Construção de Compiladores 1 - 2018.1 - Profs. Mário César San Felice (e Helena Caseli, Murilo Naldi, Daniel Lucrédio)
Tópico 02 - Análise Léxica - Roteiro de Códigos

Demonstração 1 – Analisador léxico "na mão" – parte 1
Primeira tentativa de fazer análise léxica: lendo tokens de 1 caractere (ou no máximo 2)

---

1. Criar um novo arquivo, no Desktop, com um programa de exemplo (salvar com codificação ocidental)

```
:DECLARACOES
argumento:INT
fatorial:INT

:ALGORITMO
% Calcula o fatorial de um número inteiro
LER argumento
ATRIBUIR argumento A fatorial
SE argumento = 0 ENTAO ATRIBUIR 1 A fatorial
ENQUANTO argumento > 1
   INICIO
      ATRIBUIR fatorial * (argumento - 1) A fatorial
      ATRIBUIR argumento - 1 A argumento
   FIM
IMPRIMIR fatorial
```

2. Abrir o NetBeans, e criar novo projeto Java "AlgumaLex"
3. Criar um enum algumalex.TipoToken

```
package algumalex;
public enum TipoToken {
    PCDeclaracoes, PCAlgoritmo, PCInteiro, PCReal, PCAtribuir, PCA, PCLer,
    PCImprimir, PCSe, PCEntao, PCEnquanto, PCInicio, PCFim,
    OpAritMult, OpAritDiv, OpAritSoma, OpAritSub,
    OpRelMenor, OpRelMenorIgual, OpRelMaiorIgual,
    OpRelMaior, OpRelIgual, OpRelDif,
    OpBoolE, OpBoolOu,
    Delim, AbrePar, FechaPar, Var, NumInt, NumReal, Cadeia, Fim
}
```

4. Criar a classe algumalex.Token

```
package algumalex;
public class Token {
    public TipoToken nome;
    public String lexema;
    public int indiceTS;
    public Token(TipoToken nome, String lexema) {
        this(nome, lexema, -1);
    }
```

```
    public Token(TipoToken nome, String lexema, int indiceTS) {
        this.nome = nome;
        this.lexema = lexema;
        this.indiceTS = indiceTS;
    }
    @Override
    public String toString() {
        if(indiceTS != -1)
            return "<"+nome+","+lexema+","+indiceTS+">";
        else return "<"+nome+","+lexema+">";
    }
}
```

5. Criar a classe algumalex.LeitorDeArquivosTexto

```
package algumalex;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class LeitorDeArquivosTexto {
    InputStream is;
    public LeitorDeArquivosTexto(String arquivo) {
        try {
            is = new FileInputStream(new File(arquivo));
        } catch (Exception ex) {
            ex.printStackTrace(System.err);
        }
    }
    public int lerProximoCaractere() {
        try {
            int ret = is.read();
            System.out.print((char)ret);
            return ret;
        } catch (Exception ex) {
            ex.printStackTrace(System.err);
            return -1;
        }
    }
}
```

6. Criar a classe algumalex.AlgumaLexico

```
package algumalex;
public class AlgumaLexico {
    LeitorDeArquivosTexto ldat;
    public AlgumaLexico(String arquivo) {
        ldat = new LeitorDeArquivosTexto(arquivo);
    }
    public Token proximoToken() {
        int caractereLido = -1;
        while((caractereLido = ldat.lerProximoCaractere()) != -1) {
```

```
            char c = (char)caractereLido;
            if(c == ' ' || c == '\n') continue;
        }
        return null;
    }
}
```

7. Criar a classe algumalex.Main, com o seguinte código no main() e executar

```
        AlgumaLexico lex = new AlgumaLexico(<caminho para o arquivo>);
        Token t = null;
        while((t = lex.proximoToken()) != null) {
            System.out.print(t);
        }
```

