс символа** |
| c | буква |
| o | буква |
| n | буква |
| s | буква |
| t | буква |
|  | пробел |
| E | буква |
| r | буква |
| r | буква |
| o | буква |
| r | буква |
|  | пробел |
| = | равно |
|  | пробел |
| $ | доллар |
| F | буква |
| F | буква |
| F | буква |
| F | буква |
| ; | тчзпт |

**Лексический блок:** подпрограмма, преобразующая цепочку лексем, полученную от транслитератора, в цепочку лексем вида («символ входного языка», «класс символа входного языка»).

Входные данные: двумерный массив, в котором находится строка, разбитая на символы с присвоенными им классами.

Выходные данные: двумерный массив размера length \* 2

Блок отвечает за преобразование двумерного массива лексем, полученного от блока транслитерации, в двумерный массив лексем вида [«слово входного языка»] [«класс слова входного языка»]. Например, лексический блок должен выдать цепочку лексем, описанную в таблице 3.

Таблица 3 – Пример массива лексического блока

|  |  |
| --- | --- |
| **Слово входного языка** | **Класс слова входного языка** |
| const | ИДЕНТ |
| Error | ИДЕНТ |
| = | равно |
| $ | доллар |
| FFFF | 16-РИЧ |
| ; | тчкзпт |

**Блок идентификации ключевых слов:** подпрограмма, которая устанавливает, какое из ключевых слов языка Pascal соответствует заданному идентификатору, либо сообщает, что заданный идентификатор не является ключевым словом языка Pascal.

Входные данные: words – двумерный массив, идентификаторы которого просматриваются на совпадения с ключевыми словами языка Паскаль.

Выходные данные: двумерный массив, идентификаторы, классы которых были изменены в случае совпадения с ключевыми словами языка Pascal. Блок устанавливает, присутствуют ли в двумерном массиве лексем ключевые слова языка Паскаль.

Ключевые слова: const

**Синтаксический блок:** подпрограмма, которая получает цепочку лексем вида («символ входного языка», «класс символа входного языка») и устанавливает, соответствует ли она заданным формулам Бэкуса-Наура.

Входные данные: string – двумерный массив, содержащий лексемы и их классы.

Выходные данные: если данная цепочка не совпадает с приведенными в подпрограмме примерами будет возвращено 0, иначе – 1.

# КОДИРОВАНИЕ

На данном этапе описан процесс кодирования: структура текста программы, алгоритмы реализации модулей, размер текста программы. Алгоритмы реализации модулей поделены на пять блоков: блок работы с файлом, блок транслитерации, лексический блок, блок идентификации и синтаксический блок. Программа написана на языке Python в программе PyCharm.

## Структура текста программы

На рисунке 2 представлена структура программы.

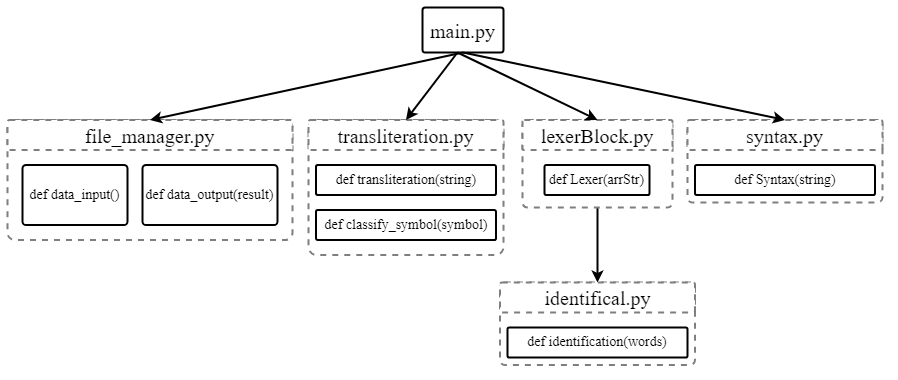


Рисунок 2 – Структура программы

main.py – головной модуль, содержащий функцию, по очереди вызывающую функции из каждого модуля.

file\_manager.py – модуль, содержащий функции def data\_input() и def data\_output(result). Функция def data\_input() считывает исходную цепочку из файла INPUT.txt, а функция data\_output(result) выводит в файл output.txt результат работы программы (ACCEPT или REJECT в зависимости от результата).

