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-104  Д. Э. Камалетдинов  Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_  Ст. преподаватель кафедры СП  Н. С. Силкина  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. Вариант A.

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: «23» июля 2021 г.

**Руководитель:**

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

Ст. преподаватель кафедры СП подпись Силкина Н. С.

**Задание принял к исполнению:**

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

подпись Камалетдинов Д.Э.

# ОГЛАВЛЕНИЕ

1. СПЕЦИФИКАЦИЯ 4
2. ПРОЕКТИРОВАНИЕ 6
   1. Модульная структура 6
   2. Интерфейсы модулей 7
3. КОДИРОВАНИЕ 9
   1. Структура текста программы 9
   2. Алгоритмы реализации модулей 11
      1. Блок транслитерации 11
      2. Лексический блок 12
      3. Синтаксический блок 19
      4. Блок идентификации ключевых слов 20
   3. Размер текста программы 20
4. ТЕСТИРОВАНИЕ 21
   1. Автономное тестирование 21
      1. Автономное тестирование блока транслитерации 21
      2. Автономное тестирование лексического блока 22
      3. Автономное тестирование блока ключевых слов 22
      4. Автономное тестирование синтаксического блока 23
   2. Комплексное тестирование 24
5. ЗАКЛЮЧЕНИЕ 25
6. ЛИТЕРАТУРА 26

# СПЕЦИФИКАЦИЯ

Необходимо разработать распознаватель заданной символьной цепочки. Символьная цепочка задается с помощью формул Бэкуса-Наура. Допустимые символьные цепочки определены вариантом A.

**<цепочка>::=<описание константы>**

<описание константы>::=**CONST** <идентификатор>**=**<значение>**;**

<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**

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

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

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

<знак>::=**+ | -**

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

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

<список 16-ричных букв и цифр>::=<цифра> | <16-ричная буква><список 16ричных букв и цифр>

<16-ричная буква>::=**A | B | C | D | E | F**

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 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.

Таблица 1 – Примеры входных и выходных данных.

| **INPUT.TXT** | **OUTPUT.TXT** |
| --- | --- |
| Const one=1; | ACCEPT |
| const RRR=$10B; | ACCEPT |
| CONST WEQ=-23; | ACCEPT |
| consT d21=$AA; | ACCEPT |
| conST SSS32aaA2= $2A32A1 | REJECT |
| conST SSaaaaaa11vvvvvvvvvvvvvvvvvaaaS32aaA2 = $2A3ABABA2A1; | REJECT |
| coonst sa1 = -23; | REJECT |
| const s11a1 = 2332 | REJECT |
| const 32wa1 = --323; | REJECT |
| const sa1 = $Q23; | REJECT |
| const sa1 = Q23;; | REJECT |

# ПРОЕКТИРОВАНИЕ

Раздел проектирования посвящен созданию модульной структуры программы для достижения простоты решения поставленной задачи. Также в разделе проектирования описана спецификация каждого модуля: назначение, входные и выходные данные.

## Модульная структура

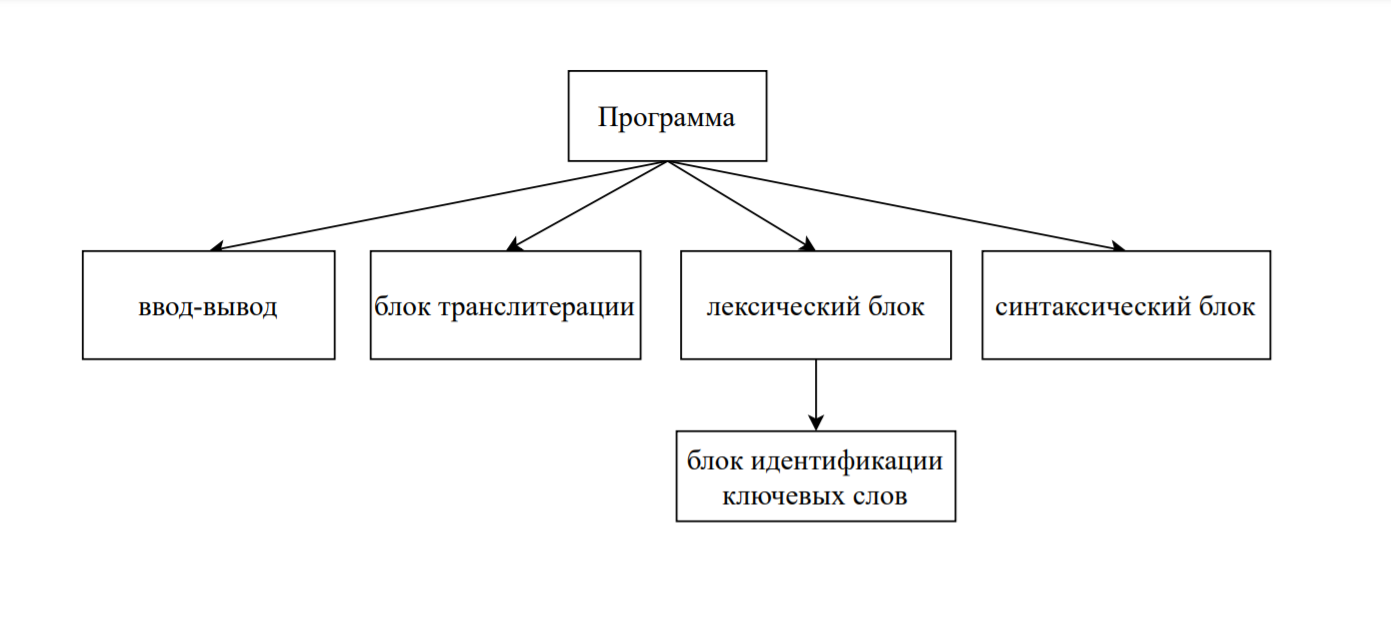
Модульная структура представляет из себя программу, содержащую в себе модули, изображенные на рисунке 1.