8. Adicionar na classe AlgumaLexico o código para os tokens com um único caractere e executar

```
            if(c == ':') {
                return new Token(TipoToken.Delim,":");
            }
            else if(c == '*') {
                return new Token(TipoToken.OpAritMult,"*");
            }
            else if(c == '/') {
                return new Token(TipoToken.OpAritDiv,"/");
            }
            else if(c == '+') {
                return new Token(TipoToken.OpAritSoma,"+");
            }
            else if(c == '-') {
                return new Token(TipoToken.OpAritSub,"-");
            }
            else if(c == '(') {
                return new Token(TipoToken.AbrePar,"(");
            }
            else if(c == ')') {
                return new Token(TipoToken.FechaPar,")");
            }
```

9. Problema: e tokens com mais de um caractere? Adicionar o seguinte código e explicar

```
            else if(c == '<') {
                c = (char)ldat.lerProximoCaractere();
                if(c == '>')
                    return new Token(TipoToken.OpRelDif,"<>");
                else if(c == '=')
                    return new Token(TipoToken.OpRelMenorIgual,"<=");
                else return new Token(TipoToken.OpRelMenor,"<");
            }
```

10. Mudar, no arquivo de entrada, a linha do ENQUANTO, e mostrar que ainda está funcionando

```
ENQUANTO argumento <= 1
```

11. Mudar, no arquivo de entrada, a linha do ENQUANTO, e mostrar que agora não está mais funcionando (cuidado para não deixar nenhum espaço depois do símbolo “<”)

```
ENQUANTO argumento <1
```

Demonstração 2 – Analisador léxico "na mão" – parte 2
Adicionando buffer duplo para possibilitar retrocesso

---

1. Abrir projeto da demonstração anterior
2. Modificar a classe LeitorDeArquivosTexto
2.1. Adicionar o código do buffer

```
    private final static int TAMANHO_BUFFER = 5;
    int[] bufferDeLeitura;
    int ponteiro;
    private void inicializarBuffer() {
        bufferDeLeitura = new int[TAMANHO_BUFFER * 2];
        ponteiro = 0;
        recarregarBuffer1();
    }
    private int lerCaractereDoBuffer() {
        int ret = bufferDeLeitura[ponteiro];
        incrementarPonteiro();
        return ret;
    }
    private void incrementarPonteiro() {
        ponteiro++;
        if (ponteiro == TAMANHO_BUFFER) {
            recarregarBuffer2();
        } else if (ponteiro == TAMANHO_BUFFER * 2) {
            recarregarBuffer1();
            ponteiro = 0;
        }
    }
    private void recarregarBuffer1() {
        try {
            for (int i = 0; i < TAMANHO_BUFFER; i++) {
                bufferDeLeitura[i] = is.read();
                if (bufferDeLeitura[i] == -1) {
                    break;
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace(System.err);
        }
    }
    private void recarregarBuffer2() {
        try {
            for (int i = TAMANHO_BUFFER; i < TAMANHO_BUFFER * 2; i++) {
                bufferDeLeitura[i] = is.read();
                if (bufferDeLeitura[i] == -1) {
                    break;
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace(System.err);
```

```
        }
    }
```

2.2. Adicionar chamada para inicializar o buffer no construtor (no final, depois da inicialização do stream)

```
            inicializarBuffer();
```

2.3. Modificar o método para ler o próximo caractere

```
    public int lerProximoCaractere() {
        int c = lerCaractereDoBuffer();
        System.out.print((char)c);
        return c;
    }
```

2.4. Adicionar o código para retroceder

```
    public void retroceder() {
        ponteiro--;
        if (ponteiro < 0) {
            ponteiro = TAMANHO_BUFFER * 2 - 1;
        }
    }
```

3. Na classe AlgumaLexico, modificar o código que reconhece operadores, para retrair antes de retornar o Token para o "<"

```
    ldat.retroceder();
```

4. Testar

4.1. Pode dar erro pois pode acontecer dele recarregar o mesmo lado do buffer 2 vezes. Será corrigido no próximo exemplo.

