МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

**"Южно-Уральский государственный университет"**

**(национальный исследовательский университет)**

**Факультет Вычислительной математики и информатики**

**Кафедра системного программирования**

**ОТЧЕТ   
по учебной практике**

бакалавра направления 09.03.02 "Фундаментальная информатика и информационные технологии"

|  |  |
| --- | --- |
|  | Выполнил:  студент группы ВМИ-105  А.Р.Ишмурзин  Проверил:  К.Ю. Никольская  старший преподаватель |

Челябинск-2016

Министерство образования и науки 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.

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: "27" июня 2016 г.

Срок сдачи законченной работы: "24" июля 2016 г.

**Руководитель:**

старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ К.Ю. Никольская

подпись

**Задание принял к исполнению:**

\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Р.Ишмурзин

подпись

**ОГЛАВЛЕНИЕ**

[1. Спецификация 4](#_Toc327919175)

[2. Проектирование 4](#_Toc327919176)

[2.1. Модульная структура 5](#_Toc327919177)

[2.2. Интерфейсы модулей 5](#_Toc327919178)

[3. Кодирование 7](#_Toc327919179)

[3.1. Структура текста программы 7](#_Toc327919180)

[3.2. Алгоритмы реализации модулей 7](#_Toc327919181)

[3.2.1. Блок транслитерации 7](#_Toc327919182)

[3.2.2. Лексический блок 8](#_Toc327919183)

[3.2.3. Синтаксический блок 12](#_Toc327919184)

[3.2.4. Блок идентификации ключевых слов 13](#_Toc327919185)

[3.3. Размер текста программы (в строках) 14](#_Toc327919186)

[4. Тестирование 14](#_Toc327919187)

[4.1. Автономное тестирование 14](#_Toc327919188)

[4.2. Комплексное тестирование 15](#_Toc327919189)

[Заключение 15](#_Toc327919190)

[Литература 15](#_Toc327919191)

# Спецификация

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

<цепочка>::=<оператор repeat-until>

<условие>::=<вызов подпрограммы>

<вызов подпрограммы>::=<идентификатор>**(** <список параметров> **)**;

<список параметров>::=<список идентификаторов>

<список идентификаторов>::=<идентификатор> |

<идентификатор> , <список идентификаторов>

<идентификатор>::=<буква>|

<идентификатор><буква>|

<идентификатор><цифра>

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

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

<список параметров>::=<список идентификаторов>

<оператор>::=<вызов подпрограммы>

<список параметров>::=<элемент массива>

<элемент массива>::=<идентификатор>**[** <индексное выражение> **]**

<индексное выражение>::=<целая константа>

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

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

<знак>::= **+ | -**

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

<список параметров>::=<целая константа>

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

Помимо этого на цепочку накладывается следующее семантическое ограничение:

идентификатор, входящий в цепочку, не должен совпадать с ключевыми словами языка Pascal .

**Описание входных данных**

Цепочка записана в текстовом файле INPUT.TXT, который состоит из одной стро-ки. 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** |
| repeat myproc(1) until myfunc(a); | ACCEPT |
| repeat myproc(+16) until myfunc(param1, param2); | ACCEPT |
| repeat myproc(-123) until myfunc(p1, p2, p3, p4); | ACCEPT |
| repeat begin(-123) until myfunc(p1, p2, p3, p4); | REJECT |
| repeat myproc(-123) until myfunc(p1, p2, p3, p4) | REJECT |
| repeat myproc(-123 par) until myfunc(p1, p2, p3, p4); | REJECT |
| repeat myproc(-123) until: myfunc(p1, p2, p3, p4); | REJECT |

# Проектирование

В данном разделе содержатся два подраздела: Модульная структура и Интерфейсы модулей. Вместе они описывают этап проектирования программы, до начала кодирования.

# Модульная структура

Программа

Синтаксический блок

**Блок транслитерации**

**Ввод/вывод данных**

**Лексический блок**

**Блок идентификации ключевых слов**

**Посимвольное разбиение строки и присваивание класса символа**

**Перевод символов в нижний регистр**

*Рисунок 1 – схема модульной структуры программы*

**Ввод/вывод данных -** ввода и вывода данных.

**Блок транслитерации** – Преобразование исходной символьной цепочки в двумерный вектор лексем вида ["символ цепочки"]["класс символа цепочки "]

**Перевод символов в нижний регистр** – Функция перевода всех символов строки в нижний регистр.

**Посимвольное разбиение строки** – Функция посимвольного разбиения строки, присваивающая символу его класс.

**Лексический блок** – Преобразование двумерного вектора лексем, полученного от блока транслитерации, в двумерный вектор лексем вида ["символ входного языка"]["класс символа входного языка"].