Рисунок 1 – модульная структура программы

***Блок ввода-вывода***

Подпрограмма ввода считывает исходную цепочку из текстового файла input.txt. Подпрограмма вывода записывает в текстовый файл output.txt строку ACCEPT, если цепочка соответствует заданным формулам Бэкуса-Наура, и REJECT, если цепочка не соответствует.

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

Например, символьную цепочку Const one=1; блок транслитерации должен преобразовать в цепочку лексем: (C,*буква*), (o,*буква*), (n,*буква*), (s,*буква*), (t,*буква*), (' ',*пробел*), (o,*буква*), (n,*буква*), (e,*буква*), (=,*равно*), (1,*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), (one, ИДЕНТИФИКАТОР), (=, РАВНО), (1, *ЦЕЛОЕ*), (;, ТЧКЗПТ).

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

***Синтаксический блок*** – подпрограмма, проверяющая цепочку лексем, полученных от лексического блока, на соответствие заданным формулам Бэкуса-Наура.

## Интерфейсы модулей

В этом пункте описаны интерфейсы модулей. Для каждого блока описаны функции и их параметры, их назначение и возвращаемый результат.

***Ввод и вывод данных***

def read\_line (): Выполняет считывание исходной строки из файла input.txt, результатом выполнения функции является строка.

def write(val): Осуществляет запись результата работы программы в файл output.txt.

***Блок транслитерации***

def transliteration(str): Преобразует строку в список ("символ цепочки", "класс символа цепочки ").

def classifier(symbol): Преобразует исходный символ в лексему вида ("символ цепочки", "класс символа цепочки ").

***Лексический блок***

def lexical(lexeme\_list): Преобразует список ("символ цепочки", "класс символа цепочки"), полученный от транслитератора, в список вида ("лексема входного языка", "класс лексемы входного языка") и осуществляет проверку идентификаторов на ключевые слова.

def classifier(lexeme, current\_state): Возвращает следующее состояние, принимая текущее и лексему входного языка.

***Блок идентификации ключевых слов***

def binary\_search(key\_words, id): Функция осуществляет бинарный поиск идентификатора в списке ключевых слов . В качестве формальных параметров принимает список ключевых слов key\_words и идентификатор id.

def search(id): Функция присвоения состояния ключевого слова или идентификатора входной лексеме. Возвращает состояние идентификатора.

***Синтаксический блок***

def syntactic(lexemes): Функция осуществляющая проверку корректности ввода строки (порядок следования лексем). В качестве формального параметра принимает список, состоящий из лексем входного языка и их классов.

def classifier(current\_state, lexeme): Функция возвращает следующее состояние, принимая текущее состояние, лексему входного языка и ее класс.

# КОДИРОВАНИЕ

Данный раздел посвящен реализации проекта программы в виде текстовых файлов на языке программирования. разработке структуры программы, разработке алгоритмов реализации модулей и их кодированию.

## Структура текста программы

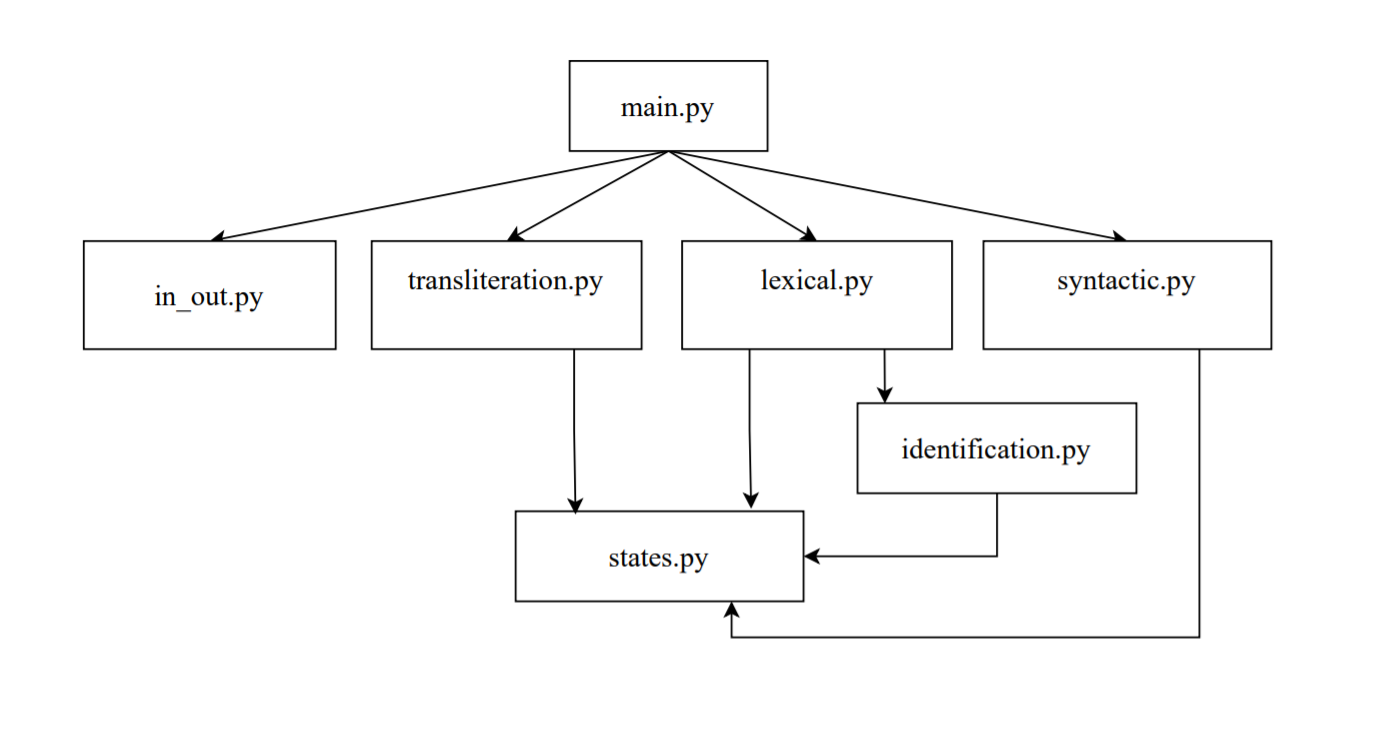
Структура текста программы идентична модульной структуре программы. Головной модуль main.py обращается к подчиненным модулям-файлам (рисунок 2).

