

Processamento de Linguagens e Compiladores (3º ano de LCC)

**Pré-processador para HTML**

TP2

Grupo 14

Artur Queiroz  
A77136

Rafael Fernandes  
A78242

Rafaela Pinho  
A77293

20 de Janeiro de 2018

## **Resumo**

Neste relatório será apresentado as ideias implementadas para criar um pré-processador de HTML através da ferramenta *Flex*. Também descrevemos as decisões tomadas e as dificuldades encontradas, bem como pequenos exemplos de forma que qualquer pessoa que leia este relatório perceba facilmente como funciona o nosso projeto.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Pré-Processador</b>	<b>3</b>
2.1	Descrição informal do problema . . . . .	3
2.2	Especificação dos requisitos . . . . .	3
2.3	Expressões regulares . . . . .	3
<b>3</b>	<b>Codificação e Testes</b>	<b>5</b>
3.1	Problemas de implementação, Decisões e Alternativas . . . . .	5
3.1.1	Problemas de implementação . . . . .	5
3.1.2	Decisões . . . . .	5
3.1.3	Alternativas . . . . .	5
3.2	Testes realizados e Resultados . . . . .	6
<b>4</b>	<b>Conclusão</b>	<b>7</b>
<b>A</b>	<b>Figuras</b>	<b>8</b>
<b>B</b>	<b>Código do Pré-Processador</b>	<b>10</b>

# Capítulo 1

## Introdução

Escrever um documento em HTML trona-se muito cansativo, devido ao peso das "tags" que são inseridas para anotar o texto. Por exemplo, para colocar alguma coisa em negrito é necessário fazer: `< b >(texto que queremos a negrito)< /b >`.

Por isso existem pré-processadores de HTML que facilitam a tarefa da inserção dessas "tags", pois permitem ao utilizador usar anotações mais leves e mais simples. Depois, o pré-processador substitui a notação abreviada para a notação de HTML.

No nosso trabalho construímos um pré-processador que ajuda a simplificar a escrita do código HTML.

# Capítulo 2

## Pré-Processador

### 2.1 Descrição informal do problema

Neste trabalho é necessário:

- i) Criar marcas menos pesadas que são inseridas para anotar o texto.
- ii) Através do Flex construir o processador.
- iii) Passar um ficheiro através do processador para *HTML*.

### 2.2 Especificação dos requisitos

Os requisitos para este trabalho são investigar o pré-processador da linguagem Wiki e especificar uma linguagem com símbolos que facilitassem a escrita de formatação, listas de tópicos numerados e não-numerados e dicionários e que não interferisse na passagem para HTML.

### 2.3 Expressões regulares

As expressões regulares usadas foram:

- i)  $[bui] < |h[1 - 6] <$
- ii)  $[ou]l\backslash[$
- iii)  $dl\backslash\{$
- iv)  $.\backslash n$
- v)  $"\backslash "$
- vi)  $strong < |em < |mark < |small <$
- vii)  $del < |ins < |sub <$
- viii)  $sup < |strike < |pre <$
- ix)  $code < |tt <$
- x)  $blockquote < |center <$
- xi)  $">"$

xii) "]"

xiii) "|"

xiv) "}"

xv) "\$"

xvi) ":"

## Capítulo 3

# Codificação e Testes

### 3.1 Problemas de implementação, Decisões e Alternativas

#### 3.1.1 Problemas de implementação

De um modo geral, não tivemos muitos problemas de implementação. Simplesmente tivemos que alterar o símbolo para abrir e fechar as listas e os dicionários, pois da forma que implementamos o array, se elas tivessem todas o mesmo símbolo não sabíamos qual das "tags" estávamos a fechar.

#### 3.1.2 Decisões

Como já foi dito no capítulo 2, neste trabalho tivemos de escolher algumas marcas menos pesadas. Para abreviar a escrita de formatação usamos:

- 1- Negrito: `\b< texto >`
- 2- Itálico: `\i< texto >`
- 3- Sublinhado: `\u< texto >`
- 4- Níveis de títulos: `\h[1 - 6]< texto >`
- 5- Listas não numeradas: `\ul[item|item|...]`
- 6- Listas numeradas: `\ol[item|item|...]`
- 7- Dicionário: `\dl {`  
    palavra: definição \$  
    palavra: definição \$  
    .....  
}

Acrescentamos outros símbolos, mas têm a mesma formatação que o 1, 2, 3 e 4.

Caso alguém queira utilizar algum dos símbolos, mas não usar a sua formatação, basta acrescentar um `\` antes dos símbolos. Por exemplo escrever a expressão matemática " $3 > 1$ " em negrito, basta fazer "`\b< 3\> 1 >`"

#### 3.1.3 Alternativas

Uma alternativa seria construir uma gramática e usar a ferramenta *yacc* em conjunto com o *Flex*. Outra alternativa era modificar a implementação do pré-processador para identificar outros símbolos.

## 3.2 Testes realizados e Resultados

Criamos um ficheiro txt com as nossas anotações mais simples e mais leves. (Figura A.1). Depois através do ficheiro makefile, compilamos o nosso ficheiro em *Flex* e passamos ao executável o ficheiro txt (Figura A.2 e A.3). Como resultado obtemos um ficheiro já com as anotações do HTML. (Figura A.4 e A.5)



## Capítulo 4

# Conclusão

Hoje em dia já existem muitos pré-processadores que facilitam a escrita de documentos em HTML. Tendo em conta os aspetos apresentados ao longo do relatório, conclui-se que o *Flex* é uma boa ferramenta para fazer o pré-processador e que com ele se torna fácil programar usando expressões regulares e um pouco de linguagem C. Como trabalho futuro poderíamos acrescentar mais símbolos para completar o pré processador.