transliteration.py – модуль транслитерации, содержащий основную функцию, которая принимает символьную цепочку string в качестве входных данных, считываемую из файла input.txt.

lexerBlock.py – лексический модуль, содержащий функцию, получающую на вход список классов «символ/тип символа», состоящий из исходной строки, разбитой на символы и возвращающий список классов «символ/тип символа», состоящий из слов входного языка и их типов.

identifical.py – модуль идентификации ключевых слов. Функция получает на вход список лексем, состоящих из слов входного языка и их типов, проверяет, есть ли среди слов входного языка ключевые слова языка Pascal, и если они присутствуют, меняет класс этих слов на само слово.

syntax.py – синтаксический модуль, содержащий функцию, получающую на вход список лексем, состоящих из слов входного языка и их типов, проверенный на наличие ключевых слов, и возвращающий значение 0 или 1 в зависимости от корректности цепочки.

## Алгоритмы реализации модулей

В данном подразделе описаны детали реализации главных модулей программы.

Программа представляет собой автоматы с явным выделением состояний, ее компоненты подразделяются на события и входные переменные.

Совокупность конечного множества состояний и конечного множества входных воздействий образует автомат. Такой автомат реагирует на входные воздействия, определенным образом изменяя текущее состояние. Смена состояний, происходит с помощью переходов автомата. Состояния хранятся в переменной status.

### Блок транслитерации

Все символы входной строки обрабатываются и разбиваются по классам, представленным в таблице 4.

Таблица 4 – Таблица транслитерации

|  |  |
| --- | --- |
| a..z, A..Z | буква |
| 0..9 | цифра |
|  | пробел |
| ' | одинкавыч |
| +- | знак |
| $ | доллар |
| ; | тчкзпт |
| = | равно |
| <Другие> | ошибка |

### Лексический блок

В таблице 5 определены состояния конечного распознавателя лексического блока символьной цепочка.

Таблица 5 – Состояния конечного распознавателя лексического блока

|  |  |  |
| --- | --- | --- |
| **№** | **Состояние** | **Семантика** |
| 1 | start | Начальное состояние до начала обработки цепочки |
| 2 | firstWord | Обработка первого идентификатора |
| 3 | space1 | Обработка пробела после ключ. слова |
| 4 | ident | Обработка идентификатора после первого пробела |
| 5 | number | Обработка чисел |
| 6 | space2 | Обработка второго пробела после идентификатора |
| 7 | hexNum | Обработка 16-ричных констант. |
| 8 | quote1 | Обработка открывающей кавычки |
| 9 | quote2 | Обработка закрывающей кавычки |
| 10 | str | Обработка строки |
| 11 | expNum | Обработка числа в экспоненциальной записи |
| 12 | expNum2 | Проверка корректной записи экспоненциального числа |
| 13 | znak | Обработка знака «+» или «–» |
| 14 | equal | Обработка символа «=» |
| 15 | space3 | Обработка третьего пробела после знака равно |
| 16 | dollar | Обработка символа «$» |
| 17 | end | Обработка символа «;» |
| 18 | E | Ошибка |

В таблице 6 представлен конечный распознаватель лексического блока.

Таблица 6 – Конечный распознаватель лексического блока

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **буква** | **цифра** | **пробел** | **знак** | **'** | **=** | **$** | **«E»** | **;** |
| start | firstWord | E | E | E | E | E | E | E | E | 0 |
| firstWord | firstWord | ERR | space1 | E | E | E | E | E | E | 0 |
| space1 | ident | ident | E | E | E | E | E | E | E | 0 |
| ident | ident | ident | space2 | E | E | E | E | E | E | 0 |
| number | E | number | E | E | E | E | E | expNum | end | 0 |
| space2 | E | E | E | E | E | equal | E | E | E | 0 |
| hexNum | hexNum | hexNum | E | E | E | E | E | E | end | 1 |
| quoute1 | str | str | E | E | E | E | E | E | E | 0 |
| quoute2 | E | E | E | E | E | E | E | E | end | 0 |
| str | str | str | E | E | quote2 | E | E | E | E | 0 |
| expNum | E | expNum | E | expNum2 | E | E | E | E | end | 0 |
| expNum2 | E | expNum | E | E | E | E | E | E | E | 0 |
| znak | E | number | E | E | E | E | E | expNum | E | 0 |
| dollar | hexNum | hexNum | E | E | E | E | E | E | E | 1 |
| equal | E | E | space3 | E | E | E | E | E | E | 0 |
| space3 | E | number | E | znak | quote1 | E | dollar | E | E | 0 |
| end | E | E | E | E | E | E | E | E | end | 0 |
| E | E | E | E | E | E | E | E | E | E | 0 |