**Блок идентификации ключевых слов** – Блок, устанавливающий присутствуют ли в двумерном векторе лексем ключевые слова языка Паскаль.

**Перевод символов в верхний регистр** – Функция перевода названия ключевого слова в верхний регистр.

**Синтаксический блок** – Блок, устанавливающий соответствует ли вектор заданным формулам Бэкуса-Наура.

## 2.2. Интерфейсы модулей

Ввод данных:

Семантика: **string input(bool err);**

Входные данные: bool err – состояние ошибки. Если на вход подана пустая строка, будет выведено сообщение: "runtime error: empty string".

Выходные данные: string s – строка, считанная из файла input.txt.

Модуль возвращает строку из файла input.txt.

Вывод данных:

Семантика: **void output(bool result);**

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

Выходные данные: –

Модуль в зависимости от значения булевой переменной result записывает в файл output.txt либо ACCEPT, либо REJECT.

Блок транслитерации:

Семантика: **string \*\* transliteration(string s, bool err);**

Входные данные: string s – строка, считанная из input.txt и bool err – состояние ошибки. Если будет встречен неописанный в блоке символ, будет выведено сообщение: "runtime error : unexpected symbol".

Выходные данные: возвращает двумерный массив типа string размера n x 2 вида ["символ цепочки"]["класс символа цепочки"], где n – длина исходной строки.

Например, символьную цепочку repeat myproc(1) until myfunc(a); блок транслитерации должен преобразовать в следующий двумерный вектор лексем:

|  |  |
| --- | --- |
| r | letter |
| e | letter |
| p | letter |
| e | letter |
| a | letter |
| t | space |
|  | letter |
| m | equal |
| y | number |
| p | number |
| r | dotcomma |
| o | letter |
| c | letter |
| ( | letter |
| 1 | letter |
| ) | letter |
|  | space |
| u | letter |
| n | equal |
| t | number |
| i | number |
| l | dotcomma |
|  | letter |
| m | letter |
| y | letter |
| f | letter |
| u | letter |
| n | space |
| c | letter |
| ( | equal |
| a | number |
| ) | number |
| ; | letter |

*Рисунок 1 - пример вектора блока транслитерации*

Перевод символов в нижний регистр:

Семантика: void tolower(string& str);

Входные данные: string& str – строка, считанная из input.txt

Выходные данные: –

Модуль проходит по строке и переводит все буквы в нижний регистр.

Лексический блок:

Семантика: **string \*\* lexicalBlock(string \*\* list, int & n);**

Входные данные:string \*\* list – двумерный массив, в котором находится строка, разбитая на символы с присвоенными им классами; int & n **–** длина входной строки.

Выходные данные: двумерный массив типа string размера n x 2

Блок отвечает за преобразование двумерного массива лексем, полученного от блока транслитерации, в двумерный массив лексем вида ["символ входного языка"]["класс символа входного языка"]. В рассматриваемом примере лексический блок должен выдать следующую цепочку лексем:

|  |  |
| --- | --- |
| repeat | id |
|  | space |
| myproc | id |
| ( | open |
| 1 | number |
| ) | close |
|  | space |
| until | id |
| myfunc | id |
| ( | open |
| a | number |
| ) | close |
| ; | dotcoma |

*Рисунок 2 - пример вектора лексического блока*

Блок идентификации ключевых слов:

Семантика: **bool keywordRecognize(string \*\* list, int & n);**

Входные данные: string \*\* list – двумерный массив, идентификаторы которого будут просматриваться на совпадения с ключевыми словами языка Паскаль; int & n – длина исходной строки.

Выходные данные: возвращает True если было обнаружено слово const в начале строки. Возвращает False если были обнаружены другие слова или слова const не было встречено в начале строки.

Блок устанавливает, присутствуют ли в двумерном массиве лексем ключевые слова языка Паскаль.

Синтаксический блок:

Семантика: **bool syntaxBlock(string \*\* list, int & n);**

Входные данные: **string \*\* list** – двумерный массив, содержащий лексемы и их классы; int & n – длина исходной строки.

