Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

**Лабораторная работа №2**

**Дисциплина «Основы разработки компиляторов»**

**Вариант 12**

**Выполнил:**

Съестов Дмитрий Вячеславович

Группа P3317

**Преподаватель:**

Логинов Иван Павлович

Санкт-Петербург

2019

**1. Порядок выполнения работы**

1.1. По варианту задания определить, какие классы лексем будут в вашем языке.

1.2. Составить контрольные примеры на реализуемом языке. Хотя бы один пример должен проверять поведение вашей программы при наличии недопустимых символов в транслируемом файле.

1.3. Запрограммировать и отладить модуль сканирования. Выполнить тестирование на контрольных примерах. Результатом работы должна быть таблица, содержащая лексемы и признаки их классов, для числовых констант их внутреннее представление (шестнадцатеричное). Необходимо включить

в результирующий файл информацию о номерах строк исходного текста

транслируемой программы.  
Одинаковые идентификаторы и константы в таблицу повторно не записываются.  
Необходимо предусмотреть восстановление после ошибок.

1.4. Оформить отчет.

**2. Содержание отчета.**

2.1. Название работы и ее исполнители.

2.2. Цель работы.

2.3. БНФ реализуемого языка.

2.4. Список классов лексем реализуемого языка.

2.5. Краткое (по 2-3 предложения) описание процедур (функций), из

которых состоит программа лексического анализа. Наилучший вариант –

включение описаний в текст программы в виде комментариев.

2.6. Листинг программы (не обязательно).

2.7. Распечатки контрольных примеров и результатов их выполнения.

2.8. Выводы по проделанной работе.

**Цель работы –** реализация лексического анализатора для языка со следующей БНФ:

**<Программа> ::=** <Объявление переменных> <Описание вычислений>

**<Описание вычислений> ::=** Begin < Список операторов > End

**<Объявление переменных> ::=** Int <Список переменных> |

Int <Список переменных> <Объявление переменных> | Bin <Список переменных> | Bin <Список переменных> <Объявление переменных>

**<Список переменных> ::=** <Идент>; | <Идент> , <Список переменных>

**<Список присваиваний>::=** <Присваивание> | <Присваивание> <Список присваиваний>

**<Присваивание> ::=** <Идент> := <Выражение> ;

**<Выражение> ::=** <Ун.оп.> <Подвыражение> | <Подвыражение>

**<Подвыражение> ::=** (<Выражение>) | <Операнд> | < Подвыражение > <Бин.оп.>

<Подвыражение>

**<Ун.оп.> ::=** "-"

**<Бин.оп.> ::=** "&" | "|" | "^" | "-" | "+" | "\*" | "/"

**<Операнд> ::=** <Идент> | <Const>

**<Идент> ::=** <Буква> <Идент> | <Буква>

**<Const> ::=** < BConst >|< DConst > |  
**<DConst> ::=** <Цифра> |<Цифра><DConst>

**<BConst> ::=** 0|1

Комментарий в стиле С++ однострочный.

**Классы лексем:**

* **Ключевые слова:** Begin, End, Int, Bin
* **Разделители:** ( ) , ;
* **Операторы:** - & | ^ - + \* / :=
* **Идентификаторы:** имена переменных
* **Литералы:** целые числа, булевы константы (0 и 1)

Разбор осуществляется классом Lexer, который последовательно находит в тексте токены:

*fun* printTokens() {  
 *do* {  
 *val* t = nextToken()  
 println(t)  
 } *while* (t *is* Error || (t *as* Token).type != TokenType.END\_OF\_INPUT)  
}

Функция nextToken возвращает либо токен, либо сообщение об ошибке. Она особым образом обрабатывает слеш (деление либо комментарий) и оператор присваивания :=, вызывая соответствующие функции.

*private fun* nextToken(): LexerResult {  
 *val* line = lineNum  
 *val* pos = linePos  
 *while* (char.isWhitespace()) nextChar()  
  
 *return when* (char) {  
 EOF -> Token(TokenType.END\_OF\_INPUT, "", *this*.lineNum, *this*.linePos)  
 '/' -> divOrComment(line, pos)  
 ':' -> assignOrError(line, pos)  
 *else* -> singleCharLexems[char]?.let **{** nextChar()  
 Token(**it**, "", line, pos)  
 **}** ?: identifierOrLiteral(line, pos)  
 }  
 }

assignOrError проверяет, что за двоеточием стоит знак равенства, а иначе возвращает ошибку:

*private fun* assignOrError(line: Int, pos: Int): LexerResult {  
 *if* (nextChar() == '=') {  
 nextChar()  
 *return* Token(TokenType.OP\_ASSIGN, "", line, pos)  
 }  
 *return* error(String.format("assignOrError: unrecognized character: (%d) '%c'", char.toInt(), char), line, pos)  
}

divOrComment проверяет, относится ли слеш к началу комментария или это оператор деления. Если это комментарий, лексер пропускает остаток текущей строки.