В полученном конечном распознавателе нет недостижимых состояний. Дальнейший процесс редукции конечного распознавателя лексического блока приведен в Таблице 7.

Таблица 7 - Редукция конечного распознавателя лексического блока

| **Шаг** | **Результат** | **Действие** |
| --- | --- | --- |
| 1 | P0 = {start, firstWord, number, hexNum, dollar, quote1, ident, quote2, str, expNum, expNum2, znak, equal, end, space1, space2, space3, end} | Разбиваем Р0 на два блока: допустимые и отвергающие состояния |
| 2 | P11 = {start, firstWord, number, quote1, ident, quote2, str, expNum, expNum2, znak, equal, end, space1, space2, space3}  P12 = {hexNum, dollar } | Разбиваем P11 и P12 по входу «;» |
| 3 | P21 = {start, firstWord, quote1, ident, str, expNum2, znak, equal, space1, space2, space3}  P22 = {dollar}  P23 = {hexNum, number, quote2, expNum, end} | Разбиваем P21 и P23 по входу «E» |
| 4 | P31 = {start, firstWord, quote1, ident, str, expNum2, equal, space1, space2, space3}  P32 = {dollar}  P33 = {hexNum, quote2, expNum}  P34 = {number, znak}  P35 = {end} | Разбиваем P31 по входу «пробел» |
| 5 | P41 = {start, quote1, str, expNum2, space1, space2, space3}  P42 = {dollar}  P43 = {hexNum, quote2, expNum}  P44 = {number, znak}  P45 = {end}  P46 = {firstWord, ident, equal } | Разбиваем P41 по входу «цифра» |
| 6 | P51 = {quote1, str, expNum2, space1, space3}  P52 = {dollar}  P53 = {hexNum, quote2, expNum}  P54 = {number, znak}  P55 = {end}  P56 = {firstWord, ident, equal }  P57 = {start, space2} | Разбиваем P51 по входу «$» |
| 7 | P61 = {quote1, str, expNum2, space1}  P62 = {dollar}  P63 = {hexNum, quote2, expNum}  P64 = {number, znak}  P65 = {end}  P66 = {firstWord, ident, equal }  P67 = {start, space2}  P68 = {space3} | Разбиваем P67 по входу «равно» |
| 8 | P61 = {quote1, str, expNum2, space1}  P62 = {dollar}  P63 = {hexNum, quote2, expNum}  P64 = {number, znak}  P65 = {end}  P66 = {firstWord, ident, equal }  P67 = {start}  P68 = {space3}  P69 = {space2} | Разбиваем P61 по входу «одинкавыч» |
| 9 | P71 = {quote1,expNum2, space1}  P72 = {dollar}  P73 = {hexNum, quote2, expNum}  P74 = {number, znak}  P75 = {end}  P76 = {firstWord, ident, equal }  P77 = {start}  P78 = {space3}  P79 = {space2}  P710 = {str} | Эквивалентными состояниями являются dollar и hexNum, далее они обозначаются как 16-РИЧ. |

Таблица 8 – Примитивные процедуры обрабатывающего автомата лексического блока

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Процедура** | **семантика** |
| 1 | Да | Остановить обработку и допустить цепочку |
| 2 | Нет | Остановить обработку и отвергнуть цепочку |
| 3 | Обработать | Добавить входной символ к значению текущей лексемы |
| 4 | Лексема (класс) | Увеличить счетчик лексем на 1, установить заданный класс текущей лексемы. |

