Лабораторная работа № 1.3 «Объектно-ориентированный лексический анализатор»

21 марта 2023 г.

Самохвалова Полина, ИУ9-62Б

Цель работы

Целью данной работы является приобретение навыка реализации лексического анализатора на объектно-ориентированном языке без применения каких-либо средств автоматизации решения задачи лексического анализа.

Индивидуальный вариант

Целые числа: последовательности цифр определенной системы счисления, предваренные соответствующим индикатором, определяющим систему счисления (для десятичных чисел — пустой индикатор, для двоичных чисел — «0b», для восьмеричных чисел — «0t», для шестнадцатеричных чисел — «0x», шестнадцатеричные цифры могут быть записаны в любом регистре). Ключевые слова: «and»,«or». Знаки операций: «(»,«)». Идентификаторы: последовательности латинских букв.

Реализация

Main.java

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Main {
    public static void main(String[] args) throws IOException {
        Compiler compiler = new Compiler();
        String program = new String(Files.readAllBytes(
                Paths.get(args[0])));
```

```
        Scanner scanner = new Scanner(program, compiler);
        System.out.println("TOKENS:");
        Token token = scanner.nextToken();
        while (token.tag != DomainTag.END_OF_PROGRAM) {
            System.out.println(token);
            token = scanner.nextToken();
        }
        if (compiler.isErrors()) {
            System.out.println("MESSAGES:");
            compiler.outputMessages();
        }
    }
}
```

DomainTag.java

```
enum DomainTag {
    OPERATION, NUMBER, KEYWORD, IDENT, UNKNOWN, END_OF_PROGRAM
}
```

Token.java

```
abstract public class Token {
    public final DomainTag tag;
    public final Fragment coords;

    protected Token(DomainTag tag, Position starting,
                    Position following) {
        this.tag = tag;
        this.coords = new Fragment(starting, following);
    }

    @Override
    public String toString() {
        return coords.toString();
    }
}
```

KeywordToken.java

```
public class KeywordToken extends Token {
    public final String value;

    protected KeywordToken(String value, Position starting, Position
            following) {
        super(DomainTag.KEYWORD, starting, following);
        this.value = value;
    }
```

```
    @Override
    public String toString() {
        return "KEYWORD " + super.toString() + ": " + value;
    }
}
```

NumberToken.java

```
public class NumberToken extends Token {
    public final long value;

    protected NumberToken(long value, Position starting, Position
            following) {
        super(DomainTag.NUMBER, starting, following);
        this.value = value;
    }

    @Override
    public String toString() {
        return "NUMBER " + super.toString() + ": " + value;
    }
}
```

OperationToken.java

```
public class OperationToken extends Token {
    public final String value;

    protected OperationToken(String value, Position starting, Position
            following) {
        super(DomainTag.OPERATION, starting, following);
        this.value = value;
    }

    @Override
    public String toString() {
        return "OPERATION " + super.toString() + ": " + value;
    }
}
```

IdentToken.java

```
public class IdentToken extends Token {
    public final int value;

    protected IdentToken(int value, Position starting, Position
            following) {
        super(DomainTag.IDENT, starting, following);
        this.value = value;
```

```
    }

    @Override
    public String toString() {
        return "IDENT " + super.toString() + ": " + value;
    }
}
```

UnknownToken.java

```
public class UnknownToken extends Token {

    public UnknownToken(DomainTag tag, Position starting,
                        Position following) {
        super(tag, starting, following);
    }
}
```

Scanner.java

```
public class Scanner {
    public final String program;

    private Compiler compiler;
    private Position cur;

    public Scanner(String program, Compiler compiler) {
        this.compiler = compiler;
        this.program = program;
        cur = new Position(program);
    }

    public Token nextToken() {
        while (!cur.isEndOfFile()) {
            while (cur.isWhitespace()) {
                cur = cur.next();
            }
            Token token;
            if (cur.isBracket()) {
                token = readOperation(cur);
            } else if (cur.isDecimalDigit()) {
                token = readNumber(cur);
            } else if (cur.isLetter()) {
                token = readKeywordOrIdent(cur);
            } else {
                token = new UnknownToken(DomainTag.UNKNOWN, cur,
                        cur.next());
            }
```