4.1.1. No Mac e Linux, o exemplo atual já causa esse erro

4.1.2. No Windows, precisa acrescentar um ou dois caracteres antes da linha do "ENQUANTO", e tirar um espaço antes do INICIO

Demonstração 3 – Analisador léxico "na mão" – parte 3
Possibilitar a análise de diversos padrões "em cascata"

---

1. Abrir projeto da demonstração anterior
2. Na classe LeitorDeArquivosTexto, fazer as seguintes modificações

```
package algumalex;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class LeitorDeArquivosTexto {
    private final static int TAMANHO_BUFFER = 20;
    int[] bufferDeLeitura;
    int ponteiro;
    int bufferAtual; // Como agora tem retrocesso, é necessário armazenar o
                     // buffer atual, pois caso contrário ao retroceder ele
                     // pode recarregar um mesmo buffer 2 vezes
    int inicioLexema;
    private String lexema;

    private void inicializarBuffer() {
        bufferAtual = 2;
        inicioLexema = 0;
        lexema = "";
        bufferDeLeitura = new int[TAMANHO_BUFFER * 2];
        ponteiro = 0;
        recarregarBuffer1();
    }
    private int lerCaractereDoBuffer() {
        int ret = bufferDeLeitura[ponteiro];
        // System.out.println(this);// descomentar depois
        incrementarPonteiro();
        return ret;
    }
    private void incrementarPonteiro() {
        ponteiro++;
        if (ponteiro == TAMANHO_BUFFER) {
            recarregarBuffer2();
        } else if (ponteiro == TAMANHO_BUFFER * 2) {
            recarregarBuffer1();
            ponteiro = 0;
        }
    }
    private void recarregarBuffer1() {
        if (bufferAtual == 2) {
            bufferAtual = 1;
            try {
                for (int i = 0; i < TAMANHO_BUFFER; i++) {
                    bufferDeLeitura[i] = is.read();
                    if (bufferDeLeitura[i] == -1) {
                        break;
```

```
                    }
                }
            } catch (Exception ex) {
                ex.printStackTrace(System.err);
            }
        }
    }
    private void recarregarBuffer2() {
        if (bufferAtual == 1) {
            bufferAtual = 2;
            try {
                for (int i = TAMANHO_BUFFER; i < TAMANHO_BUFFER * 2; i++) {
                    bufferDeLeitura[i] = is.read();
                    if (bufferDeLeitura[i] == -1) {
                        break;
                    }
                }
            } catch (Exception ex) {
                ex.printStackTrace(System.err);
            }
        }
    }
    InputStream is;
    public LeitorDeArquivosTexto(String arquivo) {
        try {
            is = new FileInputStream(new File(arquivo));
            inicializarBuffer();
        } catch (Exception ex) {
            ex.printStackTrace(System.err);
        }
    }
    public int lerProximoCaractere() {
        int c = lerCaractereDoBuffer();
        lexema += (char) c;
        return c;
    }
    public void retroceder() {
        ponteiro--;
        lexema = lexema.substring(0, lexema.length() - 1);
        if (ponteiro < 0) {
            ponteiro = TAMANHO_BUFFER * 2 - 1;
        }
    }
    public void zerar() {
        ponteiro = inicioLexema;
        lexema = "";
    }
    public void confirmar() {
        System.out.print(lexema); // comentar para ficar melhor a saída
        inicioLexema = ponteiro;
        lexema = "";
```

```
    }
    public String getLexema() {
        return lexema;
    }
    public String toString() {
        String ret = "Buffer:[";
        for (int i : bufferDeLeitura) {
            char c = (char) i;
            if (Character.isWhitespace(c)) {
                ret += ' ';
            } else {
                ret += (char) i;
            }
        }
        ret += "]\n";
        ret += "        ";
        for (int i = 0; i < TAMANHO_BUFFER * 2; i++) {
            if (i == inicioLexema && i == ponteiro) {
                ret += "%";
            } else if (i == inicioLexema) {
                ret += "^";
            } else if (i == ponteiro) {
                ret += "*";
            } else {
                ret += " ";
            }
        }
        return ret;
    }
```