Таблица 9 – Процедуры переходов обрабатывающего автомата лексического блока

| **Действие** | **Семантика** |
| --- | --- |
| ➊ | Обработать; |
| ➋ | Лексема(КЛСЛОВО\_);  Обработать; |
| ➌ | Лексема (ИДЕНТ)  Обработать; |
| ➍ | Лексема(ЦЕЛОЕ);  Обработать; |
| ➎ | Лексема(СТРКОНСТ);  Обработать; |
| ➏ | Лексема(ВЕЩКОНСТ);  Обработать; |
| ❼ | Лексема(16-РИЧ);  Обработать; |
| ❽ | Лексема(равно);  Обработать; |
| ❾ | Лексема(знак);  Обработать; |
| ❿ | Лексема(тчкзпт);  Обработать; |

Таблица 10 – Обрабатывающий автомат лексического блока

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | буква | цифра | + | - | ' | = | $ | «E» | ; |
| start | char  ➋ | number  ➋ | E | znak  ❾ | quote1  ➊ | equal  ➊ | dollar  ➊ | char  ➊ | E | 0 |
| char | char  ➊ | char  ➊ | start  ➌ | start  ➌ | start  ➌ | start  ➌ | start  ➌ | start  ➌ | start  ➌ | 0 |
| number | E | number  ➍ | E | E | E | E | E | expNum  ➏ | end  ➊ | 0 |
| hexNum | hexNum  ➏ | hexNum  ➏ | E | E | E | E | E | E | end  ➊ | 1 |
| quoute1 | str  ➊ | str  ➊ | E | E | E | E | E | E | E | 0 |
| quoute2 | E | E | E | E | E | E | E | E | end  ➊ | 1 |
| str | str  ➊ | str  ➊ | E | E | quote2  ➎ | E | E | E | E | 0 |
| expNum | E | expNum  ➊ | E | expNum2  ❾ | E | E | E | E | end  ➊ | 0 |
| expNum2 | E | expNum  ➏ | E | E | E | E | E | E | end  ➊ | 0 |
| znak | E | number  ➍ | E | E | E | E | E | expNum  ➏ | E | 0 |
| dollar | hexNum  ❼ | hexNum  ❼ | E | E | E | E | E | E | end  ➊ | 1 |
| equal | E | E | E | E | E | start  ❽ | E | E | E | 0 |
| end | E | E | E | E | E | E | E | E | end  ❿ | 1 |
| Error | E | E | E | E | E | E | E | E | E | 0 |

### Синтаксический блок

Составляется конечный распознаватель на основе исходных формул Бэкуса-Наура и лексем (КЛСЛОВО\_, ИДЕНТ, ЦЕЛОЕ, СТРКОНСТ, ВЕЩКОНСТ, 16-РИЧ, равно, знак, доллар, тчкзпт), полученных в Лексическом блоке и в Блоке идентификации ключевых слов.

Конечный распознаватель синтаксического блока рассматриваемого нами примера приведен в таблице 11.

Таблица 11 **–** Конечный распознаватель синтаксического блока

|  | КЛСЛОВО\_ | ИДЕНТ | ЦЕЛОЕ | СТРКОНСТ | ВЕЩКОНСТ | 16-РИЧ | знак | равно | тчкзпт |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| start | keyword |  |  |  |  |  |  |  |  |
| keyword |  | identification |  |  |  |  |  |  |  |
| identification |  |  |  |  |  |  |  | equal |  |
| equal |  |  | number | str | decimalNum | hexNum | znak |  |  |
| number |  |  |  |  |  |  | znak |  | end |
| znak |  |  |  |  | decimalNum |  |  |  |  |
| hexNum |  |  |  |  |  |  |  |  |  |
| str |  |  |  |  |  |  |  |  | end |
| decimalNum |  |  |  |  |  |  | znak |  | end |
| end |  |  |  |  |  |  |  |  |  |

Описанный выше автомат является минимальным. Пустым клеткам соответствует вызов примитивной процедуры НЕТ, которая отвергает цепочку.

### Блок идентификации ключевых слов

Ключевые слова языка Pascal хранятся в списке строк в алфавитном порядке. Для идентификации ключевых слов проводится линейный поиск по списку. Если идентификатор является ключевым словом, его класс меняется на это ключевое слово. Если в цепочке встречается более одного ключевого слова или любое ключевое слово, не являющееся обязательным для данной цепочки, то такая цепочка отвергается.