Выходные данные: если конечный автомат перейдет в состояние НЕТ, то будет возвращено FALSE, иначе – TRUE.

Блок устанавливает, соответствует ли двумерный вектор заданным формулам Бэкуса-Наура. Данный блок использует для работы только классы символов входного языка. В рассматриваемом примере синтаксический блок, рассматривая цепочку вида KEYWORD\_CONST, SPACE, ID, EQUAL, NUMBER, SEMICOLON должен сообщить, что она синтаксически правильна.

# 3. Кодирование

## В данном пункте описан этап кодирования: структура текста программы, алгоритмы реализации модулей, размер текста программы. Алгоритмы реализации модулей поделены на три блока: блок транслитерации, лексический блок и синтаксический блок.

## 3.1. Структура текста программы

Practice.sln

lexicalBlock

transliteration

output

syntaxBlock

input

keywordRecognize

tolower

Описание всех функций в разделе.

# 3.2. Алгоритмы реализации модулей

В данном подразделе описаны детали реализации главных модулей программы.

### 3.2.1. Блок транслитерации

Все символы входной строки обрабатываются в два этапа: сначала переводятся в нижний регистр, затем разбиваются по классам, представленным в таблице 2.

**Таблица 2 - таблица транслитерации**

|  |  |
| --- | --- |
| **Символы** | **Класс лексемы** |
| a – z | letter |
| 0 - 9 | digit |
| ( | Open |
| ) | Close |
| , | Com |
| ; | Dotcoma |
|  | space |
| <другие символы> | err |

Если хотя бы одному символу был присвоен класс err, то в результате будет выведена ошибка "runtime error: unexpected symbol", а выполнение программы завершится.

### 3.2.2. Лексический блок

Построение обрабатывающего автомата начнем с построения “аналогичного” ему конечного распознавателя. В начале определим все возможные состояния распознавателя и их семантику в таблице 3.

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Состояние** | **Семантика** |
| 11 | НАЧ | Момент до начала обработки цепочки, либо чтение пробелов в начале строки. |
| 22 | КЛСЛОВО1 | Чтение ключевого слова (**repeat**). |
| 33 | ПРОБЕЛ1 | Чтение пробелов между ключевым словом и идентификатором. |
| 44 | ИД1 | Чтение первого идентификатора. |
| 55 | (1 | Считана первая **открывающая скобка.** |
| 66 | ПРОБЕЛ2 |  |
| 77 | ЗНАК | Считан знак **плюс** или **минус**. |
| 88 | ЧИСЛО | Чтение целого числа. |
| 99 | ПРОБЕЛ3 |  |
| 110 | )1 | Считана первая **закрывающая скобка.** |
| 111 | ПРОБЕЛ4 |  |
| 112 | КЛСЛОВО2 | Чтение ключевого слова (**until**). |
| 113 | ПРОБЕЛ5 |  |
| 114 | ИД2 | Чтение второго идентификатора. |
| 115 | (2 | Считана вторая **открывающая скобка**. |
| 116 | ПРОБЕЛ6 |  |
| 117 | ИД3 | Чтения третьего и последующих идентификаторов. |
| 118 | ЗПТ | Считан знак **запятая**. |
| 119 | ПРОБЕЛ7 |  |
| 220 | )2 | Считана вторая **закрывающая скобка**. |
| 221 | ПРОБЕЛ8 |  |
| 222 | ТЧКЗПТ | Считан знак **точка с запятой**. |
| 223 | ПРОБЕЛ9 | Чтение пробелов после **точки с запятой**. |
| 224 | E | Ошибка. |

Начальное состояние НАЧ. Допустимые состояния – ТЧКЗПТ и ПРОБЕЛ8. Конечный распознаватель, построенный на основе полученных состояние, приведен в таблице 4.