# Apêndice A

## Figuras

```
\h1< Titulo >
\n
\h2< Sub-Titulo >
\n
\ul[
    Quando se faz \b<negrito> o simbolo é b.|
    Quando se faz \i<itálico> o simbolo é i.|
    Quando se faz \u<sublinhado> o simbolo é u.
]
\n
\ol[
    Quando se faz \b<negrito> o simbolo b.|
    Quando se faz \i<itálico> o simbolo i.|
    Quando se faz \u<sublinhado> o simbolo \u.
]
\n
\dl{
    \b<negrito> : b\:bond $
    \i<itálico> : i $
    \u<sublinhado> : u
}

Uma expressão matemática a negrito:
\b< 3\> 1>

\n
\h3< The End Folks >
```

Figura A.1: Exemplo com as "tags" simplificadas

```
EXEC = ex
$(EXEC): $(EXEC).l
    flex -o $(EXEC).c $(EXEC).l
    gcc -o $(EXEC) $(EXEC).c
teste: $(EXEC)
    ./$(EXEC) < exemplo1.txt
```

Figura A.2: Ficheiro makefile

```
user@user-X556UF:~/Desktop/PLC/trabalhoP2$ make
flex -o ex.c ex.l
gcc -o ex ex.c
user@user-X556UF:~/Desktop/PLC/trabalhoP2$ make teste
./ex < exemplo1.txt
```

Figura A.3: Terminal

```

<h1> Título </h1>

<h2> Sub-Título </h2>

<ul><li>
  Quando se faz <b>negrito</b> o simbolo é b.</li>
<li>
  Quando se faz <i>itálico</i> o simbolo é i.</li>
<li>
  Quando se faz <u>sublinhado</u> o simbolo é u.
</li></ul>

<ol><li>
  Quando se faz <b>negrito</b> o simbolo b.</li>
<li>
  Quando se faz <i>itálico</i> o simbolo i.</li>
<li>
  Quando se faz <u>sublinhado</u> o simbolo | u.
</li></ol>

<dl><dt>
  <b>negrito</b> </dt>
<dd> b:bond </dd>
<dt>
  <i>itálico</i> </dt>
<dd> i </dd>
<dt>
  <u>sublinhado</u> </dt>
<dd> u
</dd></dl>

Uma expressão matemática a negrito</dt>
<dd>
<b> 3> 1</b>

<h3> The End Folks </h3>

```

Figura A.4: Exemplo 1 depois do pré-processador

# Titulo

## Sub-Titulo

- Quando se faz **negrito** o simbolo é b.
- Quando se faz *itálico* o simbolo é i.
- Quando se faz sublinhado o simbolo é u.

- Quando se faz **negrito** o simbolo b.
- Quando se faz *itálico* o simbolo i.
- Quando se faz sublinhado o simbolo | u.

**negrito**  
b:bond  
*itálico*  
i  
sublinhado  
u

Uma expressão matemática a negrito  
**3> 1**

### The End Folks

Figura A.5: Exemplo HTML

## Apêndice B

# Código do Pré-Processador

```
%{
/* Declarações C diversas */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define Max 128

int n = 0;
char st[Max][32] = {0};
FILE* f;

void insere(char *id ){
    int i;
    for (i=0; id[i]!='\0'; i++){
        st[n][i] = id[i];
    }
    st[n][i-1] = '\0';
}

%}

%option noyywrap

%x TEXT SIMB

%%

                {BEGIN TEXT;}

<TEXT>{
    ">"                {fprintf(f, "</%s>", st[--n] );}
    "]"                {fprintf(f, "</li></%s>", st[--n]);}
    "|"                {fprintf(f, "</li>\n<li>");}
    "}"                {fprintf(f, "</dd></%s>", st[--n]);}
    "$"                {fprintf(f, "</dd>\n<dt>");}
    ":"                {fprintf(f, "</dt>\n<dd>");}
    "\\\"              {BEGIN SIMB; strcpy(st[n], "");}
    ".|\n              {fprintf(f, "%s", yytext );}
}

<SIMB>{
    [bui]<|h[1-6]<                {BEGIN TEXT; insere(yytext); fprintf(f, "<%s>", st[n++] );}
    strong<|em<|mark<|small<    {BEGIN TEXT; insere(yytext); fprintf(f, "<%s>", st[n++] );}
    del<|ins<|sub<                {BEGIN TEXT; insere(yytext); fprintf(f, "<%s>", st[n++] );}
    sup<|strike<|pre<            {BEGIN TEXT; insere(yytext); fprintf(f, "<%s>", st[n++] );}
    code<|tt<                    {BEGIN TEXT; insere(yytext); fprintf(f, "<%s>", st[n++] );}
    blockquote<|center<         {BEGIN TEXT; insere(yytext); fprintf(f, "<%s>", st[n++] );}
    [ou]1\|                    {BEGIN TEXT; insere(yytext); fprintf(f, "<%s><li>", st[n++] );}
    dl\{                        {BEGIN TEXT; insere(yytext); fprintf(f, "<%s><dt>", st[n++] );}
```

```

        .|\n                                {BEGIN TEXT; fprintf(f,"%s",yytext);}
    }

%%

int main(){
    f=fopen("ex.html", "w");
    yylex();
    fclose(f);
    return 0;
}

```