Ключевые слова: const, and, array, asm, begin, case, const, constructor, destructor, div, do, downto, else, end, fasle, file, for, function, goto, if, implementation, in, inherited, inline, interface, label, mod, nil, not, object, of, or, packed, procedure, program, record, reintroduce, repeat, self, set, shl, shr, string, then, to, true, type, unit, until, uses, var, while, with, xor

## Размер текста программы

Все модули программы суммарно занимают 289 строк.

Модуль lexerBlock содержит 147 строк.

Модуль syntax содержит 53 строку.

Модуль transliteration содержит 39 строку.

Модуль identifical содержит 22 строки.

Модуль file\_manager содержит 15 строк.

Модуль main содержит 13 строк.

# Тестирование

## Автономное тестирование

Далее описано тестирование автономное тестирование созданных модулей. В таблице приведены протоколы тестирования блока транслитерации.

Таблица 12 – Тестирование блока транслитерации

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Статус** |
| const Error = $FFFF; | [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('E', 'буква'), ('r', 'буква'), ('r', 'буква'), ('o', 'буква'), ('r', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('$', 'доллар'), ('F', 'буква'), ('F', 'буква'), ('F', 'буква'), ('F', 'буква'), (';', 'тчкзпт')] | Пройден |
| const Eps = -1E-3; | [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('E', 'буква'), ('p', 'буква'), ('s', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('-', 'знак'), ('1', 'цифра'), ('E', 'буква'), ('-', 'знак'), ('3', 'цифра'), (';', 'тчкзпт')] | Пройден |
| const OneHndNeg = -100; | ([('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('O', 'буква'), ('n', 'буква'), ('e', 'буква'), ('H', 'буква'), ('n', 'буква'), ('d', 'буква'), ('N', 'буква'), ('e', 'буква'), ('g', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('-', 'знак'), ('1', 'цифра'), ('0', 'цифра'), ('0', 'цифра'), (';', 'тчкзпт')] | Пройден |
| const Str = '0123456789abcdefgh'; | [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('S', 'буква'), ('t', 'буква'), ('r', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ("'", 'одинкавыч'), ('0', 'цифра'), ('1', 'цифра'), ('2', 'цифра'), ('3', 'цифра'), ('4', 'цифра'), ('5', 'цифра'), ('6', 'цифра'), ('7', 'цифра'), ('8', 'цифра'), ('9', 'цифра'), ('a', 'буква'), ('b', 'буква'), ('c', 'буква'), ('d', 'буква'), ('e', 'буква'), ('f', 'буква'), ('g', 'буква'), ('h', 'буква'), ("'", 'одинкавыч'), (';', 'тчкзпт')] | Пройден |
| const Str = ABCDefgh | [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('S', 'буква'), ('t', 'буква'), ('r', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('A', 'буква'), ('B', 'буква'), ('C', 'буква'), ('D', 'буква'), ('e', 'буква'), ('f', 'буква'), ('g', 'буква'), ('h', 'буква')] | Пройден |

Далее, в таблице 13 представлены протоколы тестирования лексического блока.

