Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática
Processamento de Linguagens

FleCa
Análise de código - Processamento de comentários

Célia Figueiredo (a67637)
Tiago Cunha (a67707)
Xavier Neves Francisco (a67725)

Ano letivo de 2014/15

2 de Abril de 2015

Conteúdo

Resumo

O presente relatório foi desenvolvido no âmbito da Unidade Curricular de Processamento de Linguagens e tem como principal objetivo descrever o processo de desenvolvimento de um analisador léxico através do *FLEX*, usando expressões regulares para ler um programa fonte em **Java**. O produto final obtido consiste num ficheiro *HTML* que organiza todos os comentários.

Palavras chave: Flex, Expressões Regulares, HTML, Java

1 Introdução

Os comentários, emboram não façam nada no programa em si, permitem que qualquer um intreprete o código descrito. Por esta razão escolhemos o enunciado 2.3, pois este trabalho permite ser utilizado no nosso dia-a-dia a programar. Para além disso poderá ainda haver uma comparação entre os geradores de documentação já existentes, como o *Javadoc* e o *Doxigen*. Neste contexto, e de forma a colocar em prática os conhecimentos adquiridos até ao momento nas aulas teórico-práticas da Unidade Curricular, fizemos uso da ferramenta Flex - The Fast Lexical Analyser. Neste trabalho será necessário implementar Expressões Regulares de modo a retirar informação relevante, neste caso todos os comentários que aparecem num ficheiro *.java*. Para isso tentamos definir da melhor forma possivel as expressões regulares utilizadas, no sentido de obter todos os comentaŕios do ficheiro, bem como os autores e versões.

2 Contextualização, descrição e implementação do problema

2.1 Contextualização

Flex é uma ferramenta para gerar automaticamente analisadores léxicos, isto é, programas que reconhecem padrões léxicos num texto. O Flex é uma evolução da ferramenta Lex, mas com a caraterística de ser mais rápido que este (Fast Lex). Lex foi desenvolvido por M.E. Lesk e E. Shmidt (Bell Laboratories - ATT) enquanto que o Flex é um produto da Free Software Foundation, Inc. Ao contrário do programador que escreve manualmente um programa que realize a identificação de padrões numa entrada, o uso do Flex/Lex permite que sejam apenas especificados os padrões desejados e as ações necessárias para processá-los. Para que Flex/Lex reconheçam padrões no texto, tais padrões devem ser descritos através de expressões regulares. A ferramenta Flex é ótima para a realização deste trabalho uma vez que nos permite analisar o conteúdo específico, através de expressões regulares de determinada informação de texto.

2.2 Descrição do Problema e Implementação

O problema escolhido foi o de análise de código JAVA, este tem o objetivo de processar os comentários introduzidos pelos autores, assim como as versões.
Os comentários, como o próprio nome indica são notas que podem ser incluídas no código

fonte para descrever o que se quiser. Assim, não modificam o programa executado e servem somente para ajudar o programador a melhor organizar os seus códigos.
Para resolver-mos este problema começamos por implementar um ficheiro HTML, com o intuito de nos guiarmos, sendo ele constituido por links para os diversos tipos de comentários existentes.
De seguida mostramos o website:

Figura 1: Imagem alusiva a um ficheiro java onde estão presentes comentários

Aqui mostramos os **comentário inline**, este que são constituidos apenas por uma linha.

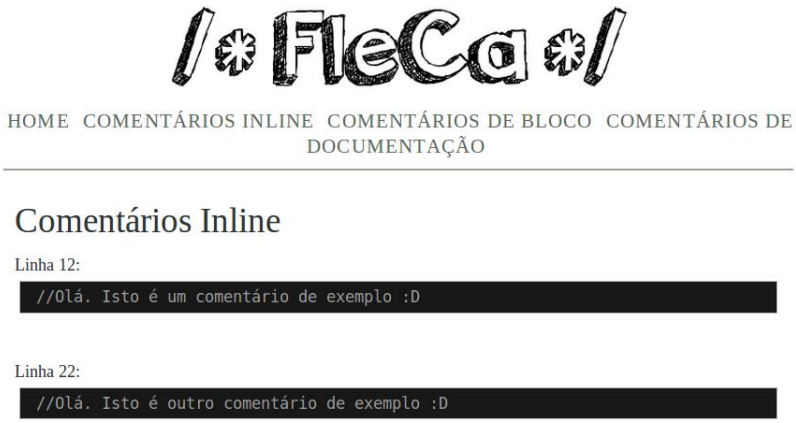

Figura 2: Exemplo de comentários inline

Comentários em Bloco:

Figura 3: Exemplos de comentários em bloco

Comentários por Documentação:

Figura 4: Exemplo de comentários por documentação

3 Uso de expressões regulares para processamento de comentários estilo JAVA

Para processar o ficheiro *.java* e com o objetivo de se obter os resultados esperados recorrendo ao analisador léxico *FLEX*, nos pontos seguintes abordamos a forma como foram definidas as expessões regulares, os estados FLEX assim como as estruturas.

3.1 Estrutura de dados

Para a criação dos ficheiros *HTML*, é necessário aramazenar determinados dados, de maneira que, após o processamento do ficheiro de entrada, seja possível imprimir esses dados para os ficheiros finais respectivos. Desta forma, é necessária uma estrutura de dados capaz de suportar estes dados e permitir o seu acesso rápido e eficiente. Portanto, foi considerada uma implementação de listas ligadas para guardar os dados referentes aos comentários do ficheiro *.java*.

3.2 Expressões Regulares

As expressões regulares definidas tratam um ficheiro *.java* com o objetivo de obter as informações necessárias para a construção de uma página *HTML*, neste caso com todo o tipo de comentários do documento.

3.2.1 Strings

Estas expressões regulares servem para ignorar todas as strings que possam dar false -positives, ou seja se houver alguma sequencia que normalmente inicie um comentário dentro de uma string, não vai contar como um comentário, como é suposto.

```
[^\\]\"                    { BEGIN(str);}

<str>\"                    { BEGIN(INITIAL);}

<str>\n                    { BEGIN(INITIAL);}

<str>[^\n\"]+              { ;}
```

3.2.2 Comentários *inline*

Começamos por implementar no ficheiro flex, as expressões que nos permitem retirar apenas os comentários com uma linha que contém os caracteres //.

```
^.*"//".*$              {comment = make_inline_comment(yytext, yylineno);
                         append_inline(&inline_linked_list, comment);
                         n_of_inlines++;
                        }
```

3.2.3 Comentários em blocos

Para abrangir os comentários que vão desde "/*"até "*/"e verificar se o comentário está em inglês ou português. Começamos por definir a primeira expressão regular que deteta a entrada num comentário em bloco, iniciando o estado *block* e inicializando o estrutura *block_comment*.
A segunda para quando o comentário acaba, voltando ao estado *INITIAL*, e completando os campos da estrutura, a linguagem (detetada pela função *detect_lang*), a linha final e o comentário em si.
O último vai detetar o comentário em si consumindo todos os caracteres.

```
\/\*                       { BEGIN(block);
                             n_of_blocks++;
                             bcomment = malloc(sizeof(block_comment));
                             bcomment -> initial_line = yylineno;
                             current = NULL;
                            }

<block>\*\/                { BEGIN(INITIAL);
                             bcomment -> final_line = yylineno;
                             bcomment -> string = strdup(current);
                             bcomment -> language = detect_lang(bcomment -> string);
                             append_block(&blocks_linked_list, bcomment);
                             free(bcomment);
                             free(current);
                            }
```

```
<block>\*[^\/]              { ;}

<block>[^\*]*               { current = str_append(current, yytext);}
```

3.2.4 Comentários por documentação

Para abrangir os comentários que vão desde "/**"até "*/"cujas linhas começam por um "*"e podem ter texto livre ou podem ter comandos para criar documentação técnica tais como *@author*. Começamos por definir a primeira expressão regular que deteta a entrada num comentário por documentação, iniciando o estado *docs* e inicializando o estrutura *docs_comment*.

A segunda para quando o comentário acaba, voltando ao estado *INITIAL*, e adicionando o comentário em si.

A terceira e a quarta vão detectar documentação técnica com os campos *@author* e *@version* respectivamente, para preencher os campos da estrutura.

