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 02.03.02 "Фундаментальная информатика   
и информационные технологии"

|  |  |
| --- | --- |
|  | Выполнил:\_\_\_\_\_\_\_\_\_\_\_\_ студент группы КЭ-102  Лежников М.В.  Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_  Ст. преподаватель кафедры СП  Силкина Н.С.  Дата:\_\_\_\_\_\_\_\_, Оценка:\_\_\_\_\_\_\_\_\_\_ |

Челябинск-2021

Министерство науки и высшего образования Российской Федерации

Южно-Уральский государственный университет

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.

**S <цепочка>::=<операторwhile-do>**

<условие>::=<логическая константа>

<оператор>::=<оператор if-then>

<условие>::=<идентификатор><операция сравнения><16-ричная константа>

<оператор1>::=<вызов подпрограммы>

<список параметров>::=<список идентификаторов>

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: "28" июня 2021 г.

Срок сдачи законченной работы: "25" июля 2021 г.

**Руководитель:**

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

должность, ученая степень подпись ФИО руководителя

**Задание принял к исполнению:**

\_\_\_\_\_\_\_\_\_\_\_\_\_ Лежников М.В.

подпись ФИО студента

**ОГЛАВЛЕНИЕ**

[1. Спецификация 1](#_Toc327919175)

[2. Проектирование 1](#_Toc327919176)

[2.1. Модульная структура 2](#_Toc327919177)

[2.2. Интерфейсы модулей 2](#_Toc327919178)

[3. Кодирование 3](#_Toc327919179)

[3.1. Структура текста программы 3](#_Toc327919180)

[3.2. Алгоритмы реализации модулей 5](#_Toc327919181)

[3.2.1. Блок транслитерации 5](#_Toc327919182)

[3.2.2. Лексический блок 5](#_Toc327919183)

[3.2.3. Синтаксический блок 7](#_Toc327919184)

[3.2.4. Блок идентификации ключевых слов 8](#_Toc327919185)

[3.3. Размер текста программы (в строках) 8](#_Toc327919186)

[4. Тестирование 8](#_Toc327919187)

[4.1. Автономное тестирование 8](#_Toc327919188)

[4.2. Комплексное тестирование 10](#_Toc327919189)

[Заключение 10](#_Toc327919190)

[Литература 10](#_Toc327919191)

# Спецификация

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

**S <цепочка>::=<оператор while-do>**

<оператор while-do>::=WHILE *<условие>* DO *<оператор>*;

<условие>::=<логическая константа>

<логическая константа>::=**TRUE** | **FALSE**

<оператор>::=<оператор if-then>

<оператор if-then>::=**IF** *<условие>* **THEN** *<оператор1>***;**

<условие>::=<идентификатор><операция сравнения><16-ричная константа>

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

<операция сравнения>::=**>** | **<** | **>=** | **<=** | **<>**

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

<оператор1>::=<вызов подпрограммы>

<список параметров>::=<список идентификаторов>

**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**

|  |  |
| --- | --- |
| INPUT.TXT | OUTPUT.TXT |
| while false do if A>$a then p(i); | ACCEPT |
| while true do if MyVar<>$FFFF then Snooze(param); | ACCEPT |
| while True do if Check<>$1F then Kill(p1, p2, p3, p4); | ACCEPT |
| while false do if B<$C then Greed(i); | ACCEPT |
| while True do if Check<>$1F then Kill(p1, p2, p3,) | REJECT |
| while false do if B\*$a then p(i); | REJECT |
| while True do if Check<>$1F then Kill(p1, p2, p3, p4); | REJECT |
| while Greed do if MyVar<>$FFFF then Snooze(param); | REJECT |
| while True do if Check-$1F then Kill(p1, p2, p3, p4); | REJECT |
| while True do if Check-$1F; | REJECT |

# Проектирование

Данный раздел следует разбить на следующие пункты:

## Модульная структура

В данном пункте следует поместить *рисунок со схемой модульной структуры* с кратким описанием назначения входящих в нее модулей**.**

***Модульная структура*** представляет собой иерархию процедур и функций (называемых *модулями*), с помощью которых программа решает поставленную задачу. При этом программа является *головным* модулем в данной иерархии.