Таблица 13 – Тестирование лексического блока

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Статус** |
| [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('E', 'буква'), ('r', 'буква'), ('r', 'буква'), ('o', 'буква'), ('r', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('$', 'доллар'), ('F', 'буква'), ('F', 'буква'), ('F', 'буква'), ('F', 'буква'), (';', 'тчкзпт')] | [('const', 'ИДЕНТ'), ('Error', 'ИДЕНТ'), ('=', 'равно'), ('$', 'доллар'), ('FFFF', '16-РИЧ'), (';', 'тчкзпт')] | Пройден |
| [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('E', 'буква'), ('p', 'буква'), ('s', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('-', 'знак'), ('1', 'цифра'), ('E', 'буква'), ('-', 'знак'), ('3', 'цифра'), (';', 'тчкзпт')] | [('const', 'ИДЕНТ'), ('Eps', 'ИДЕНТ'), ('=', 'равно'), ('-1E-3', 'ВЕЩКОНСТ'), (';', 'тчкзпт')] | Пройден |
| ([('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('O', 'буква'), ('n', 'буква'), ('e', 'буква'), ('H', 'буква'), ('n', 'буква'), ('d', 'буква'), ('N', 'буква'), ('e', 'буква'), ('g', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('-', 'знак'), ('1', 'цифра'), ('0', 'цифра'), ('0', 'цифра'), (';', 'тчкзпт')] | [('const', 'ИДЕНТ'), ('OneHndNeg', 'ИДЕНТ'), ('=', 'равно'), ('-100', 'ЦЕЛОЕ'), (';', 'тчкзпт')] | Пройден |
| [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('S', 'буква'), ('t', 'буква'), ('r', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ("'", 'одинкавыч'), ('0', 'цифра'), ('1', 'цифра'), ('2', 'цифра'), ('3', 'цифра'), ('4', 'цифра'), ('5', 'цифра'), ('6', 'цифра'), ('7', 'цифра'), ('8', 'цифра'), ('9', 'цифра'), ('a', 'буква'), ('b', 'буква'), ('c', 'буква'), ('d', 'буква'), ('e', 'буква'), ('f', 'буква'), ('g', 'буква'), ('h', 'буква'), ("'", 'одинкавыч'), (';', 'тчкзпт')] | [('const', 'ИДЕНТ'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ("'0123456789abcdefgh'", 'СТРКОНСТ'), (';', 'тчкзпт')] | Пройден |
| [('c', 'буква'), ('o', 'буква'), ('n', 'буква'), ('s', 'буква'), ('t', 'буква'), (' ', 'пробел'), ('S', 'буква'), ('t', 'буква'), ('r', 'буква'), (' ', 'пробел'), ('=', 'равно'), (' ', 'пробел'), ('A', 'буква'), ('B', 'буква'), ('C', 'буква'), ('D', 'буква'), ('e', 'буква'), ('f', 'буква'), ('g', 'буква'), ('h', 'буква')] | [('const', 'ИДЕНТ'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ('ABCDefgh', 'ИДЕНТ')] | Пройден |

Далее, в таблице 14 представлены протоколы тестирования идентификационного блока ключевых слов.

Таблица 14 – Тестирование идентификационного блока

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Статус** |
| [('const', 'ИДЕНТ'), ('Error', 'ИДЕНТ'), ('=', 'равно'), ('$', 'доллар'), ('FFFF', '16-РИЧ'), (';', 'тчкзпт')] | [('const', 'КЛСЛОВО\_CONST'), ('Error', 'ИДЕНТ'), ('=', 'равно'), ('$FFFF', '16-РИЧ'), (';', 'тчкзпт')] | Пройден |
| [('const', 'ИДЕНТ'), ('Eps', 'ИДЕНТ'), ('=', 'равно'), ('-1E-3', 'ВЕЩКОНСТ'), (';', 'тчкзпт')] | [('const', 'КЛСЛОВО\_CONST'), ('Eps', 'ИДЕНТ'), ('=', 'равно'), ('-1E-3', 'ВЕЩКОНСТ'), (';', 'тчкзпт')] | Пройден |
| [('const', 'ИДЕНТ'), ('OneHndNeg', 'ИДЕНТ'), ('=', 'равно'), ('-100', 'ЦЕЛОЕ'), (';', 'тчкзпт')] | [('const', 'КЛСЛОВО\_CONST'), ('OneHndNeg', 'ИДЕНТ'), ('=', 'равно'), ('-100', 'ЦЕЛОЕ'), (';', 'тчкзпт')] | Пройден |
| [('const', 'ИДЕНТ'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ("'0123456789abcdefgh'", 'СТРКОНСТ'), (';', 'тчкзпт')] | [('const', 'КЛСЛОВО\_CONST'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ("'0123456789abcdefgh'", 'СТРКОНСТ'), (';', 'тчкзпт')] | Пройден |
| [('const', 'ИДЕНТ'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ('ABCDefgh', 'ИДЕНТ')] | [('const', 'КЛСЛОВО\_CONST'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ('ABCDefgh', 'ИДЕНТ')] | Пройден |

В таблице 15 представлены протоколы тестирования синтаксического блока.