3. Criar o seguinte código na classe AlgumaLexico (deixar o método proximoToken por último)

```
package algumalex;
public class AlgumaLexico {
    LeitorDeArquivosTexto ldat;
    public AlgumaLexico(String arquivo) {
        ldat = new LeitorDeArquivosTexto(arquivo);
    }
    public Token proximoToken() {
        Token proximo = null;
        espacosEComentarios();
        ldat.confirmar();
        proximo = fim();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = palavrasChave();
```

```
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = variavel();
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = numeros();
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = operadorAritmetico();
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = operadorRelacional();
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = delimitador();
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = parenteses();
if (proximo == null) {
    ldat.zerar();
} else {
    ldat.confirmar();
    return proximo;
}
proximo = cadeia();
if (proximo == null) {
    ldat.zerar();
```

```
        } else {
            ldat.confirmar();
            return proximo;
        }
        System.err.println("Erro léxico!");
        System.err.println(ldat.toString());
        return null;
    }
    private Token operadorAritmetico() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == '*') {
            return new Token(TipoToken.OpAritMult, ldat.getLexema());
        } else if (c == '/') {
            return new Token(TipoToken.OpAritDiv, ldat.getLexema());
        } else if (c == '+') {
            return new Token(TipoToken.OpAritSoma, ldat.getLexema());
        } else if (c == '-') {
            return new Token(TipoToken.OpAritSub, ldat.getLexema());
        } else {
            return null;
        }
    }
    private Token delimitador() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == ':') {
            return new Token(TipoToken.Delim, ldat.getLexema());
        } else {
            return null;
        }
    }

    private Token parenteses() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == '(') {
            return new Token(TipoToken.AbrePar, ldat.getLexema());
        } else if (c == ')') {
            return new Token(TipoToken.FechaPar, ldat.getLexema());
        } else {
            return null;
        }
    }
    private Token operadorRelacional() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == '<') {
            c = (char) ldat.lerProximoCaractere();
            if (c == '>') {
                return new Token(TipoToken.OpRelDif, ldat.getLexema());
```

```
            } else if (c == '=') {
                return new Token(TipoToken.OpRelMenorIgual,
ldat.getLexema());
            } else {
                ldat.retroceder();
                return new Token(TipoToken.OpRelMenor, ldat.getLexema());
            }
        } else if (c == '=') {
            return new Token(TipoToken.OpRelIgual, ldat.getLexema());
        } else if (c == '>') {
            c = (char) ldat.lerProximoCaractere();
            if (c == '=') {
                return new Token(TipoToken.OpRelMaiorIgual,
ldat.getLexema());
            } else {
                ldat.retroceder();
                return new Token(TipoToken.OpRelMaior, ldat.getLexema());
            }
        }
        return null;
    }
    private Token numeros() {
        int estado = 1;
        while (true) {
            char c = (char) ldat.lerProximoCaractere();
            if (estado == 1) {
                if (Character.isDigit(c)) {
                    estado = 2;
                } else {
                    return null;
                }
            } else if (estado == 2) {
                if (c == '.') {
                    c = (char) ldat.lerProximoCaractere();
                    if (Character.isDigit(c)) {
                        estado = 3;
                    } else {
                        return null;
                    }
                } else if (!Character.isDigit(c)) {
                    ldat.retroceder();
                    return new Token(TipoToken.NumInt, ldat.getLexema());
                }
            } else if (estado == 3) {
                if (!Character.isDigit(c)) {