Рисунок 2 – Структура текста программы.

***main.py***

Управляет ходом исполнения программы, используя конструкцию try-except. Последовательно обращается к каждому модулю распознавателя. В случае неожиданного результата обрабатывает исключения, возникшие в ходе работы программы.

***in\_out.py*** – организует работу с текстовыми файлами. Экспортирует две функции: def read() – функцию ввода и def write() – функцию вывода.

def read\_line(): Выполняет считывание исходной строки из файла input.txt, результатом выполнения функции является строка.

def write(val): Осуществляет запись результата работы программы в файл output.txt.

***transliteration.py*** – модуль, содержащий в себе две подпрограммы. def transliteration(str: str): преобразует строку в список кортежей вида ("символ цепочки", "класс символа цепочки "). Если длина строки превышает 80 символов, то выбрасывается исключение.

def classifier(symbol): преобразует исходный символ в кортеж вида ("символ цепочки", "класс символа цепочки "). Для некорректных символов выбрасывается исключение.

***lexical.py*** – модуль, содержащий в себе две подпрограммы: def lexical(lexeme\_list: list): преобразует список кортежей вида ("символ цепочки", "класс символа цепочки"), полученную от транслитератора, в список кортежей вида ("лексема входного языка", "класс лексемы входного языка") и осуществляет проверку идентификаторов на ключевые слова.

def classifier(lexeme, current\_state): возвращает следующее состояние, принимая текущее состояние и кортеж вида ("лексема входного языка", "класс лексемы входного языка").

***syntactic.py*** – модуль, содержащий в себе две подпрограммы: def syntactic(lexemes): функция осуществляющая проверку корректности ввода строки (порядок следования лексем). В качестве формального параметра принимает список lexemes вида ("лексема входного языка", "класс лексемы входного языка"). Если строка не завершается символом ";", то выбрасывается исключение.

def classifier (current\_state, lexeme): функция возвращает следующее состояние, принимая текущее состояние и кортеж ("лексема входного языка", "класс лексемы входного языка"). Если встречается неожиданная лексема, то выбрасывается исключение.

***identification.py*** – модуль, содержащий в себе две подпрограммы. def binary\_search(key\_words, id): функция осуществляет бинарный поиск идентификатора в списке ключевых слов . В качестве формальных параметров принимает список ключевых слов key\_words и идентификатор id.

def search(id): функция присвоения состояния ключевого слова или идентификатора входной лексеме. Возвращает состояние идентификатора.

***states.py*** – класс, содержащий в себе состояния конечных автоматов для лексического, синтаксического блока и блока транслитерации.

## Алгоритмы реализации модулей

Блок транслитерации, лексический и синтаксический блок реализованы в виде детерминированных конечных распознавателей [3]. Блок идентификации написан на основе алгоритма бинарного поиска.

### **Блок транслитерации**

Блок транслитерации – детерминированный конечный автомат [3], который выполняет обработку и распознавание входной символьной цепочки. Обработка входной цепочки заключается в выполнении транслитерации – формировании цепочки лексем вида («символ цепочки», «класс символа цепочки»). Распознавание входной цепочки означает, что данный автомат должен отвергать все символьные цепочки, которые заведомо не удовлетворяют формулам Бэкуса-Наура из условия задачи (буквы кириллицы, символы «, №, ! и др.)

Реализация блока выполнена в виде явно представленного конечного распознавателя. Для определения состояний был использован встроенный класс Enum модуля enum, для блока транслитерации - функция ord().

В таблице 2 приведена спецификация транслитератора для распознавания символьной цепочки.