Por fim, como nos blocos a última expressão regular vai consumindo o comentário e adicionando-o à string current, com o str_append (função que adiciona duas strings dinâmicas).

```
\/\*{2,}                    { BEGIN(docs);
                              n_of_docs++;
                              dcomment = malloc(sizeof(docs_comment));
                              current = NULL;
                            }

<docs>\*\/                  { BEGIN(INITIAL);
                              dcomment -> string = strdup(current);
                              append_docs(&docs_linked_list, dcomment);
                              free(dcomment);
                              free(current);
                            }

<docs>"@author ".*\n        { dcomment -> author  = strdup(yytext + 8);
                              current = str_append(current, yytext); }

<docs>"@version ".*\n       { dcomment -> version = strdup(yytext + 9);
                              current = str_append(current, yytext); }

<docs>.|\n                  { current = str_append(current, yytext);}
```

4 Código teste e output

Nesta parte do relatório será mostrado o nosso ficheiro de teste assim como o respetivo output.

4.1 Ficheiro teste

Ficheiro Rectangulo.java

```
1  public class Rectangulo {
2
3      class Point {
4
5          private double x;
6          private double y;
7
8          public Point(){
9              this.x = 0;
10             this.y = 0;
11         }
12
13         public Point(Point p){
14             this.x = p.getX();
15             this.y = p.getY();
16         }
17
18         public Point(double x, double y){
19             this.x = x;
20             this.y = y;
21         }
22
23         public double distanceTo(Point p){
24             return Math.sqrt(Math.pow(this.x - p.getX(), 2) + Math.
                   pow(this.y - p.getY(), 2)); // Formula da distancia
25         }
26
27         public void move(double dx, double dy){
28             this.x += dx;
29             this.y += dy;
30         }
31
32         public double getX(){
33             return this.x;
34         }
35         public double getY(){
36             return this.y;
37         }
38         public void setX(double x) {
39             this.x = x;
40         }
```

```
41          public void setY(double y) {
42              this.y = y;
43          }
44
45          public Point clone(){
46              return new Point(x, y);
47          }
48
49          public String toString(){
50              return "(" + this.x + ", " + this.y + ")"; // A
                    representa ao de um ponto e: "(x, y)"
51          }
52
53          public boolean equals(Object o) {
54              if (this == o){ // Se a referencia e a mesma, as
                    instancias sao a mesmo
55                  return true;
56              }
57              /*
58              Se as classes forem diferentes, entao as instancias sao
                    diferentes
59              Se o "o" for null tamb m e diferente, porque tudo e
                    diferente de null em Java.
60              */
61              if ((o == null) || (this.getClass() != o.getClass())){
62                  return false;
63              }
64              Point point = (Point) o;
65
66              return point.getX() == this.x && point.getY() == this.y;
67          }
68
69          public int hashCode(){
70          /*
71          Retirado do wikipedia: http://en.wikipedia.org/wiki/
                Java_hashCode()
72
73          In the Java programming language, every class implicitly or
                explicitly provides a hashCode() method, which digests the
                 data stored in an instance of the class into a single
                hash value (a 32-bit signed integer).
74          This hash is used by other code when storing or manipulating
                the instance  the values are intended to be evenly
                distributed for varied inputs in order to use in
                clustering.
75          This property is important to the performance of hash tables
                and other data structures that store objects in groups ("
                buckets") based on their computed hash values.
