**Вариант 7. Описание заголовка функции в Паскале**

Заголовок функции в Паскале. Заголовок начинается ключевым словом «function» или «procedure», далее идет имя функции, скобка, несколько описаний аргументов. Описание аргументов содержит имена переменных через запятую, затем двоеточие, затем имя типа. Достаточно рассматривать только примитивные типы (массивы, записи и т. п. не требуется). После этого в случае функции следует двоеточие и имя типа.

Используйте один терминал для всех имен переменных и имен типов. Используйте один терминал для ключевых слов function и т. п. (не несколько ‘f’, ‘u’, ‘n’, ...).

Пример: function fib(n: integer): integer;

**1 Разработка грамматики**

Построим грамматику.

S → k v ( A ) : v ;

A → ε

A → B

B → v : v

B → B , B

|  |  |
| --- | --- |
| **Нетерминал** | **Описание** |
| S | Правильная заголовка функции в Паскале |
| A | Список аргументов функции |
| B | Непустой список аргументов функции |

В грамматике есть левая рекурсия. Устраним ее. Получится грамматика:

S → k v ( A ) : v ;

A → ε

A → B

B → v : v B'

B' → , B B'

B' → ε

|  |  |
| --- | --- |
| **Нетерминал** | **Описание** |
| S | Правильная заголовка функции в Паскале |
| A | Список аргументов функции |
| B | Непустой список аргументов функции |
| B' | Продолжение непустого списка аргументов функции |

**2 Построение лексического анализатора**

В нашей грамматике семь терминалов:

ключевое слово «function» или «procedure», имя функции/переменной/типа,

'(' ')' ',' ':' ';'

Построим лексический анализатор. Заведем класс Token для хранения терминалов. Не забудем также про конец строки.

public enum Token {

KEYWORD, NAME, LPAREN, RPAREN, COMMA, COLON, SEMICOLON, END;

}

|  |  |
| --- | --- |
| **Терминал** | **Токен** |
| function, procedure | KEYWORD |
| имя функции/переменной/типа | NAME |
| ( | LPAREN |
| ) | RPAREN |
| , | COMMA |
| : | COLON |
| ; | SEMICOLON |
| $ | END |

В файле LexicalAnalyzer.java написан лексический анализатор для этих токенов.

**3 Построение синтаксического анализатора**

Построим множества FIRST и FOLLOW для нетерминалов нашей грамматики.

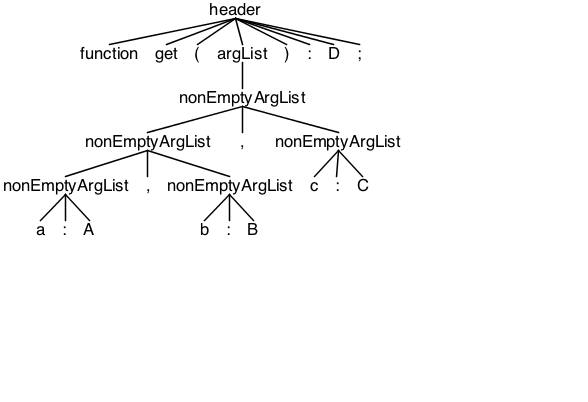
|  |  |  |
| --- | --- | --- |
| **Нетерминал** | **First** | **Follow** |
| S | k | $ |
| A | v | ) |
| B | v | , ) |
| B’ | , | , ) |

В файле Tree.java заведен структура данных для хранения дерева.

В файле Parser.java приведен синтаксический анализатор с использованием рекурсивного спуска.

**4 Визуализация дерева разбора**

Для изучения результата разработайте систему визуализации дерева разбора.



**5 Подготовка набора тестов**

|  |  |
| --- | --- |
| **Тест** | **Описание** |
| function f(): void; | Тест на правило A → ε |
| procedure test(test: TestType): Bool; | Тест на правило A → B и B → v: v |
| function get (x:XType, aParam: ParamType, dasd: sdasd) : ReturnType; | Тест на правило A → B и B → B, B |
| procedure proc(a : A, b : B, c : C, d : D, e : E, f : F, g : G, h : H, i : I, j : J, k : K, l : L, m : M) : X; |  |