Таблица 2 – Транслитерация символьной цепочки.

| **Символы** | **Класс лексемы** |
| --- | --- |
| G..Z,g..z | «чистая» буква |
| A-F | буква-цифра |
| 0..9 | цифра |
| = | равно |
| +, - | знак |
| пробел | пробел |
| ; | тчкзпт |
| $ | доллар |
| другие символы | ошибка |

### **Лексический блок**

Лексический блок – детерминированный конечный автомат, который выполняет обработку и распознавание входной цепочки лексем, полученной как результат работы блока транслитерации. Обработка входной цепочки заключается в формировании лексем вида («символ входного языка», «класс символа входного языка»). Распознавание входной цепочки означает, что данный автомат должен заведомо отвергать все цепочки лексем, содержащий лексемы, которые заведомо не удовлетворяют формулам Бэкуса-Наура из условия задачи (например, два знака числа подряд, отсутствие идентификатора или количество идентификаторов, более одного, отсутствие знака = , а также знака ; в конце строки и т.д.). Лексический блок осуществляется в один проход с идентификатором: это значит, что каждый раз, когда лексический блок формирует лексему класса ИДЕНТИФИКАТОР, запускается блок идентификации для проверки, является ли данный идентификатор ключевым словом языка Pascal.

Далее выполним редукцию построенного автомата. Редукция осуществляется следующим образом [3]: для начала должны быть отброшены все недостижимые состояния, затем найдем эквивалентные состояния и для всех групп эквивалентных состояний заменим каждое состояние в найденной группе на одно. Поиск эквивалентных состояний эффективно проводить методом разбиения [3]. Определяется список состояний и их семантику (таблица 3). Затем составляется конечный распознаватель входной цепочки лексем на основе полученных состояний (Таблица 4).

Метод разбиения заключается в последовательном разбиении множества состояний на такие непересекающиеся подмножества (блоки), что неэквивалентные состояния будут попадать в разные подмножества. Редукция конечного распознавателя описана в таблице 5. В таблице 6 приведен результат редукции конечного распознавателя лексического блока.

Реализация блока выполнена в виде явно представленного конечного распознавателя. Для определения состояний был использован встроенный класс Enum модуля enum.

Состояния обрабатывающего автомата лексического блока представлены классом LexicalStates. Если автомат получает неожиданные входные данные, то он выбрасывает исключение, которое обрабатывается в головном модуле.

Таблица 3 – Состояния конечного распознавателя лексического блока.

| **№**  **п/п** | **Состояние** | **Семантика** |
| --- | --- | --- |
|  | НАЧ | Момент до начала обработки цепочки либо чтение пробелов в начале строки. |
|  | ИД | Чтение идентификатора цепочки, имени константы |
|  | ПРОБЕЛ1 | Чтение пробелов между именем константы и идентификатором |
|  | РВН | Чтение знака равно |
|  | ПРОБЕЛ2 | Чтение пробелов между идентификатором и равно |
|  | ПРОБЕЛ3 | Чтение пробелов между равно и значением |
|  | ЗНАК | Чтение знака (- +) |
|  | ПРОБЕЛ4 | Чтение пробелов между знаком и целым |
|  | ЦЕЛ | Чтение целого |
|  | ПРОБЕЛ5 | Чтение пробелов между значением и точкой с запятой |
|  | ДОЛЛАР | Чтение знака доллара, ожидание 16-ричного числа |
|  | 16РИЧЦИФРА | Чтение 16-ричного числа |
|  | ТЧКЗПТ | Чтение точки с запятой |
|  | ПРОБЕЛ6 | Чтение пробелов после знака ; |
|  | E | Ошибка |

Таблица 4 – Конечный распознаватель лексического блока.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Чистая буква** | **Буква-цифра** | **цифра** | **равно** | **знак** | **пробел** | **$** | **;** | **др.** |  |
| **НАЧ** | ИД | ИД | E | E | E | НАЧ | E | E | E | 0 |
| **ИД** | ИД | ИД | ИД | РВН | E | ПРОБЕЛ1 | E | E | E | 0 |
| **ПРОБЕЛ1** | ИД | ИД | E | РВН | E | ПРОБЕЛ1 | E | E | E | 0 |
| **ПРОБЕЛ2** | E | E | E | РВН | E | ПРОБЕЛ2 | E | E | E | 0 |
| **РВН** | E | E | ЦЕЛ | E | ЗНАК | ПРОБЕЛ3 | ДОЛЛАР | E | E | 0 |
| **ПРОБЕЛ3** | E | E | ЦЕЛ | E | ЗНАК | ПРОБЕЛ3 | ДОЛЛАР | E | E | 0 |
| **ЗНАК** | E | E | ЦЕЛ | E | E | ЗНАК | E | E | E | 0 |
| **ПРОБЕЛ4** | E | E | ЦЕЛ | E | E | ПРОБЕЛ4 | E | E | E | 0 |
| **ЦЕЛ** | E | E | ЦЕЛ | E | E | ПРОБЕЛ5 | E | E | E | 0 |
| **ПРОБЕЛ5** | E | E | E | E | E | ПРОБЕЛ5 | E | ТЧКЗПТ | E | 0 |
| **ДОЛЛАР** | E | 16РИЧНЧИСЛ | ЦЕЛ | E | E | ПРОБЕЛ5 | E | E | E | 0 |
| **16РИЧНЧИСЛ** | E | 16РИЧНЧИСЛ | ЦЕЛ | E | E | ПРОБЕЛ5 | E | ТЧКЗПТ | E | 1 |
| **ТЧКЗПТ** | E | E | E | E | E | ТЧКЗПТ | E | E | E | 1 |
| **ПРОБЕЛ6** | E | E | E | E | E | ПРОБЕЛ6 | E | E | E | 1 |
| **E** | E | E | E | E | E | E | E | E | E | 0 |