Пример модульной структуры некоторой программы приведен на Рис. 1.

Программа

(головной модуль)

Блок идентификации ключевых слов

Лексический блок

Блок транслитерации

Синтаксический Блок

**Рис. 1.**

## Интерфейсы модулей

***Блок транслитерации***– подпрограмма, преобразующая исходную символьную цепочку в цепочку лексем вида ("символ цепочки", "класс символа цепочки ").

Например, символьную цепочку

const N=10; 6блок транслитерации должен преобразовать в цепочку лексем

(c,*буква*), (o,*буква*), (n,*буква*), (s,*буква*), (t,*буква*),(' ',*пробел*), (N,*буква*), (=,*равно*), (1,*цифра*), (0,*цифра*),(;,*тчкзпт*)

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

В рассматриваемом примере лексический блок должен выдать следующую цепочку лексем:(const, ИДЕНТИФИКАТОР),(N, ИДЕНТИФИКТОР),(=,РАВНО),

(10,ЦЕЛОЕ), (;,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. Идентификация ключевых слов может быть выделена в *отдельный проход* распознавателя символьной цепочки, то есть идентифицирующий блок будет запускаться *после* того, как лексический блок полностью подготовит *всю* цепочку лексем. Другим подходом является *объединение в один проход* распознавателя символьной цепочки лексического блока и блока идентификации. В этом случае идентификация ключевого слова осуществляется *каждый раз*, когда лексический блок выдал *одну* лексему класса "идентификатор" (без ожидания полной подготовки всей цепочки лексем).

В рассматриваемом примере после распознавания слов цепочка лексем, полученная в результате работы лексического блока, примет следующий вид:

(const,КЛСЛОВО\_CONST), (N,ИДЕНТИФИКАТОР), (=,РАВНО),

(10,ЦЕЛОЕ), (;,ТЧКЗПТ)

***Синтаксический блок***– подпрограмма, которая получает цепочку лексем вида ("символ входного языка", "класс символа входного языка") и устанавливает, соответствует ли она заданным формулам Бэкуса-Наура. Поскольку задание предполагает только распознавание исходной цепочки, фактически данный блок использует для работы только классы символов входного языка.

В рассматриваемом примере синтаксический блок, рассматривая цепочку вида

КЛСЛОВО\_CONST, ИДЕНТИФИКАТОР, РАВНО, ЦЕЛОЕ, ТЧКЗПТ

должен сообщить, что она синтаксически правильна.

# Кодирование

Данный раздел следует разбить на следующие пункты:

## Структура текста программы

В данном пункте следует поместить *рисунок со схемой структуры текста программы* с кратким описанием назначения входящих в нее модулей unit и экспортируемых данными модулями unit подпрограмм.

main.py

(Головной модуль)

Translitter.py

(Блок транслитерации)

Vocabular.py

(Лексический блок)

Syntax.py

(Синтаксический блок

)

Identification.py (Блок идентификации ключевых слов )

|  |  |  |
| --- | --- | --- |
| **Модуль** | **Тесты** | |
| **Входные данные** | **Выходные данные** |
| Блок транслитерации | Dz12;= и | (D,*буква*),  (z,*буква*),  (1,*цифра*),  (2,*цифра*),  (;,*тчкзпт*),  (=,*равно*),  ( ,*пробел*),  (и,*ошибка*) |
| Лексический блок | (D,СИМ\_БУКВА),  (z,СИМ\_БУКВА),  (1,СИМ\_ЦИФРА),  (2,СИМ\_ЦИФРА),  ( ,СИМ\_ПРОБЕЛ),  (1,СИМ\_ЦИФРА),  (2,СИМ\_ЦИФРА),  (;,СИМ\_ТЧКЗПТ),  (=,СИМ\_РАВНО) | (Dz12,ИДЕНТИФИКАТОР),  (12,ЦЕЛОЕ),  (;,ТЧКЗПТ),  (=,РАВНО) |
| Блок идентификации ключевых слов | Dz12 | НЕ\_КЛСЛОВО |
| const | КЛСЛОВО\_CONST |
| Синтаксический блок | ИДЕНТИФИКАТОР,  ЦЕЛОЕ,  ТЧКЗПТ РАВНО | ОШИБКА |
| КЛСЛОВО\_CONST,  ИДЕНТИФИКАТОР,  РАВНО, ЦЕЛОЕ | ОШИБКА |
| КЛСЛОВО\_CONST,  ИДЕНТИФИКАТОР,  РАВНО,  ЦЕЛОЕ, ТЧКЗПТ | ПРАВИЛЬНО |
| Головной модуль | const Dz=10; | ACCEPT |

