Лабораторная работа №2

© Плотников Андрей

13 декабря 2017 г.

Вариант 6. 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 – стартовый нетерминал

  D – объявление переменной

  T – тип переменных

  K – разделитель типа и имени

  F – первая переменная

  N – последующие переменные

  V – имя переменной

  S → D

  D → TKD

  D → ε

  K → ' 'F

  F → VN

  N → ,VN

  N → ;

  V → *V

  V → a

  T → a

  В грамматике нет левой рекурсии.

- **Построение лексического анализатора**

```
private enum Token {
    ASTERISK,  // *
    COMMA,     // ,
    SEMICOLON, // ;
    NAME,      // a-z
    END        // $
}

public class Parser {

    private static String processingString;
    private static Token processingToken;
    private static char proccessingChar;
    private static int offset;

    public static Node parse (String input) {
        processingString = input + "$";
        processingToken = null;
        proccessingChar = 0;
        offset = 0;

        nextToken ();
        return fetchEnterPoint ();
    }

    private static boolean isBlank (char symbol) {
        return Character.isWhitespace (symbol);
    }

    private static boolean isControl (char symbol) {
        return symbol == ',' || symbol == ';';
    }

    private static boolean hasMore () {
        return offset < processingString.length ();
    }

    private static void nextChar () {
        proccessingChar = processingString.charAt (offset ++);
    }

    private static void nextToken () {
        while (hasMore () && isBlank (proccessingChar)) {
            nextChar (); // Skipping blank spaces
        }
```

```
        if (isBlank (proccessingChar)) {
            throw new ParseException ("End of string reached at ", offset);
        }

        switch (proccessingChar) {
            case '*':
                processingToken = Token.ASTERISK;
                nextChar ();
                break;
            case ',':
                processingToken = Token.COMMA;
                nextChar ();
                break;
            case ';':
                processingToken = Token.SEMICOLON;
                nextChar ();
                break;
            case '$':
                processingToken = Token.END;
                break;

            default:
                while (hasMore ()
                        && !isBlank (proccessingChar)
                        && !isControl (proccessingChar)) {
                    nextChar ();
                }

                processingToken = Token.NAME;
        }
    }

    ...
}
```

- **Построение синтаксического анализатора**

  Построим множества FIRST и FOLLOW для нетерминалов грамматики

Нетерминал	**FIRST**	**FOLLOW**
S	ε, a	$
D	ε, a	$
K	space	a
F	*, a	a
N	,, ;	a
V	*, a	,, ;
T	a	space

Структура данных для хранения дерева

```
public class Node {

    private final List <Node> _list;
    private final String _name;

    public Node (String name, Node... children) {
        this._list = new ArrayList <> (Arrays.asList (children));
        this._name = name;
    }

}
```

Синтаксический анализатор

```
public class Parser {
    ...

    // S
    private static Node fetchEnterPoint () {
        String method = "Enter point";
        switch (processingToken) {
            case NAME:
                return new Node (method,
                                 fetchVariableDeclaration ());
            case END:
                return new Node (method, new Node ("$"));

            default:
                throw new ParseException ("Enter point not found at ",
                                          offset);
        }
    }

    // D
    private static Node fetchVariableDeclaration () {
        String method = "Variable declaration";
        switch (processingToken) {
            case NAME:
                Node type = fetchVariableType ();
                Node skip = fetchVariableSkiper ();
                return new Node (method, type, skip,
                                 fetchVariableDeclaration ());
            case END:
                return new Node (method, new Node ("$"));
```

```
        default:
            throw
                new ParseException ("Variable declaration not found at ",
                                    offset);
    }
}

// T
private static Node fetchVariableType () {
    String method = "Variable type";
    switch (processingToken) {
        case NAME:
            nextToken ();
            return new Node (method, new Node ("a"));

        default:
            throw new ParseException ("Variable type not found at ",
                                    offset);
    }
}

// K
private static Node fetchVariableSkiper () {
    String method = "Variable skiper";
    switch (processingToken) {
        case ASTERISK:
        case NAME:
            return new Node (method, fetchVariableEnterance ());

        default:
            throw new ParseException ("Variable skiper not found at ",
                                    offset);
    }
}

// F
private static Node fetchVariableEnterance () {
    String method = "Variable enterance";
    switch (processingToken) {
        case ASTERISK:
        case NAME:
            Node variable = fetchVariable ();
            Node next = fetchNextVariable ();
            return new Node (method, variable, next);

        default:
```

```
                throw
                    new ParseException ("Variable enterance not found at ",
                                        offset);
            }
        }

        // V
        private static Node fetchVariable () {
            String method = "Variable";
            switch (processingToken) {
                case ASTERISK:
                    nextToken ();
                    Node variable = fetchVariable ();
                    return new Node (method, new Node ("*"),
                                     variable);
                case NAME:
                    nextToken ();
                    return new Node (method, new Node ("a"));

                default:
                    throw new ParseException ("Variable not found at ",
                                        offset);
            }
        }

        // N
        private static Node fetchNextVariable () {
            String method = "Next variable";
            switch (processingToken) {
                case COMMA:
                    nextToken ();
                    Node variable = fetchVariable ();
                    Node next = fetchNextVariable ();
                    return new Node (method, new Node (","),
                                     variable, next);
                case SEMICOLON:
                    nextToken ();
                    return new Node (method);

                default:
                    throw new ParseException ("Variable not found at ",
                                        offset);
            }
        }
    }
```

- **Визуализация дерева разбора**

  В качестве примера для визуализации возьмём следующий тест:

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

  Дерево разбора

```
Enter point (S)
|- Variable declaration (D)
    |- Variable type (T)
        |- a
    |- Variable skiper (K)
        |- Variable enterance (F)
            |- Variable (V)
                |- a
            |- Next variable (N)
                |- ,
                |- Variable (V)
                    |- *
                    |- Variable (V)
                        |- a
            |- Next variable (N)
                |- ,
                |- Variable (V)
                    |- *
                    |- Variable (V)
                        |- *
                        |- Variable (V)
                            |- a
                |- Next variable (N)
    |- Variable declaration (D)
        |- Variable type (T)
            |- a
        |- Variable skiper (K)
            |- Variable enterance (F)
                |- Variable (V)
                    |- a
            |- Next variable (N)
                |- ,
                |- Variable (V)
                    |- a
                |- Next variable (N)
        |- Variable declaration (D)
            |- $
```

- **Подготовка набора тестов**

Тест	**Описание**
int a;	Простой тест
int *a;	Простой тест №2
int a, b, c;	Тест на правило N $\rightarrow$,VN
int *a, **b;	Тест на правило V $\rightarrow$ *V
int a; int c;	Тест на правило D $\rightarrow$ TKD
	Тест на правило D $\rightarrow \varepsilon$
int *a, b; double c, *d; float e;	Случайный тест
int ***a, ******b, ***c; double n, *m;	Случайный тест №2