Таблица 5 – Редукция конечного распознавателя лексического блока.

| **Шаг** | **Результат (блоки состояний)** | **Действия** |
| --- | --- | --- |
|  | P0 = {НАЧ, ИД, ПРОБЕЛ1, РВН, ПРОБЕЛ3, ЗНАК, ПРОБЕЛ4, ЦЕЛ, ПРОБЕЛ5, ДОЛЛАР, 16РИЧНЧИСЛ, ТЧКЗПТ, E} | Разбиваем P0 на два блока: допустимые и отвергающие состояния |
|  | P11 = {НАЧ, ИД, ПРОБЕЛ1, РВН, ПРОБЕЛ3, ЗНАК, ПРОБЕЛ4, ЦЕЛ, ПРОБЕЛ5, ДОЛЛАР, 16РИЧНЧИСЛ, ПРОБЕЛ6, E}  P12 = {ТЧКЗПТ} | Разбиваем P11 по входу чистая буква |
|  | P21 = {НАЧ, ИД, ПРОБЕЛ1}  P22 = {РВН, ПРОБЕЛ3, ЗНАК, ПРОБЕЛ4, ЦЕЛ, ПРОБЕЛ5, ДОЛЛАР, 16РИЧНЧИСЛ, ТЧКЗПТ, E}  P23 = {ТЧКЗПТ} | 1. Разбиваем P21 по входу цифра.  2. Разбиваем P22 по входу 16риччисл. |
|  | P31 = {НАЧ, ПРОБЕЛ1}  P32 = {ИД}  P33 = {РВН, ПРОБЕЛ3, ЗНАК, ПРОБЕЛ4, ЦЕЛ}  P34 = {ДОЛЛАР, 16РИЧНЧИСЛ}  P35 = {ПРОБЕЛ5, ТЧКЗПТ, E} | 1. Разбиваем P33 по входу знак  2. Разбиваем P35 по входу  равно. |
|  | P41 = {НАЧ, ПРОБЕЛ1}  P42 = {ИД}  P43 = {РВН, ПРОБЕЛ3}  P44 = {ДОЛЛАР, 16РИЧНЧИСЛ}  P45 = {ЗНАК, ПРОБЕЛ4, ЦЕЛ}  P46 = {ПРОБЕЛ5, ТЧКЗПТ, E} | 1. Разбиваем P44 по входу тчкзпт.  2. Разбиваем P45 по входу пробел  3. Разбиваем P46 по входу тчкзпт и пробел. |
|  | P51 = {НАЧ}  P52 = {ИД}  P53 = {РВН, ПРОБЕЛ3}  P54 = {ДОЛЛАР}  P55 = {ЗНАК}  P56 = {ПРОБЕЛ5}  P57 = {ПРОБЕЛ4}  P58 = {ЦЕЛ}  P59 = {ТЧКЗПТ}  P510 = {E}  P511 = {16РИЧНЧИСЛ}  P512 = {ПРОБЕЛ1} | В результате редукции были отобраны  состояния автомата в котором дальнейшие разбиения невозможны. |

Таблица 6 – Минимальный конечный распознаватель лексического блока.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Чистая буква** | **Буква-цифра** | **цифра** | **равно** | **знак** | **пробел** | **$** | **;** | **др.** |  |
| **НАЧ** | ИД | ИД | E | E | E | НАЧ | E | E | E | 0 |
| **ИД** | ИД | ИД | ИД | РВН | E | ПРОБЕЛ1 | E | E | E | 0 |
| **ПРОБЕЛ1** | ИД | ИД | E | РВН | E | ПРОБЕЛ1 | E | E | E | 0 |
| **РВН** | E | E | ЦЕЛ | E | ЗНАК | РВН | ДОЛЛАР | E | E | 0 |
| **ЗНАК** | E | E | ЦЕЛ | E | E | ЗНАК | E | E | E | 0 |
| **ПРОБЕЛ4** | E | E | ЦЕЛ | E | E | ПРОБЕЛ4 | E | E | E | 0 |
| **ЦЕЛ** | E | E | ЦЕЛ | E | E | ПРОБЕЛ5 | E | ТЧКЗПТ | E | 0 |
| **ПРОБЕЛ5** | E | E | E | E | E | ПРОБЕЛ5 | E | ТЧКЗПТ | E | 0 |
| **ДОЛЛАР** | E | 16РИЧНЧИСЛ | 16РИЧНЧИСЛ | E | E | ПРОБЕЛ5 | E | E | E | 0 |
| **16РИЧНЧИСЛ** | E | 16РИЧНЧИСЛ | 16РИЧНЧИСЛ | E | E | ПРОБЕЛ5 | E | ТЧКЗПТ | E | 0 |
| **ТЧКЗПТ** | E | E | E | E | E | ТЧКЗПТ | E | E | E | 1 |
| **E** | E | E | E | E | E | E | E | E | E | 0 |

Затем полученный минимальный конечный распознаватель преобразовывается в обрабатывающий автомат. Переходы дополняются вызовом примитивных процедур, приведенных в таблице 7. Для удобства чтения действия обрабатывающего автомата, выполняемые перед переходом в новое состояние, обозначены цифрами. Семантика действий дана в таблице 8. Обрабатывающий автомат лексического блока приведен в таблице 9. Пустым клеткам в Таблице 8 соответствует примитивная процедура НЕТ.