Таблица 15 – Тестирование синтаксического блока

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Статус** |
| [('const', 'КЛСЛОВО\_CONST'), ('Error', 'ИДЕНТ'), ('=', 'равно'), ('$FFFF', '16-РИЧ'), (';', 'тчкзпт')] | 1 | Пройден |
| [('const', 'КЛСЛОВО\_CONST'), ('Eps', 'ИДЕНТ'), ('=', 'равно'), ('-1E-3', 'ВЕЩКОНСТ'), (';', 'тчкзпт')] | 1 | Пройден |
| [('const', 'КЛСЛОВО\_CONST'), ('OneHndNeg', 'ИДЕНТ'), ('=', 'равно'), ('-100', 'ЦЕЛОЕ'), (';', 'тчкзпт')] | 1 | Пройден |
| [('const', 'КЛСЛОВО\_CONST'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ("'0123456789abcdefgh'", 'СТРКОНСТ'), (';', 'тчкзпт')] | 1 | Пройден |
| [('const', 'КЛСЛОВО\_CONST'), ('Str', 'ИДЕНТ'), ('=', 'равно'), ('ABCDefgh', 'ИДЕНТ')] (';', 'тчкзпт')] | 0 | Пройден |

В таблице 16 представлены протоколы тестирования блока работы с файлом.

Таблица 16 – Тестирование блока работы с файлом

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Статус** |
| 1 | ACCEPT | Пройден |
| 1 | ACCEPT | Пройден |
| 1 | ACCEPT | Пройден |
| 1 | ACCEPT | Пройден |
| 0 | REJECT | Пройден |

## Комплексное тестирование

В таблице 19 приведен пример тестирования головного модуля программы.

**Таблица 19.** Тестирование головного модуля

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Статус** |
| const Error = $FFFF; | ACCEPT | Пройден |
| const hexNum = $3CEDB4019539E0; | ACCEPT | Пройден |
| const Eps = -1E-3; | ACCEPT | Пройден |
| const OneHndNeg = -100; | ACCEPT | Пройден |
| const Str = '0123456789abcdefgh'; | ACCEPT | Пройден |
| const Eps = 1E-3; | ACCEPT | Пройден |
| const num = $ABF33CF; | ACCEPT | Пройден |
| const Str = ABCDefgh | REJECT | Пройден |
| const Hex = $GHIJ; | REJECT | Пройден |
| const expression = -True; | REJECT | Пройден |
| const abs: (-$3F9EA); | REJECT | Пройден |
| const Real = 1.23.45; | REJECT | Пройден |
| const Eps = -1E-3 + $FFFF - 100; | REJECT | Пройден |
| const var = 'var'; | REJECT | Пройден |
| const function = 1E-; | REJECT | Пройден |

# Заключение

В заключение, учебная практика является неотъемлемой частью образовательного процесса, позволяющей студентам приобретать и развивать практические навыки и компетенции, необходимые для будущей профессиональной деятельности. В ходе моей учебной практики я получил ценный практический опыт, особенно в области распознавания цепочки, заданной формулами Бэкуса-Наура, работы с автоматами, алгоритмического проектирования и кодирования.

Каждый этап разработки программы – анализ, спецификация, проектирование, разработка алгоритмов, кодирование, тестирование и сопровождение – был осуществлен в рамках моей практики. В моем отчете подробно описана проделанная работа и ее результаты на каждом этапе, за исключением сопровождения.

Полученные знания и навыки я считаю ценными и применимыми в будущей профессиональной деятельности. Я уверен, что они будут полезны при создании синтаксических и лексических анализаторов, тестировании программного обеспечения на основе моделей и разработке искусственного интеллекта. Учебная практика дала мне возможность понять и применить теоретические знания на практике, а также развить навыки самостоятельной работы.

В целом, учебная практика была полезным и продуктивным периодом, который позволил мне приобрести практический опыт и подготовиться к будущей карьере в области разработки программного обеспечения.

# Литература

1. Йенсен К., Вирт Н. Паскаль. Руководство пользователя и описание языка. - М.: Компьютер, 1995.

2. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. - М.: Мир, 1979.