76          Technically, in Java, hashCode() by default is a native
```

```
            method, meaning, it has the modifier 'native', as it is
            implemented directly in the native code in the JVM.
77          */
78              int hash = 7;
79              hash = hash * 31 + Double.hashCode(this.x);
80              hash = hash * 31 + Double.hashCode(this.y);
81              return hash;
82          }
83
84
85      }
86
87      private Point p1;
88      private Point p2;
89      private Point p3;
90      private Point p4;
91
92      /**
93       * Construtor por defeito.
94       *
95       * @author Celia Figueiredo
96       * @version 2014.03.11
97       *
98       */
99      Rectangulo(){
100         this.p1 = new Point(); this.p2 = new Point();
101         this.p3 = new Point(); this.p4 = new Point();
102     }
103
104     /**
105      * Construtor de copia.
106      *
107      * @author Xavier Francisco
108      * @version 2014.03.10
109      *
110      */
111     Rectangulo(Rectangulo r){
112         this.p1 = r.getP1(); this.p2 = r.getP2();
113         this.p3 = r.getP3(); this.p4 = r.getP4();
114     }
115
116     /**
117      * Construtor parametrizado (Recebe as coordenadas dos pontos).
118      *
119      * @author Tiago Cunha
120      * @version 2014.03.12
121      *
122      */
123     Rectangulo(double x1, double y1, double x2, double y2) {
```

```
124         this.p1 = new Point(x1, y1); this.p2 = new Point(x2, y1);
125         this.p3 = new Point(x1, y2); this.p4 = new Point(x2, y2);
126     }
127 
128     /**
129      * Construtor parametrizado (Recebe instancias de Pontos).
130      *
131      * @author Celia Figueiredo
132      * @version 2014.03.12
133      *
134      */
135     Rectangulo(Point p1, Point p4) {
136         this.p1 = p1;                                   this.p2 = new
               Point(p4.getX(), p1.getY());
137         this.p3 = new Point(p1.getX(), p4.getY()); this.p4 = p4;
138     }
139 
140     public double baseLength(){
141         return p1.distanceTo(p2);
142     }
143     public double diagonalLength(){
144         return p1.distanceTo(p4);
145     }
146     public double verticalLength(){
147         return p1.distanceTo(p3);
148     }
149 
150     public double area(){
151         return verticalLength()*baseLength();
152     }
153     public double perimeter(){
154         return 2*(verticalLength()+baseLength());
155     }
156 
157     public void move(double dx, double dy){
158         this.p1.move(dx, dy);
159         this.p2.move(dx, dy);
160         this.p3.move(dx, dy);
161         this.p4.move(dx, dy);
162     }
163 
164     public Rectangulo clone(){
165         return new Rectangulo(p1.clone(), p4.clone());
166     }
167 
168     public String toString(){
169         return p1.toString() + p4.toString();
170     }
171 
```

```
172     public boolean equals(Object o) {
173         if (this == o){
174             return true;
175         }
176         if ((o == null) || (this.getClass() != o.getClass())){
177             return false;
178         }
179         Rectangulo r = (Rectangulo) o;
180 
181         return this.p1.equals(r.getP1()) &&
182                this.p2.equals(r.getP2()) &&
183                this.p3.equals(r.getP3()) &&
184                this.p4.equals(r.getP4());
185     }
186 
187     public int hashCode(){
188         int hash = 7;
189         hash = hash * 31 + p1.hashCode();
190         hash = hash * 31 + p2.hashCode();
191         hash = hash * 31 + p3.hashCode();
192         hash = hash * 31 + p4.hashCode();
193         return hash;
194     }
195 
196 
197     public Point getP1() {
198         return p1.clone();
199     }
200     public Point getP2() {
201         return p2.clone();
202     }
203     public Point getP3() {
204         return p3.clone();
205     }
206     public Point getP4() {
207         return p4.clone();
208     }
209 
210     public void setP1(Point p1) {
211         this.p1 = p1;
212     }
213     public void setP2(Point p2) {
214         this.p2 = p2;
215     }
216     public void setP3(Point p3) {
217         this.p3 = p3;
218     }
219     public void setP4(Point p4) {
220         this.p4 = p4;
```

```
221         }
222
223 }
```