Таблица 7 – Примитивные процедуры автомата лексического блока.

|  |  |  |
| --- | --- | --- |
| № п/п | Процедура | Семантика |
|  | ДА | Остановить обработку и допустить цепочку. |
|  | НЕТ | Остановить обработку и отвергнуть цепочку. |
|  | Обработать | Добавить входной символ к значению текущей лексемы. |
|  | Лексема(класс) | Увеличить счетчик лексем на 1, установить заданный класс текущей лексемы. |

Таблица 8 – Процедуры переходов автомата лексического блока.

|  |  |
| --- | --- |
| Действие | Семантика |
| ❶ | *ОБРАБОТАТЬ;* |
| ❷ | ЛЕКСЕМА(ИД);  *ОБРАБОТАТЬ;* |
| ❸ | ЛЕКСЕМА(РВН);  *ОБРАБОТАТЬ;* |
| ❹ | ЛЕКСЕМА(ЗНАК);  *ОБРАБОТАТЬ;* |
| ❺ | ЛЕКСЕМА(ЦЕЛ);  *ОБРАБОТАТЬ;* |
| ❻ | ЛЕКСЕМА(ДОЛЛАР);  *ОБРАБОТАТЬ;* |
| ❼ | ЛЕКСЕМА(15РИЧНЧИСЛ);  *ОБРАБОТАТЬ;* |
| ❽ | ЛЕКСЕМА(ТЧКЗПТ);  *ОБРАБОТАТЬ;* |

Таблица 9 – Обрабатывающий автомат лексического блока.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Чистая буква** | **Буква-цифра** | **цифра** | **равно** | **знак** | **пробел** | **$** | **;** | **др.** |  |
| **НАЧ** | ❷  ИД | ❷  ИД |  |  |  | НАЧ |  |  |  | 0 |
| **ИД** | ❶  ИД | ❶  ИД | ❶  ИД | ❸  РВН |  | ПРОБЕЛ1 |  |  |  | 0 |
| **ПРОБЕЛ1** | ❶  ИД | ❶  ИД |  | ❸  РВН |  | ПРОБЕЛ1 |  |  |  | 0 |
| **РВН** |  |  | ❺  ЦЕЛ |  | ❹  ЗНАК | РВН | ❻  ДОЛЛАР |  |  | 0 |
| **ЗНАК** |  |  | ❺  ЦЕЛ |  |  | ЗНАК |  |  |  | 0 |
| **ПРОБЕЛ4** |  |  | ❺  ЦЕЛ |  |  | ПРОБЕЛ4 |  |  |  | 0 |
| **ЦЕЛ** |  |  | ❺  ЦЕЛ |  |  | ПРОБЕЛ5 |  |  |  | 0 |
| **ПРОБЕЛ5** |  |  | E |  |  | ПРОБЕЛ5 |  | ❽  ТЧКЗПТ |  | 0 |
| **ДОЛЛАР** |  | ❼  16РИЧНЧИСЛ | ❼  16РИЧНЧИСЛ |  |  | ПРОБЕЛ5 |  | E |  | 0 |
| **16РИЧНЧИСЛ** |  | ❷  16РИЧНЧИСЛ | ❷  16РИЧНЧИСЛ |  |  | ПРОБЕЛ5 |  | ❽  ТЧКЗПТ |  | 0 |
| **ТЧКЗПТ** |  |  |  |  |  | ❽  ТЧКЗПТ |  |  |  | 1 |
| **E** |  |  |  |  |  |  |  |  |  | 0 |

### **Синтаксический блок**

Синтаксический блок – реализован в виде детерминированного конечного распознавателя, который выполняет обработку и распознавание входной цепочки классов лексем, полученной от лексического блока. Распознавание цепочки означает, что данный автомат проверяет, соответствует ли порядок цепочки классов лексем формулам Бэкуса-Наура, описанным в разделе «Спецификация».

Конечный автомат создаем аналогично конечному автомату лексического блока. Используем метод разметки символов. Реализуем синтаксический блок в виде явно представленного конечного распознавателя. Для определения состояний был использован встроенный класс Enum модуля enum.

Состояния обрабатывающего автомата синтаксического блока представлены отдельным классом. Переход от состояния к состоянию осуществляется посредством сравнения переменной current\_state c текущим, сравнения 1 элемента кортежа с ожидаемым состоянием. Если автомат получает неожиданные входные данные, то он выбрасывает исключение, которое обрабатывается в головном модуле. Когда автомат попадает в состояние ошибки, автомат прекращает работу и возвращает строку «REJECT» в файл output.txt.

Таблица 10 – Конечный распознаватель синтаксического блока.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **ид** | **const** | **рвн** | **знак** | **цел** | **16риччисл** | **тчкзпт** |  |
| **НАЧ** | E | КЛСЛCONST | E | E | E | E | E | 0 |
| **КЛСЛCONST** | ИМЯ | E | E | E | E | E | E | 0 |
| **ИМЯ** | E | E | РАВНО | E | E | E | E | 0 |
| **РАВНО** | E | E | E | ЗНАК | ЦЕЛ | ДОЛЛАР | E | 0 |
| **ЗНАК** | E | E | E | E | ЦЕЛ | E | E | 0 |
| **ЦЕЛ** | E | E | E | E | E | E | ТЧКЗПТ | 0 |
| **ДОЛЛАР** | E | E | E | E | E | 16РИЧНЧИСЛ | E |  |
| **16РИЧНЧИСЛ** | E | E | E | E | E | E | ТЧКЗПТ | 0 |
| **ТЧКЗПТ** | E | E | E | E | E | E | E | 1 |

### **Блок идентификации ключевых слов**

Двоичный (бинарный) поиск представляет собой классический алгоритм поиска элемента в отсортированном массиве (Ключевые слова упорядочены по алфавиту в кортеже) использующий деление массива на половины [4].