*private fun* divOrComment(line: Int, pos: Int): LexerResult {  
 *if* (nextChar() != '/') {  
 *return* Token(TokenType.OP\_DIV, "", line, pos)  
 }  
 *while* (char != EOF && char != '\n') nextChar()  
 *return* nextToken()  
}

identifierOrLiteral обрабатывает все остальные лексемы. Если лексема равна 0 или 1, это логическая константа. Если все символы – цифры, то это число. Если все символы – буквы, это либо ключевое слово, либо идентификатор. В противном случае возвращается ошибка.

*private fun* identifierOrLiteral(line: Int, pos: Int): LexerResult {  
 *var* allLetters = *true  
 var* allDigits = *true  
 var* text = ""  
  
 *while* (char.isLetterOrDigit()) {  
 text += char  
 allLetters = allLetters && char.isLetter()  
 allDigits = allDigits && char.isDigit()  
 nextChar()  
 }  
  
 *return when* (text) {  
 "" -> error(String.format("identifierOrLiteral: unrecognized character: (%d) %c", char.toInt(), char), line, pos)  
 "0" -> Token(TokenType.B\_ZERO, "0", line, pos)  
 "1" -> Token(TokenType.B\_ONE, "1", line, pos)  
 *else* -> {  
 *when* {  
 allDigits -> Token(TokenType.INTEGER, text, line, pos)  
 allLetters -> keywords[text]?.let **{** Token(it, "", line, pos)  
 **}** ?: Token(TokenType.IDENTIFIER, text, line, pos)  
 text[0].isDigit() && !allDigits -> error(String.format("identifierOrLiteral: invalid number: %s", text), line, pos)  
 *else* -> error(String.format("identifierOrLiteral: invalid identifier: %s", text), line, pos)  
 }  
 }  
 }  
}

**Примеры программ**

|  |  |
| --- | --- |
| Корректная программа | |
| Int foo, bar, baz;  Bin a, b, c;  Begin //This is a comment    foo := 5 \* 20 - 25;  bar := -foo / 5;  b := 1;    End | 1 0 KEYWORD\_INT  1 3 IDENTIFIER foo  1 7 COMMA  1 8 IDENTIFIER bar  1 12 COMMA  1 13 IDENTIFIER baz  1 17 SEMICOLON  1 18 KEYWORD\_BIN  2 4 IDENTIFIER a  2 6 COMMA  2 7 IDENTIFIER b  2 9 COMMA  2 10 IDENTIFIER c  2 12 SEMICOLON  2 13 KEYWORD\_BEGIN  5 0 IDENTIFIER foo  6 5 OP\_ASSIGN  6 8 INTEGER 5  6 10 OP\_MUL  6 12 INTEGER 20  6 15 OP\_SUB  6 17 INTEGER 25  6 20 SEMICOLON  6 21 IDENTIFIER bar  7 5 OP\_ASSIGN  7 8 OP\_SUB  7 10 IDENTIFIER foo  7 13 OP\_DIV  7 15 INTEGER 5  7 17 SEMICOLON  7 18 IDENTIFIER b  8 3 OP\_ASSIGN  8 6 B\_ONE  8 8 SEMICOLON  8 9 KEYWORD\_END  10 4 END\_OF\_INPUT |

|  |  |
| --- | --- |
| Программа с ошибками | |
| Int foo, bar, baz;  Bin a> b> c;  Begin    foo := 5 \* 20 - 25;  bar := {foo / 5  b = 1;    End | 1 0 KEYWORD\_INT  1 3 IDENTIFIER foo  1 7 COMMA  1 8 IDENTIFIER bar  1 12 COMMA  1 13 IDENTIFIER baz  1 17 SEMICOLON  1 18 KEYWORD\_BIN  2 4 IDENTIFIER a  identifierOrLiteral: unrecognized character: (62) > in lineNum 2, linePos 6  2 7 IDENTIFIER b  identifierOrLiteral: unrecognized character: (62) > in lineNum 2, linePos 9  2 10 IDENTIFIER c  2 12 SEMICOLON  2 13 KEYWORD\_BEGIN  4 6 IDENTIFIER foo  6 5 OP\_ASSIGN  6 8 INTEGER 5  6 10 OP\_MUL  6 12 INTEGER 20  6 15 OP\_SUB  6 17 INTEGER 25  6 20 SEMICOLON  6 21 IDENTIFIER bar  7 5 OP\_ASSIGN  identifierOrLiteral: unrecognized character: (123) { in lineNum 7, linePos 8  7 10 IDENTIFIER foo  7 13 OP\_DIV  7 15 INTEGER 5  7 17 IDENTIFIER b  identifierOrLiteral: unrecognized character: (61) = in lineNum 8, linePos 3  8 5 B\_ONE  8 7 SEMICOLON  8 8 KEYWORD\_END  10 4 END\_OF\_INPUT |

**Вывод**

В ходе выполнения данной работы был реализован простой лексический анализатор, способный выделять в тексте программы лексемы, а в случае ошибки выдавать сообщение и продолжать работу.