4.2 Output do ficheiro

Output do ficheiro teste em HTML

Figura 5: Página Home com os dados do ficheiro teste

Figura 6: Página dos Comentários Inline

Figura 7: Página dos Comentários de bloco

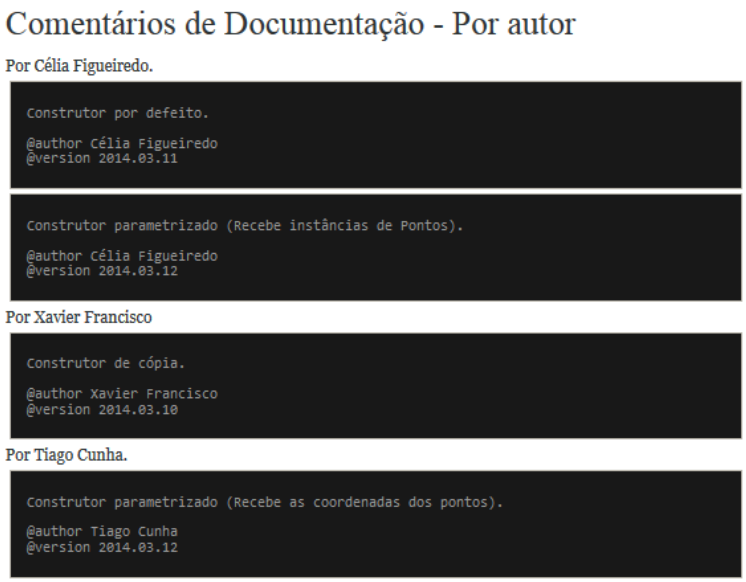

Figura 8: Página dos Comentários de documentação - Por nome

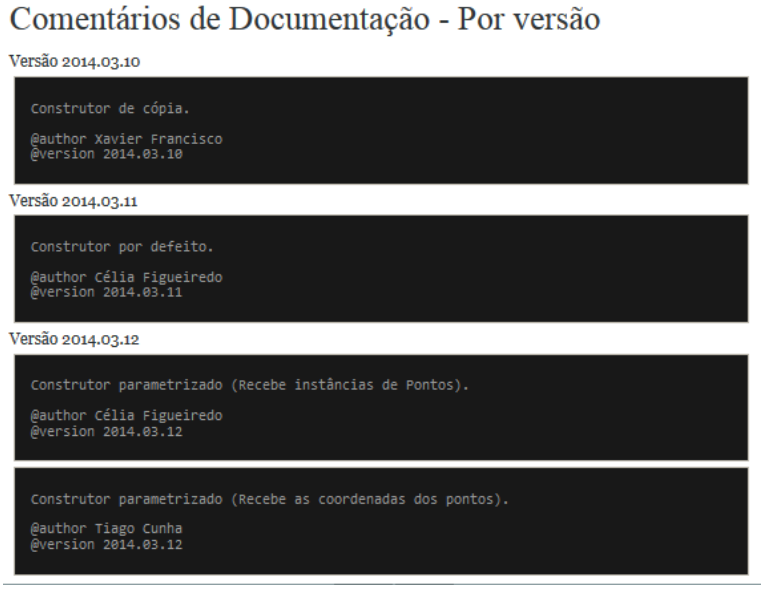

Figura 9: Página dos Comentários de documentação - Por versão

5 Conclusão

Em modo de conclusão, vemos este projeto como mais um forma de pôr em prática os conhecimentos adquiridos e as ferramentas disponibilizadas na Unidade Curricular de Processamento de Linguagens. Como o enunciado deste trabalho prático implica um domínio de todos os conceitos abordados nas aulas, este ajudou a consolidar os conhecimentos e a combater algumas dificuldades que tinhamos anteriormente. Para além de ter sido um desafio implementar um programa de processamento de ficheiros *.java*, foi também aliciante concluir o trabalho. Ao longo do desenvolvimento do projeto o grupo desenvolveu competências em diversas ferramentas como C, HTML e Flex. Os resultados obtidos foram positivos pois conseguiram fazer com sucesso o que era pedido no enunciado do projeto, e tem a mais valia que esta ferramenta poderá ser útil no futuro ao grupo.