Первым этапом бинарного поиска является определение значения элемента в середине структуры данных, затем полученное значение сравнивается с ключом. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй. Структура данных разделяется, поиск повторяется вновь до тех пор, пока не будет найден элемент со значением ключа или пока интервал не станет пустым.

Идентификация ключевого слова происходит всякий раз, когда лексический блок формирует строку, содержащую в себе лексему идентификатора. Функция binary\_search() осуществляет бинарный поиск ключевого слова и возвращает число.

Функция search() обрабатывает результат функции бинарного поиска. Если подпрограмма возвращает индекс «-1», то идентификатору присваивается класс идентификатора, если подпрограмма возвращает индекс «4» и, то идентификатору присваивается класс ключевого слова const. В иных случаях – класс другого ключевого слова.

## Размер текста программы (в строках)

Общий размер созданных исходных текстов в строках (включая пустые строки, комментарии, спецификации программных файлов и др.) 400 строк.

# ТЕСТИРОВАНИЕ

Тестирование программы проводится в два этапа: этап автономного тестирования каждого модуля программы и комплексного тестирования – тестирования головного модуля программы. Каждый модуль имеет свой определенный набор тестов (чтобы проверить работу модуля на обычных и граничных данных). Целью тестирования программных модулей служит демонстрация наличия ошибок.

## Автономное тестирование

Для поиска ошибок в модулях следует проводить автономное тестирование каждого модуля. используя отладчика системы программирования с "сухой" отладкой (работа без компьютера).

* + 1. Автономное тестирование блока транслитерации

В Таблице 11 приведен протокол тестирования модуля лексического блока. Результаты совпали с ожидаемыми, а значит, тестирование прошло успешно. Для удобства были использованы наборы различных символов, очевидно не являющихся цепочкой.

Таблица 11 – Протокол тестирования блока транслитерации.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ п/п** | **Входные данные** | **Выходные данные** | **Действительный результат** | **Тест**  **пройден?** |
| 1. | C o=+$1; | [('C', <SymbolType.SYMBOL\_NUMBER\_LETTER: 1>),  (' ', <SymbolType.SYMBOL\_SPACE: 5>),  ('o', <SymbolType.SYMBOL\_TRUE\_LETTER: 2>),  ('=', <SymbolType.SYMBOL\_EQUALS: 4>),  ('+', <SymbolType.SYMBOL\_SIGN: 6>),  ('$', <SymbolType.SYMBOL\_DOLLAR: 8>),  ('1', <SymbolType.SYMBOL\_NUMBER: 3>),  (';', <SymbolType.SYMBOL\_SEMICOLON: 7>)] | [('C', <SymbolType.SYMBOL\_NUMBER\_LETTER: 1>),  (' ', <SymbolType.SYMBOL\_SPACE: 5>),  ('o', <SymbolType.SYMBOL\_TRUE\_LETTER: 2>),  ('=', <SymbolType.SYMBOL\_EQUALS: 4>),  ('+', <SymbolType.SYMBOL\_SIGN: 6>),  ('$', <SymbolType.SYMBOL\_DOLLAR: 8>),  ('1', <SymbolType.SYMBOL\_NUMBER: 3>),  (';', <SymbolType.SYMBOL\_SEMICOLON: 7>)] | ДА |

* + 1. Автономное тестирование лексического блока

В Таблице 12 приведен протокол тестирования модуля лексического блока. Результаты совпали с ожидаемыми, а значит, тестирование прошло успешно.

Таблица 12 – Протокол тестирования лексического блока.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ п/п** | **Входные данные** | **Выходные данные** | **Действительный результат** | **Тест**  **пройден?** |
| 1. | Const one=1; | [('Const', <StatesLexical.STATE\_KEY\_CONST: 12>),  ('one', <StatesLexical.STATE\_ID: 2>),  ('=', <StatesLexical.STATE\_EQUALS: 5>),  ('1', <StatesLexical.STATE\_INTEGER: 7>),  (';', <StatesLexical.STATE\_SEMICOLON: 10>)] | [('Const', <StatesLexical.STATE\_KEY\_CONST: 12>),  ('one', <StatesLexical.STATE\_ID: 2>),  ('=', <StatesLexical.STATE\_EQUALS: 5>),  ('1', <StatesLexical.STATE\_INTEGER: 7>),  (';', <StatesLexical.STATE\_SEMICOLON: 10>)] | ДА |
| 2. | RRR=$10B; | [('RRR', <StatesLexical.STATE\_ID: 2>),  ('=', <StatesLexical.STATE\_EQUALS: 5>),  ('$', <StatesLexical.STATE\_DOLLAR: 14>),  ('10B', <StatesLexical.STATE\_HEXADECIMAL\_WORD: 9>),  (';', <StatesLexical.STATE\_SEMICOLON: 10>)] | [('RRR', <StatesLexical.STATE\_ID: 2>),  ('=', <StatesLexical.STATE\_EQUALS: 5>),  ('$', <StatesLexical.STATE\_DOLLAR: 14>),  ('10B', <StatesLexical.STATE\_HEXADECIMAL\_WORD: 9>),  (';', <StatesLexical.STATE\_SEMICOLON: 10>)] | ДА |

* + 1. Автономное тестирование блока ключевых слов

В Таблице 13 приведен протокол тестирования модуля блока идентификации ключевых слов. Блок тестируется вручную, аргументом функции является строка. Результатом – класс лексемы. Программа не чувствительна к регистру и обнаруживает ключевые слова независимо от него.