**Таблица 4 – конечный распознаватель лексического блока**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | letter | digit | space | sign | semicolon | left( | right) | zpt |
| НАЧ | КЛСЛОВО1 | Е | НАЧ | Е | Е | Е | Е | Е |
| КЛСЛОВО1 | КЛСЛОВО1 | Е | ПРОБЕЛ1 | Е | Е | Е | Е | Е |
| ПРОБЕЛ1 | ИД1 | Е | ПРОБЕЛ1 | Е | Е | Е | Е | Е |
| ИД1 | ИД1 | ИД1 | Е | Е | Е | (1 | Е | Е |
| (1 | Е | ЧИСЛО | ПРОБЕЛ2 | ЗНАК | Е | Е | Е | Е |
| ПРОБЕЛ2 | Е | Е | ПРОБЕЛ2 | Е | Е | Е | Е | Е |
| ЗНАК | Е | ЧИСЛО | Е | Е | Е | Е | Е | Е |
| ЧИСЛО | Е | ЧИСЛО | ПРОБЕЛ3 | Е | Е | Е | )1 | Е |
| ПРОБЕЛ3 | Е | Е | ПРОБЕЛ3 | Е | Е | Е | )1 | Е |
| )1 | Е | Е | ПРОБЕЛ4 | Е | Е | Е | Е | Е |
| ПРОБЕЛ4 | КЛСЛОВО2 | Е | ПРОБЕЛ4 | Е | Е | Е | Е | Е |
| КЛСЛОВО2 | КЛСЛОВО2 | Е | ПРОБЕЛ5 | Е | Е | Е | Е | Е |
| ПРОБЕЛ5 | ИД2 | Е | ПРОБЕЛ5 | Е | Е | Е | Е | Е |
| ИД2 | ИД2 | ИД2 | Е | Е | Е | (2 | Е | Е |
| (2 | ИД3 | Е | ПРОБЕЛ6 | Е | Е | Е | Е | Е |
| ПРОБЕЛ6 | ИД3 | Е | ПРОБЕЛ6 | Е | Е | Е | Е | Е |
| ИД3 | ИД3 | ИД3 | Е | Е | Е | Е | Е | ЗПТ |
| ЗПТ | Е | Е | ПРОБЕЛ7 | Е | Е | Е | Е | Е |
| ПРОБЕЛ7 | ИД3 | Е | ПРОБЕЛ7 | Е | Е | Е | )2 | Е |
| )2 | Е | Е | ПРОБЕЛ8 | Е | ТЧКЗПТ | Е | Е | Е |
| ПРОБЕЛ8 | Е | Е | ПРОБЕЛ8 | Е | ТЧКЗПТ | Е | Е | Е |
| ТЧКЗПТ | Е | Е | ПРОБЕЛ9 | Е | Е | Е | Е | Е |
| ПРОБЕЛ9 | Е | Е | ПРОБЕЛ9 | Е | Е | Е | Е | Е |
| E | Е | Е | Е | Е | Е | Е | Е | Е |

В полученном конечном распознавателе нет недостижимых состояний. Идентичными (а значит эквивалентными) состояниями являются (2 и ПРОБЕЛ6, )2 и ПРОБЕЛ8, ТЧКЗПТ и ПРОБЕЛ9. В дальнейшем будем обозначать их как (2, )2 И ТЧКЗПТ соответственно. Редукция данного конечного распознавателя приведена в таблице 5.