                    ldat.retroceder();
                    return new Token(TipoToken.NumReal, ldat.getLexema());
                }
            }
        }
    }
```

```
private Token variavel() {
    int estado = 1;
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (estado == 1) {
            if (Character.isLetter(c)) {
                estado = 2;
            } else {
                return null;
            }
        } else if (estado == 2) {
            if (!Character.isLetterOrDigit(c)) {
                ldat.retroceder();
                return new Token(TipoToken.Var, ldat.getLexema());
            }
        }
    }
}
private Token cadeia() {
    int estado = 1;
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (estado == 1) {
            if (c == '\'') {
                estado = 2;
            } else {
                return null;
            }
        } else if (estado == 2) {
            if (c == '\n') {
                return null;
            }
            if (c == '\'') {
                return new Token(TipoToken.Cadeia, ldat.getLexema());
            } else if (c == '\\') {
                estado = 3;
            }
        } else if (estado == 3) {
            if (c == '\n') {
                return null;
            } else {
                estado = 2;
            }
        }
    }
}
private void espacosEComentarios() {
    int estado = 1;
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (estado == 1) {
```

```
            if (Character.isWhitespace(c) || c == ' ') {
                estado = 2;
            } else if (c == '%') {
                estado = 3;
            } else {
                ldat.retroceder();
                return;
            }
        } else if (estado == 2) {
            if (c == '%') {
                estado = 3;
            } else if (!(Character.isWhitespace(c) || c == ' ')) {
                ldat.retroceder();
                return;
            }
        } else if (estado == 3) {
            if (c == '\n') {
                return;
            }
        }
    }
}
private Token palavrasChave() {
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (!Character.isLetter(c)) {
            ldat.retroceder();
            String lexema = ldat.getLexema();
            if (lexema.equals("DECLARACOES")) {
                return new Token(TipoToken.PCDeclaracoes, lexema);
            } else if (lexema.equals("ALGORITMO")) {
                return new Token(TipoToken.PCAlgoritmo, lexema);
            } else if (lexema.equals("INT")) {
                return new Token(TipoToken.PCInteiro, lexema);
            } else if (lexema.equals("REAL")) {
                return new Token(TipoToken.PCReal, lexema);
            } else if (lexema.equals("ATRIBUIR")) {
                return new Token(TipoToken.PCAtribuir, lexema);
            } else if (lexema.equals("A")) {
                return new Token(TipoToken.PCA, lexema);
            } else if (lexema.equals("LER")) {
                return new Token(TipoToken.PCLer, lexema);
            } else if (lexema.equals("IMPRIMIR")) {
                return new Token(TipoToken.PCImprimir, lexema);
            } else if (lexema.equals("SE")) {
                return new Token(TipoToken.PCSe, lexema);
            } else if (lexema.equals("ENTAO")) {
                return new Token(TipoToken.PCEntao, lexema);
            } else if (lexema.equals("ENQUANTO")) {
                return new Token(TipoToken.PCEnquanto, lexema);
            } else if (lexema.equals("INICIO")) {
```

```
                    return new Token(TipoToken.PCInicio, lexema);
                } else if (lexema.equals("FIM")) {
                    return new Token(TipoToken.PCFim, lexema);
                } else if (lexema.equals("E")) {
                    return new Token(TipoToken.OpBoolE, lexema);
                } else if (lexema.equals("OU")) {
                    return new Token(TipoToken.OpBoolOu, lexema);
                } else {
                    return null;
                }
            }
        }
    }
    private Token fim() {
        int caractereLido = ldat.lerProximoCaractere();
        if (caractereLido == -1) {
            return new Token(TipoToken.Fim, "Fim");
        }
        return null;
    }
}
```