Таблица 13 – Протокол тестирования блока идентификации.

| **№ п/п** | **Входные данные** | **Выходные данные** | **Действительный результат** | **Тест**  **пройден?** |
| --- | --- | --- | --- | --- |
| 1. | 'and' | StatesLexical.STATE\_KEY\_ANOTHER | StatesLexical.STATE\_KEY\_ANOTHER | ДА |
| 2. | 'unTIl' | StatesLexical.STATE\_KEY\_ANOTHER | StatesLexical.STATE\_KEY\_ANOTHER | ДА |
| 3. | 'const' | StatesLexical.STATE\_KEY\_CONST | StatesLexical.STATE\_KEY\_CONST | ДА |
| 4. | 'coNSt' | StatesLexical.STATE\_KEY\_CONST | StatesLexical.STATE\_KEY\_CONST | ДА |
| 5. | 'wowow' | StatesLexical.STATE\_ID | StatesLexical.STATE\_ID | ДА |

* + 1. Автономное тестирование синтаксического блока

В Таблице 14 приведен протокол тестирования модуля синтаксического блока. Для некорректной цепочки должно выбрасываться исключение. Для корректной None, это значит, что исключений не последовало и можно записать в файл результат.

Таблица 14 – Протокол тестирования синтаксического блока.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ п/п** | **Входные данные** | **Выходные данные** | **Действительный результат** | **Тест**  **пройден?** |
| 1. | const sa1 = 23;; | Error. Неожиданая лексема ";" | Error. Неожиданая лексема ";" | ДА |
| 2. | Const one=1; | None | None | ДА |
| 3. | consT d21=AA; | Error. Неожиданая лексема "A" | Error. Неожиданая лексема "A" | ДА |
| 4. | CONST q21 = $A =; | Error. Неожиданая лексема "=" | Error. Неожиданая лексема "=" | ДА |
| 5. | Const one = 23 | Syntactic error. Строка не завершилась символом ";" | Syntactic error. Строка не завершилась символом ";" | ДА |

Так как после автономного тестирования блоков ошибок не обнаружено, можно переходить к комплексному тестированию программы.

## Комплексное тестирование

Для проверки корректности работы программы, нужно записать исходную строку в файл input.txt, запустить модуль main.py и проверить результат работы программы в файле output.txt. В Таблице 15 приведен протокол тестирования головного модуля.

Таблица 15 – Протокол тестирования синтаксического блока.

| **№ п/п** | **Входные данные** | **Выходные данные** | **Действительный результат** | **Тест**  **пройден?** |
| --- | --- | --- | --- | --- |
| 1. | Const one=1; | ACCEPT | ACCEPT | ДА |
| 2. | const RRR=$10B; | ACCEPT | ACCEPT | ДА |
| 3. | CONST WEQ=-23; | ACCEPT | ACCEPT | ДА |
| 4. | consT d21=$AA; | ACCEPT | ACCEPT | ДА |
| 5. | conST SSS32aaA2= $2A32A1 | REJECT | REJECT | ДА |
| 6. | conST SSaaaaaa11vvvvvvvvvvvvvvvvvaaaS32aaA2 = $2A3ABABA2A1; | REJECT | REJECT | ДА |
| 7. | coonst sa1 = -23; | REJECT | REJECT | ДА |
| 8. | const s11a1 = 2332 | REJECT | REJECT | ДА |
| 9. | const 32wa1 = --323; | REJECT | REJECT | ДА |
| 10. | const sa1 = $Q23; | REJECT | REJECT | ДА |
| 11. | const sa1 = Q23;; | REJECT | REJECT | ДА |

# ЗАКЛЮЧЕНИЕ

В рамках выполнения летней учебной практики, был разработан распознаватель символьной цепочки, задаваемой формулами Бэкуса-Наура для варианта A. Реализация программы происходила в несколько этапов.

1) Анализ условия поставленной задачи, разработка спецификации программы на основе требований БНФ для варианта A.

2) Проектирование модульной структуры программы;

3)Разработка алгоритмов и кодирование спроектированных модулей;

4) Тестирование и отладки модулей, написание отчета по проделанной работе.

# ЛИТЕРАТУРА

1. Йенсен К., Вирт Н. Паскаль. Руководство пользователя и описание языка. - М.: Компьютер, 1995.
2. Левитин А. В. Глава 4. Метод декомпозиции: Бинарный поиск // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 180—183. — 576 с.
3. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. - М.: Мир, 1979.
4. Enum в Python [Электронный ресурс]: Все о Python. Программирование на Python 3 - Режим доступа: https://all-python.ru/osnovy/enum.html (дата обращения: 05.09.2021).
5. Python. Enum здорового человека [Электронный ресурс]: блог Антона Жиянова. - Режим доступа: https://antonz.ru/enum/ (дата обращения: 05.09.2021).