**Таблица 5 – редукция конечного распознавателя лексического блока**

|  |  |  |
| --- | --- | --- |
| **Шаг** | **Результат (блоки состояний)** | **Действия** |
| 0 | Р0={НАЧ, КЛСЛОВО1, ПРОБЕЛ1, ИД1, (1, ЗНАК, ЧИСЛО, ПРОБЕЛ3, )1, ПРОБЕЛ4, КЛСЛОВО2, ПРОБЕЛ5, ИД2, (2, ИД3, ПРОБЕЛ7, )2, ПРОБЕЛ8, ТЧКЗПТ, Е} | Разбиваем Р0 на два блока: допустимые и отвергающие. |
| 1 | Р={НАЧ, КЛСЛОВО1, ПРОБЕЛ1, ИД1, (1, ЗНАК, ЧИСЛО, ПРОБЕЛ3, )1, ПРОБЕЛ4, КЛСЛОВО2, ПРОБЕЛ5, ИД2, (2, ИД3, ПРОБЕЛ7, )2, ПРОБЕЛ8, Е}  Р={ТЧКЗПТ} | Разбиваем Р последовательно по входу semicolon, right), space, zpt, letter, left(, letter, space, letter, space, right). |
| 2 | Р={НАЧ, КЛСЛОВО1, ПРОБЕЛ1, ИД1, (1, ЗНАК, Е}  Р={ЧИСЛО, ПРОБЕЛ3}  Р={)1}  Р={ПРОБЕЛ4}  Р={КЛСЛОВО2}  Р={ПРОБЕЛ5}  Р={ИД2}  Р={(2}  Р={ИД3}  Р={ЗПТ}  Р={)2}  Р={ТЧКЗПТ} | Разбиваем Р по входу digit.  Разбиваем Р по входу digit. |
| 3 | Р={НАЧ, КЛСЛОВО1, ПРОБЕЛ1, ИД1, Е}  Р={(1, ЗНАК}  Р={ЧИСЛО}  Р={ПРОБЕЛ3}  Р={)1}  Р={ПРОБЕЛ4}  Р={КЛСЛОВО2}  Р={ПРОБЕЛ5}  Р={ИД2}  Р={(2}  Р={ИД3}  Р={ЗПТ}  Р={)2}  Р={ТЧКЗПТ} | Разбиваем Рпоследовательно по входу left(, letter, space, letter, space.  Разбиваем Р по входу sign. |
| 4 | Р={НАЧ}  Р={КЛСЛОВО1}  Р={ПРОБЕЛ1}  Р={ИД1}  Р={(1}  Р={ЗНАК}  Р={ЧИСЛО}  Р={ПРОБЕЛ3}  Р={)1}  Р={ПРОБЕЛ4}  Р={КЛСЛОВО2}  Р={ПРОБЕЛ5}  Р={ИД2}  Р={(2}  Р={ИД3}  Р={ЗПТ}  Р={)2}  Р={ТЧКЗПТ}  Р={E} | Разбиваем по входу Р по space.  Разбиваем по входу Р по space.  Разбиваем по входу Р по space. |

Проведенные преобразования доказывают, что конечный распознаватель, полученный после замены эквивалентных состояний, является минимальным. Данный распознаватель приведен в таблице 6.

**Таблица 6 – минимальный конечный распознаватель лексического блока.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | letter | digit | space | sign | semicolon | left( | right) | zpt |
| НАЧ | КЛСЛОВО1 | Е | НАЧ | Е | Е | Е | Е | Е |
| КЛСЛОВО1 | КЛСЛОВО1 | Е | ПРОБЕЛ1 | Е | Е | Е | Е | Е |
| ПРОБЕЛ1 | ИД1 | Е | ПРОБЕЛ1 | Е | Е | Е | Е | Е |
| ИД1 | ИД1 | ИД1 | Е | Е | Е | (1 | Е | Е |
| (1 | Е | ЧИСЛО | (1 | ЗНАК | Е | Е | Е | Е |
| ЗНАК | Е | ЧИСЛО | Е | Е | Е | Е | Е | Е |
| ЧИСЛО | Е | ЧИСЛО | ПРОБЕЛ3 | Е | Е | Е | )1 | Е |
| ПРОБЕЛ3 | Е | Е | ПРОБЕЛ3 | Е | Е | Е | )1 | Е |
| )1 | Е | Е | ПРОБЕЛ4 | Е | Е | Е | Е | Е |
| ПРОБЕЛ4 | КЛСЛОВО2 | Е | ПРОБЕЛ4 | Е | Е | Е | Е | Е |
| КЛСЛОВО2 | КЛСЛОВО2 | Е | ПРОБЕЛ5 | Е | Е | Е | Е | Е |
| ПРОБЕЛ5 | ИД2 | Е | ПРОБЕЛ5 | Е | Е | Е | Е | Е |
| ИД2 | ИД2 | ИД2 | Е | Е | Е | (2 | Е | Е |
| (2 | ИД3 | Е | (2 | Е | Е | Е | Е | Е |
| ИД3 | ИД3 | ИД3 | Е | Е | Е | Е | Е | ЗПТ |
| ЗПТ | Е | Е | ПРОБЕЛ7 | Е | Е | Е | Е | Е |
| ПРОБЕЛ7 | ИД3 | Е | ПРОБЕЛ7 | Е | Е | Е | )2 | Е |
| )2 | Е | Е | )2 | Е | )2 | Е | Е | Е |
| ТЧКЗПТ | Е | Е | ТЧКЗПТ | Е | Е | Е | Е | Е |
| E | Е | Е | Е | Е | Е | Е | Е | Е |

Теперь по полученному конечному распознавателю составим обрабатывающий автомат. Для этого введем примитивные процедуры, описанные в таблице 7.