4. Mudar o método no void main para parar ao encontrar o token Fim

```
        Token t = null;
        while((t=lex.proximoToken()).nome != TipoToken.Fim) {
            System.out.println(t);
        }
```

5. Executar e mudar o arquivo de entrada para testar

Demonstração 4 – Analisador léxico "na mão" – parte 4
Construindo uma tabela de símbolos

---

1. Abrir projeto da demonstração anterior
2. Criar enum TipoEntradaTabelaDeSimbolos

```
package algumalex;
public enum TipoEntradaTabelaDeSimbolos {
    Variavel, PalavraChave
};
```

3. Criar classe EntradaTabelaDeSimbolos

```
package algumalex;
public class EntradaTabelaDeSimbolos {
    public String nome;
    public TipoEntradaTabelaDeSimbolos tipoEntrada;
    public EntradaTabelaDeSimbolos(String nome, TipoEntradaTabelaDeSimbolos
tipoEntrada) {
        this.nome = nome;
        this.tipoEntrada = tipoEntrada;
    }
}
```

4. Criar classe TabelaDeSimbolos

```
package algumalex;
import java.util.ArrayList;
import java.util.List;
public class TabelaDeSimbolos {
    private static TabelaDeSimbolos instancia;
    public static TabelaDeSimbolos getTabelaDeSimbolos() {
        if (instancia == null) {
            instancia = new TabelaDeSimbolos();
        }
        return instancia;
    }
    private List<EntradaTabelaDeSimbolos> tabela;
    public TabelaDeSimbolos() {
        tabela = new ArrayList<EntradaTabelaDeSimbolos>();
    }
    public EntradaTabelaDeSimbolos getEntrada(int indice) {
        return tabela.get(indice);
    }
    public int instalarOuEncontrarSimbolo(String lexema,
TipoEntradaTabelaDeSimbolos tipo) {
        int ret = encontrarSimbolo(lexema);
        if(ret != -1)
            return ret;
        else {
            return instalarNovoSimbolo(lexema, tipo);
```

```
        }
    }
    private int instalarNovoSimbolo(String nome, TipoEntradaTabelaDeSimbolos
tipo) {
        tabela.add(new EntradaTabelaDeSimbolos(nome, tipo));
        return tabela.size() - 1;
    }
    private int encontrarSimbolo(String nome) {
        int entrada = 0;
        for (EntradaTabelaDeSimbolos etds : tabela) {
            if (etds.nome.equals(nome)) {
                return entrada;
            }
            entrada++;
        }
        return -1;
    }
    @Override
    public String toString() {
        String ret = "";
        int entrada = 0;
        for (EntradaTabelaDeSimbolos etds : tabela) {
            ret += entrada + ":" + etds.nome + "("+etds.tipoEntrada+")\n";
            entrada++;
        }
        return ret;
    }
}
```

5. Modificar o arquivo AlgumaLexico

```
    private Token variavel() {
        int estado = 1;
        while (true) {
            char c = (char) ldat.lerProximoCaractere();
            if (estado == 1) {
                if (Character.isLetter(c)) {
                    estado = 2;
                } else {
                    return null;
                }
            } else if (estado == 2) {
                if (!Character.isLetterOrDigit(c)) {
                    ldat.retroceder();
                    int indiceTS =
TabelaDeSimbolos.getTabelaDeSimbolos().instalarOuEncontrarSimbolo(ldat.getLex
ema(), TipoEntradaTabelaDeSimbolos.Variavel);
                    return new Token(TipoToken.Var, ldat.getLexema(),
indiceTS);
                }
            }
```

```
        }
    }
```

6. Adicionar, na classe Main, o código para imprimir a tabela de símbolos (no final)

```
        System.out.println("------------");
        System.out.println(TabelaDeSimbolos.getTabelaDeSimbolos());
```

7. Testar
8. Modificar o enum TipoToken, para remover as palavras chave (e os operadores booleanos)

```
package algumalex;
public enum TipoToken {
    PalavraChave,
    OpAritMult, OpAritDiv, OpAritSoma, OpAritSub,
    OpRelMenor, OpRelMenorIgual, OpRelMaiorIgual,
    OpRelMaior, OpRelIgual, OpRelDif,
    Delim,
    AbrePar, FechaPar,
    Var,
    NumInt, NumReal,
    Cadeia,
    Fim
}
```