**main.py** (Головной модуль)– Модуль, в которой вызываются все модули, считываются данные из файла INPUT.TXT , последовательно выполняются подпрограммы и выводятся в файл OUTPUT.TXT.

**Translitter.py (**Блок транслитерации) – Модуль , в который поступает последовательность, делится посимвольно, каждый символ распределяется в нужный класс лексем , выводится список с определением каждого символа.

**Sintax.py (**Синтаксический Блок)– модуль , отвечает за правильное расположение символов, получает на вход цепочку, устанавливает отвечает ли требованиям формул Бэкуса-Наура, которые были заданы. В конце сообщает правильно или нет(ПРАВИЛЬНО, ОШИБКА).

**Vocabular.py (**Лексический блок) – модуль, преобразующий цепочку лексем, полученную от транслитератора, в цепочку лексем вида, на выходе выводит цепочку лексем.

**Identification.py** (Блок идентификации ключевых слов)– модуль, который устанавливает, какое из ключевых слов языка Pascal соответствует заданному идентификатору, либо сообщает, что за-данный идентификатор не является ключевым словом языка Pascal. Идентификация ключевых слов может быть выделена в отдельный проход распознавателя символьной цепочки, то есть идентифицирующий блок будет запускаться после того, как лексический блок полностью подготовит всю цепочку лексем. Другим подходом является объединение в один проход распознавателя символьной цепочки лексического блока и блока идентификации. В этом случае идентификация ключевого слова осуществляется каждый раз, когда лексический блок вы-дал одну лексему класса "идентификатор" .

## Алгоритмы реализации модулей

Данный пункт следует разбить на следующие подпункты:

### Блок транслитерации

**Транслитерация символьной цепочки**

|  |  |
| --- | --- |
| **Символы** | **Класс лексемы** |
| a..z, A..Z | буква |
| 0..9 | цифра |
| = | равно |
| ; | тчкзпт |
| +,- | знак |
| пробел | пробел |
| Другие символы | ошибка |

### Лексический блок

Лексический блок необходимо реализовать в виде детерминированного конечного автомата, который выполняет обработку и распознавание входной цепочки лексем (полученной как результат работы блока транслитерации). *Обработка* входной цепочки заключается в формировании цепочки лексем вида ("символ входного языка", "класс символа входного языка"). *Распознавание* входной цепочки означает, что данный автомат должен отвергать все цепочки лексем, содержащие лексемы, которые заведомо не удовлетворяют формулам Бэкуса-Наура из условия задачи (например, два знака числа подряд, знак числа после десятичной точки и др.).

Построение конечного автомата лексического блока необходимо выполнить следующим образом. Сначала, используя метод разметки символов [3], нужно построить *конечный распознаватель* входной цепочки лексем: определить список состояний (название/номер и семантика состояния), выделить в данном списке начальное состояние и допускающие состояния и указать функцию переходов.

Далее необходимо выполнить *редукцию* построенного автомата (или доказать, что построенный автомат является минимальным). Редукция автомата осуществляется следующим образом [3]. Сначала необходимо найти и отбросить недостижимые состояния построенного автомата. Затем необходимо найти группы эквивалентных состояний и для всех групп заменить каждое состояние в найденной группе на одно состояние. Поиск эквивалентных состояний наиболее эффективно проводить методом разбиения [3].