**Таблица 7 – 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 | *Проверить id* | Проверить, совпадает ли текущая лексема с ключевыми словами языка Pascal. Если нет – продолжить обработку, если да – вызвать процедуру *Нет.* |
| 5 | *Проверить key\_word* | Проверить, является ли текущая лексема ключевым словом языка Pascal. Если да – продолжить обработку, если нет – вызвать процедуру *Нет.* |
| 6 | *Сохранить* | Сохранить текущую лексему в памяти. |
| 7 | *Лексема (класс)* | Перейти к рассмотрению следующей лексемы с заданным классом |

*Проверить id* и *Проверить key\_word* проверяются с помощью вызова блока идентификации ключевых слов.

Для удобства определим процедуры переходов, которые будем обозначать цифрами. Данные процедуры приведены в таблице 8. В случае, если встречается несколько процедур, то они выполняются последовательно, и лишь потом происходит переход.

**Таблица 8 – процедуры переходов обрабатывающего автомата лексического блока**

|  |  |
| --- | --- |
| **Действие** | **Семантика** |
| 1 | Обработать; |
| 2 | Сохранить; |
| 3 | Лексема (key\_word);  Обработать; |
| 4 | Лексема (id);  Обработать; |
| 5 | Лексема ( ( );  Обработать; |
| 6 | Лексема ( ) );  Обработать; |
| 7 | Лексема (number);  Обработать; |
| 8 | Лексема (semicolon);  Обработать; |
| 9 | Лексема (zpt);  Обработать; |
| 10 | Проверить id; |
| 11 | Проверить key\_word; |

Теперь можно на основе минимального конечного распознавателя составить соответствующий обрабатывающий автомат, он приведен в таблице 9.

**Таблица 9 – обрабатывающего автомата лексического блока.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | letter | digit | space | sign | semicolon | left( | right) | zpt |
| НАЧ | **3**  КЛСЛОВО1 |  | НАЧ |  |  |  |  |  |
| КЛСЛОВО1 | **1**  КЛСЛОВО1 |  | **2, 11**  ПРОБЕЛ1 |  |  |  |  |  |
| ПРОБЕЛ1 | **4**  ИД1 |  | ПРОБЕЛ1 |  |  |  |  |  |
| ИД1 | **1**  ИД1 | **1**  ИД1 |  |  |  | **2,10,5**  (1 |  |  |
| (1 |  | **7**  ЧИСЛО | **7**  ЗНАК | **7**  ЗНАК |  |  |  |  |
| ЗНАК |  | **1**  ЧИСЛО |  |  |  |  |  |  |
| ЧИСЛО |  | **1**  ЧИСЛО | ПРОБЕЛ3 |  |  |  | **6**  )1 |  |
| ПРОБЕЛ3 |  |  | ПРОБЕЛ3 |  |  |  | **6**  )1 |  |
| )1 |  |  | ПРОБЕЛ4 |  |  |  |  |  |
| ПРОБЕЛ4 | **3**  КЛСЛОВО2 |  | ПРОБЕЛ4 |  |  |  |  |  |
| КЛСЛОВО2 | **1**  КЛСЛОВО2 |  | **2, 11**  ПРОБЕЛ5 |  |  |  |  |  |
| ПРОБЕЛ5 | **4**  ИД2 |  | ПРОБЕЛ5 |  |  |  |  |  |
| ИД2 | **1**  ИД2 | **1**  ИД2 |  |  |  | **2,10,5**  (2 |  |  |
| (2 | **4**  ИД3 |  | (2 |  |  |  |  |  |
| ИД3 | **1**  ИД3 | **1**  ИД3 |  |  |  |  |  | **2,10,9**  ЗПТ |
| ЗПТ |  |  | ПРОБЕЛ7 |  |  |  |  |  |
| ПРОБЕЛ7 | **4**  ИД3 |  | ПРОБЕЛ7 |  |  |  | **6**  )2 |  |
| )2 |  |  | )2 |  | **8**  ТЧКЗПТ |  |  |  |
| ТЧКЗПТ |  |  | ТЧКЗПТ |  |  |  |  |  |

Все пустые клетки обозначают вызов примитивной процедуры *НЕТ.*

### 3.2.3. Синтаксический блок

Составим конечный распознаватель, опираясь на исходные формулы Бэкуса-Наура и полученные в лексическом блоке лексемы: id, key\_word, equal, number, semicolon. Данный распознаватель приведен в таблице 10, он имеет только одно допускающее состояние ТЧКЗПТ.