9. Na classe AlgumaLexico, remover o metodo palavrasChave(), criar o método instalarPalavrasChave e chamar no construtor

```
    public AlgumaLexico(String arquivo) {
        ldat = new LeitorDeArquivosTexto(arquivo);
        instalarPalavrasChave();
    }

    ...

    private void instalarPalavrasChave() {
        TabelaDeSimbolos ts = TabelaDeSimbolos.getTabelaDeSimbolos();
        ts.instalarOuEncontrarSimbolo("DECLARACOES",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("ALGORITMO",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("INT",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("REAL",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("ATRIBUIR",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("A",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("LER",
TipoEntradaTabelaDeSimbolos.PalavraChave);
```

```
        ts.instalarOuEncontrarSimbolo("IMPRIMIR",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("SE",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("ENTAO",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("ENQUANTO",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("INICIO",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("FIM",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("E",
TipoEntradaTabelaDeSimbolos.PalavraChave);
        ts.instalarOuEncontrarSimbolo("OU",
TipoEntradaTabelaDeSimbolos.PalavraChave);
    }
```

10. Transformar o método variavel() em variavelOuPalavraChave()

```
    private Token variavelOuPalavraChave() {
        int estado = 1;
        while (true) {
            char c = (char) ldat.lerProximoCaractere();
            if (estado == 1) {
                if (Character.isLetter(c)) {
                    estado = 2;
                } else {
                    return null;
                }
            } else if (estado == 2) {
                if (!Character.isLetterOrDigit(c)) {
                    ldat.retroceder();
                    int indiceTS =
TabelaDeSimbolos.getTabelaDeSimbolos().instalarOuEncontrarSimbolo(ldat.getLex
ema(), TipoEntradaTabelaDeSimbolos.Variavel);
                    EntradaTabelaDeSimbolos etds =
TabelaDeSimbolos.getTabelaDeSimbolos().getEntrada(indiceTS);
                    if(etds.tipoEntrada ==
TipoEntradaTabelaDeSimbolos.Variavel)
                        return new Token(TipoToken.Var, ldat.getLexema(),
indiceTS);
                    else return new Token(TipoToken.PalavraChave,
ldat.getLexema(), indiceTS);
                }
            }
        }
    }
```

11. Modificar o método proximoToken() para fazer a chamada correta dos demais métodos

```
    public Token proximoToken() {
        Token proximo = null;
        espacosEComentarios();
        ldat.confirmar();
        proximo = fim();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = variavelOuPalavraChave();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = numeros();
        if (proximo == null) {
...
```

12. Apagar o método palavraChave(), e sua chamada no proximoToken()
13. Testar (fazer o teste com ENTAOAO para mostrar que prefere sempre a maior cadeia)
14. Descomentar a linha do LeitorDeArquivoTexto onde imprime o estado atual de leitura, para mostrar o buffer sendo utilizado
    14.1. No método proximoToken(), antes de cada chamada, colocar um print de qual padrão será testado:

```
    public Token proximoToken() {
        Token proximo = null;
        System.out.println("Tentando espaços e comentários ...");
        espacosEComentarios();
        ldat.confirmar();
        System.out.println("Tentando fim de arquivo ...");
        proximo = fim();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        System.out.println("Tentando variável ou palavra-chave ...");
        proximo = variavelOuPalavraChave();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        ...
```

Demonstração 5 – Analisador léxico com ANTLR

---

1. Mostrar o site do ANTLR (www.antlr.org)
   1.1. Baixar o ANTLR (complete ANTLR Java binaries jar)
2. Abrir o NetBeans e criar novo projeto Java: CompiladorAlguma
3. Criar um novo arquivo do tipo ANTLR Lexer Grammar, chamado AlgumaLexer.g4 (dentro da pasta src, pacote compiladoralguma)

```
lexer grammar AlgumaLexer;

Letra       :     'a'..'z' | 'A'..'Z';
Digito:     '0'..'9';
Variavel    :     Letra(Letra|Digito)* {
System.out.print("[Var,"+getText()+"]");};
```