6 Anexos

6.1 Código do programa: Flex

```
%{
#include <stdio.h>
#include "inline_struts.h"
#include "blocks_struts.h"
#include "docs_struts.h"
#include "utils.h"

inline_ll* inline_linked_list = NULL;
inline_comment* icomment;

char* current;

blocks_ll* blocks_linked_list = NULL;
block_comment* bcomment;

docs_ll* docs_linked_list = NULL;
docs_comment* dcomment;

unsigned int n_of_inlines = 0;
unsigned int n_of_blocks = 0;
unsigned int n_of_docs = 0;

%}


%x block docs str
%%

    /**
    This regex below tests for strings.
    We want to ignore all strings, so there's no false comments(for example the follow
    */

[^\\]\"                    { BEGIN(str);}

<str>\"                    { BEGIN(INITIAL);}

<str>\n                    { BEGIN(INITIAL);}

<str>[^\n\"]+              { ;}
```

```
  /* First regex to match comments. In this case documentating comments. */

\/\*{2,}                    { BEGIN(docs);
                              n_of_docs++;
                              dcomment = malloc(sizeof(docs_comment));
                              current = NULL;
                            }

<docs>\*\/                  { BEGIN(INITIAL);
                              dcomment -> string = strdup(current);
                              append_docs(&docs_linked_list, dcomment);
                              free(dcomment);
                              free(current);
                            }

<docs>"@author ".*\n        { dcomment -> author  = strdup(yytext + 8);
                              current = str_append(current, yytext); }

<docs>"@version ".*\n       { dcomment -> version = strdup(yytext + 9);
                              current = str_append(current, yytext); }

<docs>.|\n                  { current = str_append(current, yytext);}


\/\*                        { BEGIN(block);
                              n_of_blocks++;
                              bcomment = malloc(sizeof(block_comment));
                              bcomment -> initial_line = yylineno;
                              current = NULL;
                            }

<block>\*\/                 { BEGIN(INITIAL);
                              bcomment -> final_line = yylineno;
                              bcomment -> string = strdup(current);
                              bcomment -> language = detect_lang(bcomment -> string);
                              append_block(&blocks_linked_list, bcomment);
                              free(bcomment);
                              free(current);
                            }

<block>\*[^\/]              { ;}

<block>[^\*]*               { current = str_append(current, yytext);}
```

```
 /*
     Next inline comments
 */

^.*"//".*\$                 { icomment = make_inline_comment(yytext, yylineno);
                              append_inline(&inline_linked_list, icomment);
                              free(icomment);
                              n_of_inlines++;
                            }


.|\n                        {;}

%%


int yywrap(){
    return(1);
}

int main(){

    yylex();

    print_html(inline_linked_list, n_of_inlines, blocks_linked_list, n_of_blocks, docs

    return 0;
}
```

6.2 Código do programa: Estruturas linha

Ficheiro inline.h

```
1 #include <string.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 typedef struct {
6     char *string;
7     int line;
8 } inline_comment;
9
10 typedef struct node_ll_inline{
11     struct node_ll_inline *next;
12     inline_comment    comment;
13 } inline_ll;
14
15
```

```
16 inline_comment* make_inline_comment(char* str, int line);
17
18 void append_inline(inline_ll** linked_list, inline_comment* c);
19
20 inline_comment* pop_inline(inline_ll** linked_list);
```

6.3 Código do programa: Estruturas bloco

Ficheiro blocks.h

```
1 #include <string.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 typedef struct {
6     char *string;
7     int initial_line;
8     int final_line;
9     int language;
10 } block_comment;
11
12 typedef struct node_ll_blocks{
13     struct node_ll_blocks *next;
14     block_comment          comment;
15 } blocks_ll;
16
17
18
19 void append_block(blocks_ll** linked_list, block_comment* c);
20
21 block_comment* pop_block(blocks_ll** linked_list);
22
23 int detect_lang(char* s);
```

6.4 Código do programa: Estruturas documentação

Ficheiro docs.h

```
1 #include <string.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 typedef struct {
6     char *string;
7     char *author;
8     char *version;
9 } docs_comment;
10
11 typedef struct node_ll_docs{
```

```
12     struct node_ll_docs *next;
13     docs_comment          comment;
14 } docs_ll;
15
16
17 void append_docs(docs_ll** linked_list, docs_comment* c);
18
19 docs_comment* pop_docs(docs_ll** linked_list);
20
21 docs_ll* sort_by_authors(docs_ll* linked_list);
22
23 docs_ll* sort_by_version(docs_ll* linked_list);
```

6.5 Código do programa: Utils

Ficheiro utils.h

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include <stdio.h>
4
5 char* str_append(char* fst, char* snd);
```

6.6 Código do programa: Makefile

Ficheiro makefile

```
1 all:
2         flex fleca.l && gcc lex.yy.c inline_struts.c blocks_struts.c
             docs_struts.c print_html.c utils.c -o fleca
3
4 debug:
5         flex fleca.l && gcc --debug lex.yy.c inline_struts.c
             blocks_struts.c docs_struts.c print_html.c utils.c -o
             fleca
6
7 clean:
8         rm fleca lex.yy.c a.out *.html *.o
```