**Таблица 10 – конечный распознаватель синтаксического блока**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | key\_word | id | ( | ) | number | semicolon | zpt | E |  |
| **НАЧ** | КЛСЛОВО1 | Е | Е | Е | Е | Е | Е | Е | 0 |
| **КЛСЛОВО1** | Е | ИД1 | Е | Е | Е | Е | Е | Е | 0 |
| **ИД1** | Е | Е | (1 | Е | Е | Е | Е | Е | 0 |
| **(1** | Е | Е | Е | Е | ЧИСЛО | Е | Е | Е | 0 |
| **ЧИСЛО** | Е | Е | Е | )1 | Е | Е | Е | Е | 0 |
| **)1** | КЛСЛОВО2 | Е | Е | Е | Е | Е | Е | Е | 0 |
| **КЛСЛОВО2** | Е | ИД2 | Е | Е | Е | Е | Е | Е | 0 |
| **ИД2** | Е | Е | (2 | Е | Е | Е | Е | Е | 0 |
| **(2** | Е | ИД3 | Е | Е | Е | Е | Е | Е | 0 |
| **ИД3** | Е | Е | Е | )2 | Е | Е | ЗАП | Е |  |
| **ЗАП** | Е | ИД3 | Е | Е | Е | Е | Е | Е | 0 |
| **)2** | Е | Е | Е | Е | Е | ТЧКЗПТ | Е | Е | 0 |
| **ТЧКЗПТ** | Е | Е | Е | Е | Е | Е | Е | Е | 1 |
| **Е** | Е | Е | Е | Е | Е | Е | Е | Е | 0 |

Данный распознаватель является минимальным, это можно доказать методом разбиений, последовательно разбив Р0 на допустимые и отвергающие состояние.

Покажем, что этот распознаватель является минимальным. Для этого проведем редукцию методом разбиения. Ее пошаговое выполнение приведено в таблице 11.

**Таблица 11 - Редукция конечного распознавателя синтаксического блока**

|  |  |  |
| --- | --- | --- |
| **Шаг** | **Результат (блоки состояний)** | **Действия** |
| 0 | Р0={НАЧ, КЛСЛОВО1, ИД1, (1, ЧИСЛО, )1, КЛСЛОВО2, ИД2, (2, ИД3, )2, ТЧКЗПТ, Е} | Разбиваем Р0 на два блока: допустимые и отвергающие. |
| 1 | Р={НАЧ, КЛСЛОВО1, ИД1, (1, ЧИСЛО, )1, КЛСЛОВО2, ИД2, (2, ИД3, )2, Е}  Р={ТЧКЗПТ} | Разбиваем Р последовательно по входу semicolon, ), id, (, id, key\_word, ), number, (, id, key\_word. |
| 2 | Р={НАЧ}  Р={КЛСЛОВО1}  Р={ИД1}  Р={(1}  Р={ЧИСЛО}  Р={)1}  Р={КЛСЛОВО2}  Р={ИД2}  Р={(2}  Р={ИД3}  Р={)2}  Р={Е} |  |

Для составления обрабатывающего автомата вводим примитивные процедуры, приведенные в таблице 12.

**Таблица 12 – примитивные процедуры обрабатывающего автомата синтаксического блока**

|  |  |  |
| --- | --- | --- |
| **№п/п** | **Процедура** | **Семантика** |
| 1 | Да | Остановить обработку и допустить цепочку. |
| 2 | Нет | Остановить обработку и отвергнуть цепочку. |
| 3 | Проверить key\_word | Проверить, совпадает ли текущая лексема с ключевым словом const. Если да, то продолжить обработку. Иначе остановить обработку и отвергнуть цепочку. |

Обрабатывающий автомат, построенный на основе минимального конечного распознавателя, приведен в таблице 13.