В рассматриваемом нами примере у конечного распознавателя лексического блока нет недостижимых состояний. Состояния РАВНО и ПРОБЕЛ3, ЗНАК и ПРОБЕЛ4, ТЧКЗПТ и ПРОБЕЛ6 попарно идентичны, а следовательно, эквиваленты.

**Конечный распознаватель**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **буква** | **пробел** | **цифра** | **тчкзпт** | **равно** | **Откр. скобка** | **Закр. скобка** | **больше** | **меньше** | **Шест. знак** | **запятая** |
| **Нач** | Идент. | Нач | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Клслово\_wh** | Идент. | Пробел1 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел1** | Логконст | Пробел1 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Логконст** | Логконст | Пробел2 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел2** | Идент. | Пробел2 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Клслово\_do** | Идент. | Пробел3 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел3** | Идент. | Пробел3 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Клслово\_if** | Идент. | Пробел4 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел4** | Идент. | Пробел4 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **имя** | Идент. | Пробел5 | Идент. | Е | Е | Е | Е | Оп.срав. | Оп.срав. | Е | Е |
| **Пробел5** | E | Пробел5 | Е | Е | Е | Е | Е | Оп.срав. | Оп.срав. | Е | Е |
| **сравнение** | E | Пробел6 | Е | Е | Оп.срав. | Е | Е | Оп.срав. | Е | Шест.  Конст | Е |
| **Пробел6** | Е | Пробел6 | Е | Е | Е | Е | Е | Е | Е | Шест.  Конст | Е |
| **Шест-ная** | Шест.  Конст. | Пробел7 | Шест. | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел7** | Идент. | Пробел7 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Клслово\_then** | Идент | Пробел8 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел8** | Идент | Пробел8 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Клслово\_**  **Вызов\_**  **подпрограммы** | Идент | Пробел9 | Идент. | Е | Е | Отк.скобка | Е | Е | Е | Е | Е |
| **Пробел9** | Е | Пробел9 | Е | Е | Е | Отк.скобка | Е | Е | Е | Е | Е |
| **Отк. скобка** | Е | Пробел10 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел10** | Идент | Пробел10 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **параметр** | Идент | Пробел11 | Идент. | Е | Е | Е | Зак. скобка | Е | Е | Е | запятая |
| **Пробел11** | Е | Пробел11 | Е | Е | Е | Е | Е | Е | Е | Е | запятая |
| **запятая** | Е | Пробел12 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел12** | Е | Пробел12 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел13** | Е | Пробел13 | Е | Е | Е | Е | Зак. скобка | Е | Е | Е | Е |
| **Зак. скобка** | Е | Пробел14 | Е | тчкзпт | Е | Е | Е | Е | Е | Е | Е |
| **Пробел14** | Е | Пробел14 | Е | тчкзпт | Е | Е | Е | Е | Е | Е | Е |
| **тчкзпт** | Е | Пробел15 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Пробел15** | Е | Пробел15 | Е | Е | Е | Е | Е | Е | Е | Е | Е |
| **Ошибка** | Е | Е | Е | Е | Е | Е | Е | Е | Е | Е | Е |