```
            if (token.tag == DomainTag.UNKNOWN) {
                compiler.addMessage(true, cur, "unknown token");
                cur = token.coords.following;
            } else {
                cur = token.coords.following;
                return token;
            }
        }
        return new UnknownToken(DomainTag.END_OF_PROGRAM, cur, cur);
    }

    private Token readOperation(Position cur) {
        Position p = cur.next();
        return new OperationToken(program.substring(cur.getIndex(),
                p.getIndex()), cur, p);
    }
    private Token readKeywordOrIdent(Position cur) {
        Position p = new Position(cur);
        while (p.isLetter()) {
            p = p.next();
        }
        String s = program.substring(cur.getIndex(), p.getIndex());
        if ((s.equals("and")) || (s.equals("or"))) {
            return new KeywordToken(program.substring(cur.getIndex(),
                    p.getIndex()), cur, p);
        } else {
            return new IdentToken(compiler.addName(program.substring(
                    cur.getIndex(), p.getIndex())), cur, p);
        }
    }

    private Token readNumber(Position cur) {
        Position p = cur.next();
        boolean error;
        if (cur.isZero()) {
            if (p.isBinaryInd()) {
                p = p.next();
                error = false;
                if (p.isDecimalDigit()) {
                    while (p.isDecimalDigit()) {
                        if (!p.isBinaryDigit()) {
                            error = true;
                        }
                        p = p.next();
                    }
                } else {
```

```
            error = true;
        }
        if (!error) {
            return new NumberToken(Long.parseLong(
                    program.substring(cur.getIndex() + 2,
                            p.getIndex()), 2), cur, p);
        } else {
            return new UnknownToken(DomainTag.UNKNOWN, cur, p);
        }
    } else if (p.isOctalInd()) {
        p = p.next();
        error = false;
        if (p.isDecimalDigit()) {
            while (p.isDecimalDigit()) {
                if (!p.isOctalDigit()) {
                    error = true;
                }
                p = p.next();
            }
        } else {
            error = true;
        }
        if (!error) {
            return new NumberToken(Long.parseLong(
                    program.substring(cur.getIndex() + 2,
                            p.getIndex()), 8), cur, p);
        } else {
            return new UnknownToken(DomainTag.UNKNOWN, cur, p);
        }
    } else if (p.isHexInd()) {
        p = p.next();
        error = false;
        if (p.isDecimalDigit() || p.isLetter()) {
            while (p.isDecimalDigit() || p.isLetter()) {
                if (!p.isHexDigit()) {
                    error = true;
                }
                p = p.next();
            }
        } else {
            error = true;
        }
        if (!error) {
            return new NumberToken(Long.parseLong(
                    program.substring(cur.getIndex() + 2,
                            p.getIndex()), 16), cur, p);
```

```
                } else {
                    return new UnknownToken(DomainTag.UNKNOWN, cur, p);
                }
            }
        }
        while (p.isDecimalDigit()) {
            p = p.next();
        }
        return new NumberToken(Long.parseLong(program.substring(
                cur.getIndex(), p.getIndex())), cur, p);
    }

}
```

Position.java

```
public class Position {
    private String text;

    private int line, pos, index;

    public String getText() {
        return text;
    }

    public int getLine() {
        return line;
    }

    public int getPos() {
        return pos;
    }

    public int getIndex() {
        return index;
    }

    public Position(String text) {
        this.text = text;
        line = pos = 1;
        index = 0;
    }

    public Position(Position p) {
        this.text = p.getText();
        this.line = p.getLine();
        this.pos = p.getPos();
```

```
    this.index = p.getIndex();
}

@Override
public String toString() {
    return "(" + line + ", " + pos + ')';
}

public boolean isEndOfFile() {
    return index == text.length();
}

public int getCode() {
    return isEndOfFile() ? -1 : text.codePointAt(index);
}

public boolean isWhitespace() {
    return !isEndOfFile() && Character.isWhitespace(getCode());
}

public boolean isDecimalDigit() {
    return !isEndOfFile() && text.charAt(index) >= '0' &&
            text.charAt(index) <= '9';
}

public boolean isBinaryDigit() {
    return !isEndOfFile() && text.charAt(index) >= '0' &&
            text.charAt(index) <= '1';
}

public boolean isOctalDigit() {
    return !isEndOfFile() && text.charAt(index) >= '0' &&
            text.charAt(index) <= '7';
}

public boolean isHexDigit() {
    return !isEndOfFile() && ((text.charAt(index) >= '0' &&
            text.charAt(index) <= '9') ||
            (text.charAt(index) >= 'A' &&
                    text.charAt(index) <= 'F') ||
            (text.charAt(index) >= 'a' &&
                    text.charAt(index) <= 'f'));
}

public boolean isLetter() {
    return !isEndOfFile() && Character.isLetter(getCode());
```