**Таблица 13 – обрабатывающий автомат синтаксического блока**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | key\_word | id | ( | ) | number | semicolon | zpt | ˧ |  |
| **НАЧ** | *Проверить key\_word*  КЛСЛОВО1 |  |  |  |  |  |  |  |  |
| **КЛСЛОВО1** |  | ИД1 |  |  |  |  |  |  |  |
| **ИД1** |  |  | (1 |  |  |  |  |  |  |
| **(1** |  |  |  |  | ЧИСЛО |  |  |  |  |
| **ЧИСЛО** |  |  |  | )1 |  |  |  |  |  |
| **)1** | *Проверить key\_word*  КЛСЛОВО2 |  |  |  |  |  |  |  |  |
| **КЛСЛОВО2** |  | ИД2 |  |  |  |  |  |  |  |
| **ИД2** |  |  | (2 |  |  |  |  |  |  |
| **(2** |  | ИД3 |  |  |  |  |  |  |  |
| **ИД3** |  |  |  | )2 |  |  | ИД3 |  |  |
| **)2** |  |  |  |  |  | ТЧКЗПТ |  |  |  |
| **ТЧКЗПТ** |  |  |  |  |  |  |  | *ДА* |  |
| **Е** |  |  |  |  |  |  |  |  |  |

Все пустые клетки обозначают вызов примитивной процедуры *НЕТ*.

Реализация данного блока почти полностью идентична реализация лексического блока.

### 3.2.4. Блок идентификации ключевых слов

Реализация функции bool keywordRecognize(string \*\* list, int & n);

Подключается файл keywords.txt, содержащий ключевые слова языка Паскаль.

С помощью цикла for в массиве, содержащем ["лексема"]["класс лексемы"] находим класс лексемы **идент.**

Далее соответствующую классу лексему сравниваем со всеми идентификаторами в файлеkeyword.txt. Если совпадение есть, то функция заменяет возвращает False, иначе продолжится поиск идентификаторов. Кроме этого проверяется, что лексема класса **кл\_слово\_1** – слово repeat, а кл\_слово\_2 - until. Как только все идентификаторы будут просмотрены, функция оканчивает работу.

## 3.3. Размер текста программы (в строках)

Всего программа занимает 800 строк.

# 4. Тестирование

Данный раздел содержит протоколы автономного и комплексного тестирования различных блоков распознавателя цепочки.

## 4.1. Автономное тестирование

Часть протоколов тестирования для блоков транслитерации, лексического блока, блока идентификации ключевых слов и синтаксического блока приведены соответственно в таблицах 13, 14, 15 и 16.

**Таблица 13 – часть протокола автономного тестирования блока транслитерации**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Данные в векторе** | **Верно ли?** |
| repeat mypr | r(буква)  e(буква)  p(буква)  e(буква)  a(буква)  t(буква)  (прбел)  m(буква)  y(буква)  p(буква)  r(буква) | **Да** |
| R 45; 5 | R(буква)  (пробел)  4(цифра)  5(цифра)  ;(тчкзпт)  (пробел)  5(цифра) | **Да** |

**Таблица 14 – часть протокола автономного тестирования лексического блока**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| r(буква)  e(буква)  p(буква)  e(буква)  a(буква)  t(буква)  (прбел)  m(буква)  y(буква)  p(буква)  r(буква) | Reapeat(кл\_слово\_1)  (пробел)  Mypr(идент\_1) | **Да** |

**Таблица 15 – часть протокола автономного тестирования блока идентификации ключевых слов**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| const | false | Да |
| fgR | true | Да |
| repeat | false | Да |
| ringf | true | Да |

## 4.2. Комплексное тестирование

В таблице 16 приведена часть протокола тестирования головного модуля программы.

**Таблица 16 – часть протокола тестирования головного модуля**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Тест пройден?** |
| repeat myproc(1) until myfunc(a); | ACCEPT | да |
| repeat myproc(+16) until myfunc(param1, param2); | ACCEPT | да |
| repeat myproc(-123) until myfunc(p1, p2, p3, p4); | ACCEPT | да |
| repeat begin(-123) until myfunc(p1, p2, p3, p4); | REJECT | да |
| repeat myproc(-123) until myfunc(p1, p2, p3, p4) | REJECT | да |
| repeat myproc(-123 par) until myfunc(p1, p2, p3, p4); | REJECT | да |

# Заключение

Данная работа посвящена разработке распознавателя символьной цепочки, заданной формулами Бэкуса-Наура. В ходе выполнения работы были пройдены все основные этапы разработки программного обеспечения: анализ, написание спецификации, проектирование, разработка алгоритмов, кодирование, тестирование и сопровождение. Каждому этапу, за исключением сопровождения, в данной работе уделен отдельный пункт. В данных пунктах описана проделанная в каждом случае работа и ее результаты.

# Литература

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

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