**Состояния конечного распознавателя лексического блока**

|  |  |  |
| --- | --- | --- |
| **№** | **Состояние** | **семантика** |
|  | **Нач** | **Момент до начала обработки цепочки либо чтение пробелов в начале строки** |
|  | **Клслово\_wh** | **Чтение ключевого слова while** |
|  | **Пробел1** | **Чтение пробелов, находищихся между ключевым словом while и логической константой** |
|  | **Логконст** | **Чтение логической константы** |
|  | **Пробел2** | **Чтение пробелов находящихся между логической константой и ключ. словом do** |
|  | **Клслово\_do** | **Чтение ключевого слова do** |
|  | **Пробел3** | **чтение пробелов находищихся между кл. словом do и кл. словом if** |
|  | **Клслово\_if** | **Чтение ключевого слова if** |
|  | **Пробел4** | **Чтение пробелов находящихся между кл. словом if и именем переменной** |
|  | **имя** | **Чтение имени переменной** |
|  | **Пробел5** | **Чтение пробелов между именем константы и оператором сравнения** |
|  | **сравнение** | **Чтение оператора сравнения** |
|  | **Пробел6** | **Чтение пробелов между оператором сравнения и шеснадцатиничной константой** |
|  | **Шест-ная** | **Чтение шеснадцатиричной константы** |
|  | **Пробел7** | **Чтение пробелов между шеснадцатиричной константой и кл. словом then** |
|  | **Клслово\_then** | **Чтение кл. слова then** |
|  | **Пробел8** | **Чтение пробелов находящихся между кл. словом then и вызовом подпрограммы** |
|  | **Клслово\_**  **Вызов\_**  **подпрограммы** | **Чтение кл. слова вызова подпрограммы** |
|  | **Пробел9** | **Чтение пробелов находящихся между кл. словом вызова подпрограммы и открывающей скобкой** |
|  | **Отк. скобка** | **Чтение открывающей скобки** |
|  | **Пробел10** | **Чтение пробелов между открывающей скобкой и параметром** |
|  | **параметр** | **Чтение кл. слова параметра** |
|  | **Пробел11** | **Чтение пробелов между параметром и запятой** |
|  | **запятая** | **Чтение запятой** |
|  | **Пробел12** | **Чтение пробелов находящихся между запятой и параметром** |
|  | **Пробел13** | **Чтение пробелов находящихся между параметром и закрывающей скобкой** |
|  | **Зак. скобка** | **Чтение закрывающей скобки** |
|  | **Пробел14** | **Чтение пробелов находящихся между закрывающей скобки и точки с запятой** |
|  | **тчкзпт** | **Чтение точки с запятой** |
|  | **Пробел15** | **Чтение пробелов в конце строки** |
|  | **Е** | **Ошибка** |

Затем полученный минимальный конечный распознаватель необходимо преобразовать в *обрабатывающий автомат*. Алфавит распознавателя следует дополнить символом "концевой маркер", а переходы в состояние "ошибка" заменить на вызов некоторой примитивной процедуры "отвергнуть", которая останавливает обработку и отвергает входную цепочку. Остальные переходы также следует дополнить вызовами некоторых примитивных процедур, которые обеспечивают формирование выходной цепочки лексем (например, "допустить", "сформировать лексему" и др.).

### Синтаксический блок

*Синтаксический блок* необходимо реализовать в виде детерминированного конечного распознавателя. Построенный автомат необходимо подвергнуть редукции способом, аналогичным приведенному выше в указаниях к разработке алгоритмов для лексического блока.

### Блок идентификации ключевых слов

Ключевые слова идентифицируются с помощью проверки на нахождение слова в списке ключевых слов, взятых из документации проекта FreePascal [1] и документа Pascal ISO 7185:1990 [2].

## Размер текста программы (в строках)

Размер текста исходного кода в строках main – 20, StringCharTypes – 34, TransliterationBlock – 11, FinalRecoginzerStates – 75, InputLanguageCharClass – 44, LexicalBlock2 –35, MinimalFinalRecognizer– 124, KeywordIdentificationBlock –21, KeyWords –7, AnalysisResults – 11, SyntaxBlock – 26.

# Тестирование

Данный раздел следует разбить на следующие пункты:

## Автономное тестирование

|  |  |  |
| --- | --- | --- |
| **Модуль** | **Тесты** | |
| **входные** | **выходные** |
| **Блок транслитерации** | while True do if Check<>$1F then Kill(p1, p2, p3); | [('w', 'буква'), ('h', 'буква'), ('i', 'буква'), ('l', 'буква'), ('e', 'буква'), (' ', 'пробел'), ('T', 'буква'), ('r', 'буква'), ('u', 'буква'), ('e', 'буква'), (' ', 'пробел'), ('d', 'буква'), ('o', 'буква'), (' ', 'пробел'), ('i', 'буква'), ('f', 'буква'), (' ', 'пробел'), ('C', 'буква'), ('h', 'буква'), ('e', 'буква'), ('c', 'буква'), ('k', 'буква'), ('<', 'меньше'), ('>', 'больше'), ('$', 'шестнадцатиричный знак'), ('1', 'цифра'), ('F', 'буква'), (' ', 'пробел'), ('t', 'буква'), ('h', 'буква'), ('e', 'буква'), ('n', 'буква'), (' ', 'пробел'), ('K', 'буква'), ('i', 'буква'), ('l', 'буква'), ('l', 'буква'), ('(', 'открывающая скобка'), ('p', 'буква'), ('1', 'цифра'), (',', 'запятая'), (' ', 'пробел'), ('p', 'буква'), ('2', 'цифра'), (',', 'запятая'), (' ', 'пробел'), ('p', 'буква'), ('3', 'цифра'), (',', 'запятая'), (')', 'закрывающая скобка'), (';', 'тчкзпт')] |
| **Лексический блок** | [('w', 'буква'), ('h', 'буква'), ('i', 'буква'), ('l', 'буква'), ('e', 'буква'), (' ', 'пробел'), ('T', 'буква'), ('r', 'буква'), ('u', 'буква'), ('e', 'буква'), (' ', 'пробел'), ('d', 'буква'), ('o', 'буква'), (' ', 'пробел'), ('i', 'буква'), ('f', 'буква'), (' ', 'пробел'), ('C', 'буква'), ('h', 'буква'), ('e', 'буква'), ('c', 'буква'), ('k', 'буква'), ('<', 'меньше'), ('>', 'больше'), ('$', 'шестнадцатиричный знак'), ('1', 'цифра'), ('F', 'буква'), (' ', 'пробел'), ('t', 'буква'), ('h', 'буква'), ('e', 'буква'), ('n', 'буква'), (' ', 'пробел'), ('K', 'буква'), ('i', 'буква'), ('l', 'буква'), ('l', 'буква'), ('(', 'открывающая скобка'), ('p', 'буква'), ('1', 'цифра'), (',', 'запятая'), (' ', 'пробел'), ('p', 'буква'), ('2', 'цифра'), (',', 'запятая'), (' ', 'пробел'), ('p', 'буква'), ('3', 'цифра'), (',', 'запятая'), (')', 'закрывающая скобка'), (';', 'тчкзпт')] | ('while', 'ИДЕНТИФИКАТОР'), ('True', 'ЛОГКОНСТ'), ('do', 'ИДЕНТИФИКАТОР'), ('if', 'ИДЕНТИФИКАТОР'), ('Check', 'ИДЕНТИФИКАТОР'), ('<>', 'ОПЕРАТОР СРАВНЕНИЯ'), ('$1F', 'ШЕСНАДЦАТИРИЧНАЯ КОНСТАНТА'), ('then', 'ИДЕНТИФИКАТОР'), ('Kill', 'ИДЕНТИФИКАТОР'), ('(', 'ОТКРЫВАЮЩАЯ СКОБКА'), ('p1', 'ИДЕНТИФИКАТОР'), (',', 'ЗАПЯТАЯ'), ('p2', 'ИДЕНТИФИКАТОР'), (',', 'ЗАПЯТАЯ'), ('p3', 'ИДЕНТИФИКАТОР'), (',', 'ЗАПЯТАЯ'), (')', 'ЗАКРЫВАЮЩАЯ СКОБКА'), (';', 'ТЧКЗПТ') |
| **Идентефикация ключевых слов** | ('while', 'ИДЕНТИФИКАТОР'), ('True', 'ЛОГКОНСТ'), ('do', 'ИДЕНТИФИКАТОР'), ('if', 'ИДЕНТИФИКАТОР'), ('Check', 'ИДЕНТИФИКАТОР'), ('<>', 'ОПЕРАТОР СРАВНЕНИЯ'), ('$1F', 'ШЕСНАДЦАТИРИЧНАЯ КОНСТАНТА'), ('then', 'ИДЕНТИФИКАТОР'), ('Kill', 'ИДЕНТИФИКАТОР'), ('(', 'ОТКРЫВАЮЩАЯ СКОБКА'), ('p1', 'ИДЕНТИФИКАТОР'), (',', 'ЗАПЯТАЯ'), ('p2', 'ИДЕНТИФИКАТОР'), (',', 'ЗАПЯТАЯ'), ('p3', 'ИДЕНТИФИКАТОР'), (',', 'ЗАПЯТАЯ'), (')', 'ЗАКРЫВАЮЩАЯ СКОБКА'), (';', 'ТЧКЗПТ') | 'КЛСЛОВО\_WHILE', 'ЛОГКОНСТ', 'КЛСЛОВО\_DO', 'КЛСЛОВО\_IF', 'ИДЕНТИФИКАТОР', 'ОПЕРАТОР СРАВНЕНИЯ', 'ШЕСНАДЦАТИРИЧНАЯ КОНСТАНТА', 'КЛСЛОВО*THEN', 'КЛСЛОВО*ВЫЗОВ*ПОДПРОГРАММЫ', 'ОТКРЫВАЮЩАЯ СКОБКА', 'КЛСЛОВО*ПАРАМЕТР', 'ЗАПЯТАЯ', 'КЛСЛОВО*ПАРАМЕТР', 'ЗАПЯТАЯ', 'КЛСЛОВО*ПАРАМЕТР', 'ЗАПЯТАЯ', 'ЗАКРЫВАЮЩАЯ СКОБКА', 'ТЧКЗПТ' |
| **Синтаксический блок** | 'КЛСЛОВО\_WHILE', 'ЛОГКОНСТ', 'КЛСЛОВО\_DO', 'КЛСЛОВО\_IF', 'ИДЕНТИФИКАТОР', 'ОПЕРАТОР СРАВНЕНИЯ', 'ШЕСНАДЦАТИРИЧНАЯ КОНСТАНТА', 'КЛСЛОВО*THEN', 'КЛСЛОВО*ВЫЗОВ*ПОДПРОГРАММЫ', 'ОТКРЫВАЮЩАЯ СКОБКА', 'КЛСЛОВО*ПАРАМЕТР', 'ЗАПЯТАЯ', 'КЛСЛОВО*ПАРАМЕТР', 'ЗАПЯТАЯ', 'КЛСЛОВО*ПАРАМЕТР', 'ЗАПЯТАЯ', 'ЗАКРЫВАЮЩАЯ СКОБКА', 'ТЧКЗПТ' | ACCEPT |

