Лабораторная работа №2

© Дарья Яковлева

31 марта 2016 года

Вариант 8. Визуализация переменных в Си

Описания переменных в Си. Сначала следует имя типа, затем разделенные запятой имена переменных. Переменная может быть указателем, в этом случае перед ней идет звездочка (возможны и указатели на указатели, и т. д.). Описаний может быть несколько. Используйте один терминал для всех имен переменных и имен типов.

Пример

```
int a, *b, ***c, d;
```

- **Разработка грамматики**

  Построим грамматику.

  S → D

  D → TAD

  D → ε

  A → ' 'B

  B → VC

  C → ,VC

  C → ;

  V → *V

  V → a

  T → a

  S – стартовый нетерминал

  D – одно описание переменных

  T – тип переменных

  A – переход от типа к переменным

  B – первая переменная

  C – вторая и все остальные переменные

  V – переменная

  Левой рекурсии нет.

- **Построение лексического анализатора**

```
enum Token {
    COMMA, SEMICOLON, STAR, NAME, SPACE, END;
}

public class LexicalAnalyzer {

    String input;
    int curPos;
    Token curToken;
    char curChar;
    int haveType;

    public LexicalAnalyzer(String input) {
        this.input = input + "$";
        curPos = 0;
        nextChar();
        haveType = 0;
    }

    void nextChar() {
        curChar = input.charAt(curPos);
        curPos++;
    }

    public Token curToken() {
        return curToken;
    }

    boolean isBlank(char c) {
        return c == ' ' || c == '\r' || c == '\n' || c == '\t';
    }

    boolean next() {
        return curPos < input.length();
    }

    public void nextToken() throws ParseException {
        while (next() && isBlank(curChar)) {
            nextChar();
        }
        if (isBlank(curChar)) {
            throw new ParseException("End of input string at position ", curPos)
        }
        if (haveType == 1) {
            curToken = Token.SPACE;
            haveType = 2;
            return;
```

```
        }
        if (curChar == '*') {
            curToken = Token.STAR;
            nextChar();
        } else if (curChar == ';') {
            curToken = Token.SEMICOLON;
            nextChar();
            haveType = 0;
        } else if (curChar == ',') {
            curToken = Token.COMMA;
            nextChar();
        } else if (curChar == '$') {
            curToken = Token.END;
        } else {
            String type = "" + curChar;
            while (next() && !isBlank(curChar) && curChar != ',' && curChar != '
                nextChar();
                type += curChar;
            }
            curToken = Token.NAME;
            if (haveType == 0) {
                haveType = 1;
            }
        }
    }
}
```

- **Построение синтаксического анализатора**

Нетерминал	**FIRST**	**FOLLOW**
S	ε, a	$
D	ε, a	$
A	space	a
B	*, a	a
C	,, ;	a
V	*, a	,, ;
T	a	' '

Структура данных для хранения дерева

```
class Tree {
    String node;
    List<Tree> children;
    public Tree(String node, Tree... children) {
        this.node = node;
        this.children = Arrays.asList(children);
    }
    public Tree(String node) {
```

```
        this.node = node;
        this.children = new ArrayList<>();
    }
}
```

Синтаксический анализатор

```
public class Parser {
    LexicalAnalyzer lex;

    Tree parse(String input) throws ParseException {
        lex = new LexicalAnalyzer(input);
        lex.nextToken();
        return S();
    }

    Tree S() throws ParseException {
        switch (lex.curToken()) {
            case NAME:
                //D
                Tree sub = D();
                return new Tree("S", sub);
            case END:
                //eps
                return new Tree("S");
            default:
                throw new AssertionError();
        }
    }
    Tree D() throws ParseException {
        switch (lex.curToken()) {
            case NAME:
                //T
                Tree sub = T();
                //A
                Tree cont = A();
                //D
                Tree cont2 = D();
                return new Tree("D", sub, cont, cont2);
            case END:
                //eps
                return new Tree("D", new Tree("$"));
            default:
                throw new AssertionError();
        }
    }
    Tree T() throws ParseException {
```

```
    switch (lex.curToken()) {
        case NAME:
            lex.nextToken();
            return new Tree("T", new Tree("a"));
        default:
            throw new AssertionError();
    }
}
Tree A() throws ParseException {
    switch (lex.curToken()) {
        case SPACE:
            lex.nextToken();
            Tree cont = B();
            return new Tree("A", new Tree("space"), cont);
        default:
            throw new AssertionError();
    }
}
Tree B() throws ParseException {
    switch (lex.curToken()) {
        case STAR:
            Tree sub = V();
            Tree cont = C();
            return new Tree("B", sub, cont);
        case NAME:
            sub = V();
            cont = C();
            return new Tree("B", sub, cont);
        default:
            throw new AssertionError();
    }
}
Tree V() throws ParseException {
    switch (lex.curToken()) {
        case STAR:
            lex.nextToken();
            Tree cont = V();
            return new Tree("V", new Tree("*"), cont);
        case NAME:
            lex.nextToken();
            return new Tree("V", new Tree("a"));
        default:
            throw new AssertionError();
    }
}

Tree C() throws ParseException {
```

```
        switch (lex.curToken()) {
            case COMMA:
                lex.nextToken();
                Tree cont = V();
                Tree cont2 = C();
                return new Tree("C", new Tree(","), cont, cont2);
            case SEMICOLON:
                lex.nextToken();
                return new Tree("C", new Tree(";"));
            default:
                throw new AssertionError();
        }
    }
}
```

- **Визуализация дерева разбора**

  Тест

  ```
  int a, **b; double c;
  ```

  Дерево разбора

  ```
  S -> D -> T -> a
       | -> A -> space
       |    | -> B -> V -> a
       |         | -> C -> ,
       |              | -> V -> *
       |              |    | -> V -> *
       |              |         | -> V -> a
       |              | -> C -> ;
       | -> D -> T -> a
            | -> A -> space
            |    | -> B -> V -> a
            |         | -> C -> ;
            | -> D -> $
  ```

- **Подготовка набора тестов**

Тест	**Описание**
int a;	Простой тест
int *a;	Простой тест №2
double a, b, c;	Тест на правило C → ,VC
int *a, **b;	Тест на правило V → *V
int a; double c;	Тест на правило D → TAD
	Тест на правило D → ε
int a; float *s, *****d, e;	Случайный тест