

Processamento de Linguagens

Trabalho Prático N^o2

Pré-processador para HTML



Francisco Lira
A73909



Francisco Costa
A70922

06 de Maio 2018

Conteúdo

1	Introdução	2
2	Descrição do Problema	3
2.1	Anotações de Markdown	3
2.1.1	Títulos	3
2.1.2	Realçar	3
2.1.3	Listas	3
2.1.4	Links	4
2.1.5	Blocos de código	4
3	Decisões	5
3.1	Makefile	5
3.2	Stack	5
3.3	Nível de prioridade	6
4	Implementação	7
4.1	Títulos	8
4.2	Realçar	8
4.3	Listas	9
4.4	Links	9
4.5	Blocos de código	9
5	Exemplos de Utilização	10
6	Apreciação Crítica	12
A	Código Fonte	13

1 Introdução

Este relatório é o resultado do segundo trabalho prático proposto e elaborado no âmbito da unidade curricular de Processamento de Linguagens. Este trabalho tem como principal finalidade desenvolver os conhecimentos da ferramenta *flex*. Resolvemos a proposta número 3, o **Pré-processador para HTML**. O terceiro problema escolhido através do número de aluno mais baixo do nosso grupo.

$$(70922 \% 5) + 1 = 3$$

2 Descrição do Problema

O objetivo deste enunciado é converter um ficheiro do tipo *Wiki* ou *Markdown* para *HTML* para que consigamos de uma maneira mais simples ter anotações num texto que irá ser posteriormente convertido e colocadas as tags no respetivo local. Visto que já existe o formato *Markdown* num formato mais ou menos padrão, optamos por utilizar essas mesmas *tags* para que fosse possível usar, no futuro, este pré-processador.

2.1 Anotações de Markdown

Definimos as anotações que vamos usar com base nas anotações definidas em *Markdown* que se usam em páginas como o GitHub. Deste modo iremos ficar com uma ferramenta bastante útil para o futuro.

2.1.1 Títulos

Foram definidos seis níveis de título possíveis. Para os anotar deve-se usar desde 1 caracter cardinal, até 6 caracteres conforme a importância vai diminuindo.

```
# Header1  
## Header2  
### Header3  
#### Header4  
##### Header5  
##### Header6
```

2.1.2 Realçar

Existem 3 tipos diferentes de realce que podem ser usados, itálico, negrito e riscado, sendo que para o itálico ou negrito foram definidas duas anotações diferentes para tornar compatível com mais tipos de *Markdown*. Deste modo, se for usado ******ou _tem-se texto em itálico, ******ou __tem-se texto a negrito e, por fim, texto riscado.

```
*itálico* ou _itálico_  
**negrito** ou __negrito__  
~~riscado~~
```

2.1.3 Listas

Para a definição das listas optamos por definir listas ordenadas e listas não ordenadas. Nas listas não ordenadas os elementos que tenham "0.", isto é, zero seguido de um ponto, ficam marcadas como sendo elementos de uma lista não ordenada. Caso tenham qualquer outro inteiro, como por exemplo "3.", são marcados como sendo integrantes de uma lista ordenada.

0. Elemento de uma lista não ordenada
0. Outro elemento para a mesma lista
1. Aqui começa uma lista ordenada
2. Mais um elemento para a lista
4. Aqui podemos ver que não interessa que inteiro é, desde que não seja 0, é o próximo elemento da lista

2.1.4 Links

Para serem definidos links para websites também definimos a anotação com base na anotação de Markdown. Assim, para serem criados links, temos de colocar o texto que queremos tornar numa hiperligação dentro de parênteses retos e o respetivo link dentro de parenteses curvos logo de seguida.

[Texto tornado em hiperligação] (www.natura.di.uminho.pt)

2.1.5 Blocos de código

De modo a ser apresentado código, foi usado, mais uma vez, anotação com base na anotação de Markdown. Com vista a serem criados blocos de código, bastará, apenas que sejam colocados três acentos graves “““no inicio e no fim do bloco.

```
```
int addOne(int n)
{
 return n+1;
}
```
```

3 Decisões

3.1 Makefile

Para facilitar o desenvolvimento desta ferramenta, começamos por elaborar uma makefile simples. Esta, além de realizar a compilação do programa, gera o ficheiro executável. Temos também testes que nos auxiliaram no momento de testar funcionalidades para podermos facilmente ver o output gerado. Por último, temos também uma regra no momento de remover os executáveis criados e ficheiros gerados por nós.

```
t1: do
    ./exec <test/in1.md >test/out1.html

do:
    flex converter.l
    gcc lex.yy.c -o exec

clean:
    rm lex.yy.c exec
    rm test/out*.html
```

3.2 Stack

Como o objetivo é a criação de um ficheiro *HTML*, e tendo em conta que em *HTML* as tags são abertas e fechadas como uma fila LIFO, criamos uma estrutura de stack que auxilia no momento de decisão de abertura e fecho das tags.

Para isto foi criado um array com algumas posições e um inteiro que indica em que posição está a ser feita a leitura e a escrita. Foram também desenvolvidas as funções necessárias para a manipulação da mesma.

Desta forma, é possível abrir tags antes de fechar a anterior, algo que é necessário quando fazemos por exemplo:

```
**negrito e _itálico**.
```

Neste exemplo podemos ver que toda a frase está a negrito e que a palavra itálico está não só a negrito mas também a itálico, uma vez que apenas no final da frase é que a anotação é terminada.