## Комплексное тестирование

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | входные | выходные | результат | Тест пройден? |
| 1 | while false do if A>$a then p(i); | ACCEPT | ACCEPT | Да |
| 2 | while true do if MyVar<>$FFFF then Snooze(param); | ACCEPT | ACCEPT | Да |
| 3 | while True do if Check<>$1F then Kill(p1, p2, p3, p4); | ACCEPT | ACCEPT | Да |
| 4 | while false do if B<$C then Greed(i); | ACCEPT | ACCEPT | Да |
| 5 | while True do if Check<>$1F then Kill(p1, p2, p3,) | REJECT | REJECT | Да |
| 6 | while false do if B\*$a then p(i); | REJECT | REJECT | Да |
| 7 | while True do if Check<>$1F then Kill(p1, p2, p3, p4); | REJECT | REJECT | Да |
| 8 | while Greed do if MyVar<>$FFFF then Snooze(param); | REJECT | REJECT | Да |
| 9 | while True do if Check-$1F then Kill(p1, p2, p3, p4); | REJECT | REJECT | Да |
| 10 | while True do if Check-$1F; | REJECT | REJECT | Да |

# Заключение

В ходе практической работы был реализован распознаватель символьной цепочки, заданной с помощью формул Бэкуса-Наура. Работа была разделена на несколько этапов: реализация блока транслитерации, лексического блока, блока идентификации ключевых слов, синтаксического блока. За время работы были пройдены данные циклы разработки: анализ, спецификация, проектирование, кодирование и тестирование..

# 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.
3. Free Pascal Reserved words. URL: https://wiki.freepascal.org/Reserved\_words (дата обращения: 29.06.21)
4. Pascal ISO 7185:199. Online copy of the unextended Pascal standard. URL: https://www.pascal-central.com/docs/iso7185.pd