4. Mandar gerar o reconhecedor
   4.1. Rodar o seguinte comando pelo terminal, dentro da pasta onde está o arquivo .g4

```
java -jar antlr-xxx-complete.jar -package compiladoralguma AlgumaLexer.g4
```

5. Adicionar biblioteca do antlrworks no projeto
   5.1. Clicar com botão direito sobre o projeto -> Properties -> Libraries
   5.2. Adicionar o jar do antlr (antlr-xxx-complete.jar)
6. No método principal, criar o seguinte código (adicionar os imports)

```
        try {
            ANTLRInputStream ais = new ANTLRInputStream(new
FileInputStream(<caminho do arquivo>));
            AlgumaLexer lex = new AlgumaLexer(ais);
            while (lex.nextToken().getType() != Token.EOF) {
                System.out.println("");
            }
        } catch (IOException ex) { }
```

7. Testar

Demonstração 6 – Expressões regulares no antlr

---

1. Abrir no NetBeans o projeto da demonstração anterior
2. Modificar a gramática AlgumaLexer

```
lexer grammar AlgumaLexer;

PALAVRA_CHAVE
      :     'DECLARACOES' | 'ALGORITMO' | 'INT' | 'REAL' | 'ATRIBUIR' | 'A'
| 'LER' | 'IMPRIMIR' | 'SE' | 'ENTAO'
      | 'ENQUANTO' | 'INICIO' | 'FIM' | 'E' | 'OU'
      ;
NUMINT: ('+'|'-')?('0'..'9')+
      ;
NUMREAL     : ('+'|'-')?('0'..'9')+ ('.' ('0'..'9')+)?
      ;
VARIAVEL : ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9')*
       ;
CADEIA      : '\'' ( ESC_SEQ | ~('\''|'\\') )* '\''
      ;
fragment
ESC_SEQ     : '\\\'';
COMENTARIO
    :   '%' ~('\n'|'\r')* '\r'? '\n' {skip();}
    ;
WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        ) {skip();}
    ;
OP_REL:     '>' | '>=' | '<' | '<=' | '<>' | '='
      ;
OP_ARIT     :     '+' | '-' | '*' | '/'
      ;
DELIM :     ':'
      ;
ABREPAR :   '('
      ;
FECHAPAR:   ')'
      ;
```

3. Modificar o método principal

```
        try {
            ANTLRInputStream ais = new ANTLRInputStream(new
FileInputStream(<caminho do arquivo>));
            AlgumaLexer lex = new AlgumaLexer(ais);

            Token t = null;
            while ((t = lex.nextToken()).getType() != Token.EOF) {
```

```
                System.out.println("<"+t.getType() + "," + t.getText()+">");
            }


        } catch (IOException ex) {
        }
```

4. Testar e rodar
5. Tentar colocar a regra de palavras-chave depois da regra de variáveis e testar novamente
  5.1. Observar o resultado, e ver que agora todas as palavras-chave são reconhecidas como variáveis
6. Mostrar os exemplos de regras gananciosas/não gananciosas, para o caso das cadeias
  6.1. Usar a seguinte regra para cadeia

```
CADEIA: '\'' .* '\''
            ;
```

  6.2. Mostrar que ele gera warning
  6.3. Modificar o arquivo de entrada, para ter uma cadeia

```
LER 'abcd' argumento 'defg'
ATRIBUIR argumento A fatorial 'hij'
```

  6.4. Usar o operador não-ganancioso

```
CADEIA: '\'' .*? '\''
            ;
```

  6.5. Testar de novo
  6.6. Tentar outra regra gananciosa

```
CADEIA      : '\'' ( ~('\n') )* '\''
            ;
```

  6.7. Mostrar que agora nem deu warning (era por causa do wildcard)
  6.8. Fazer agora a versão não-gananciosa

```
CADEIA      : '\'' ( ~('\n') )*? '\''
            ;
```

  6.9. Testar de novo