```
}

public boolean isBracket() {
    return !isEndOfFile() && ((text.charAt(index) == '(') ||
            text.charAt(index) == ')');
}

public boolean isBinaryInd() {
    return !isEndOfFile() && text.charAt(index) == 'b';
}

public boolean isOctalInd() {
    return !isEndOfFile() && text.charAt(index) == 't';
}

public boolean isHexInd() {
    return !isEndOfFile() && text.charAt(index) == 'x';
}

public boolean isZero() {
    return !isEndOfFile() && text.charAt(index) == '0';
}

public boolean isNewLine() {
    if (index == text.length()) {
        return true;
    } else {
        return (text.charAt(index) == '\n');
    }
}

public Position next() {
    Position p = new Position(this);
    if (!p.isEndOfFile()) {
        if (p.isNewLine()) {
            p.line++;
            p.pos = 1;
        } else {
            if (Character.isHighSurrogate(
                    p.text.charAt(p.index)))
                p.index++;
            p.pos++;
        }
        p.index++;
    }
    return p;
```

```
    }

}
```

Fragment.java

```
public class Fragment {
    public final Position starting, following;

    public Fragment(Position starting, Position following) {
        this.starting = starting;
        this.following = following;
    }

    public String toString() {
        return starting.toString() + " - " + following.toString();
    }
}
```

Compiler.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Compiler {
    private List<Message> messages;
    private Map<String, Integer> nameCodes;
    private List<String> names;

    public Compiler() {
        messages = new ArrayList<Message>();
        nameCodes = new HashMap<String, Integer>();
        names = new ArrayList<String>();
    }

    public int addName(String name) {
        if (nameCodes.containsKey(name))
            return nameCodes.get(name);
        else {
            int code = names.size();
            names.add(name);
            nameCodes.put(name,code);
            return code;
        }
    }
```

```
    public void addMessage(boolean isErr, Position c, String text) {
        messages.add(new Message(isErr, text, c));
    }

    public void outputMessages() {
        for (Message m : messages) {
            System.out.print(m.isError ? "Error" : "Warning");
            System.out.print(" " + m.position + ": ");
            System.out.println(m.text);
        }
    }

    public boolean isErrors() {
        return messages.size() > 0;
    }
    public Scanner getScanner(String program) {
        return new Scanner(program, this);
    }
}
```

Message.java

```
public class Message {
    public final boolean isError;
    public final String text;
    public final Position position;

    public Message(boolean isError, String text, Position position) {
        this.isError = isError;
        this.text = text;
        this.position = position;
    }
}
```

Тестирование

Входные данные:

```
  0b010011   snjss  (23 4843)
0xA328FF   0t99    ?
and    0t2356761   or 325  andersen
```

Вывод на `stdout`:

```
TOKENS:
NUMBER (1, 3) - (1, 11): 19
IDENT (1, 14) - (1, 19): 0
```

```
OPERATION (1, 21) - (1, 22): (
NUMBER (1, 22) - (1, 24): 23
NUMBER (1, 25) - (1, 29): 4843
OPERATION (1, 29) - (1, 30): )
NUMBER (2, 1) - (2, 9): 10692863
KEYWORD (3, 1) - (3, 4): and
NUMBER (3, 8) - (3, 17): 646641
KEYWORD (3, 20) - (3, 22): or
NUMBER (3, 23) - (3, 26): 325
IDENT (3, 28) - (3, 36): 1
MESSAGES:
Error (2, 12): unknown token
Error (2, 20): unknown token
```

Вывод

В результате выполнения лабораторной работы был приобретен навык реализации лексического анализатора на объектно-ориентированном языке без применения каких-либо средств автоматизации решения задачи лексического анализа. На языке программирования Java были реализованы две первые фазы стадии анализа: чтение входного потока и лексический анализ. Реализация осуществлялась согласно схеме реализации объектно-ориентированного лексического анализатора, рассмотренной на лекции. Чтение входного потока осуществляется из файла, при этом лексический анализатор вычисляет текущие координаты в обрабатываемом тексте. В результате работы программы в стандартный поток вывода выдаются описания распознанных лексем.