3.3 Nível de prioridade

Foi implementado também um outro elemento que faz validação do nível de prioridades. Este serve para que no momento em que abrimos ou fechamos alguma coisa, não tenhamos tags incompatíveis por fechar. Isto é, se tentarmos abrir um *header* mas não tivermos fechado um *ítalico*, automaticamente esse itálico será fechado porque não é possível que o mesmo se estenda mais.

Para tal, definimos então um "nível" para cada uma das tags html, onde caso estejamos a abrir um elemento de valor superior ao que está no topo da stack, o que lá está será fechado, até que o elemento que estamos a adicionar seja de valor igual ou inferior ao que se encontra na stack.

Tabela dos níveis definidos:

Tags	Nível
h1,h2,h3,h4,h5,h6	1
code	2
ol,ul	3
li	4
i,b,s	5

Tabela 1: Níveis estabelecidos

4 Implementação

Para a implementação, desenvolvemos uma função que trata de abrir ou fechar tags, conforme for necessário, o seu código apresenta-se a seguir:

```
void starttag(char* tag){  
if(q>0){  
if(strcmp(tag,stackGet())==0){  
printf("</%s>",tag);  
stackRemove();  
}  
else{  
int lvl = getLevel(tag);  
while(q>0 && ((getLevel(stackGet())<lvl)){  
printf("<%s>",stackGet());  
stackRemove();  
}  
if(getLevel(stackGet())==1){  
puttab(q+1);  
}  
printf("<%s>",tag);  
stackAdd(tag);  
}  
}  
else{  
if(getLevel(tag)<2) puttab(q+1);  
printf("<%s>",tag);  
stackAdd(tag);  
}  
}
```

Esta função verifica se há alguma coisa na stack, se não houver a anotação em questão ainda não foi iniciada então é a única coisa que faz, iniciar a anotação. Caso haja alguma coisa na stack, vamos verificar se o que está no topo da stack é a mesma tag que estamos a inserir, caso se verifique que sim, fechamos essa tag, pois não temos nenhuma que seja possível abrir duas vezes. Caso não seja a mesma é verificado qual o nível da anotação que lá está, se for de nível superior ao que está a ser inserida então é fechada essa anotação que está no topo da stack, se não for então abrimos essa anotação que está a ser inserida.

Para além desta função, há outra bastante importante, a função que faz a verificação se devem ser fechadas as tags que estão abertas quando há uma nova linha.

```

oid checkheader(){
if(q>0){
if(((strcmp(stackGet(),"h6")) == 0)
||((strcmp(stackGet(),"h5")) == 0)
||((strcmp(stackGet(),"h4")) == 0)
||((strcmp(stackGet(),"h3")) == 0)
||((strcmp(stackGet(),"h2")) == 0)
||((strcmp(stackGet(),"h1")) == 0)
||((strcmp(stackGet(),"li")) == 0)){
printf("</%s>\n",stackGet());
stackRemove();
}
else if(((strcmp(stackGet(),"ol")) == 0)
||((strcmp(stackGet(),"ul")) == 0)){
puttab(q-1);
printf("</%s>\n",stackGet());
stackRemove();
}
}
else printf("\n");
}
\\%
\\n  checkheader();
\\%

```

Se for inserida uma nova linha, é verificada a existencia de anotações na stack, se as mesmas existirem, são terminadas e removidas da stack.

4.1 Títulos

As expressões regulares usadas para os títulos são estas e é usada a função que trata de lidar com a inserção e remoção de anotações:

```

#{6} starttag("h6");
#{5} starttag("h5");
#{4} starttag("h4");
### starttag("h3");
## starttag("h2");
# starttag("h1");

```

4.2 Realçar

Tal como nos títulos, usamos a função *starttag* para gerir a inserção de anotações e usamos as seguintes expressões regulares:

```

/* starttag("b");
_starttag("i");
/* starttag("i");

```

4.3 Listas

A função usada pelas listas, é como se pode verificar diferente, e são usadas as seguintes expressões regulares:

```
\n[1-9]+.\ startlist("ol");
\n0.\ startlist("ul");
```

Esta função *startlist* gera as anotações apenas quando se trata de listas, uma vez que aqui são usadas anotações de lista ordenada ("ol"), não ordenada("ul") e também ("li"), e esta última nunca aparece nas inserções, pois é completamente gerida por esta função.

4.4 Links

Esta é a expressão regular usada mais complexa:

```
\[ BEGIN LINK;
<LINK>{
[^\\n]+\\] {tmpLink[0]=strdup(yytext);tmpLink[0][yylen-1] = 0;}
\\((https:\\/\\/)?(www.)?
[A-Za-z]+\\.com\\) {tmpLink[1]=strdup(yytext);tmpLink[1][yylen-1] = 0;createLink(); BEGIN }
```

Com esta expressão regular limitamos ao utilizador o uso dos parenteses retos apenas e só para a indicação de links. Ao encontrar um parenteses reto esquerdo, a verificação LINK é iniciada, e depois guardamos o interior da anotação «a_l» numa variável e em seguida fazemos o mesmo com a link, por fim imprimimos tudo co o auxilio da função *createLink()*

4.5 Blocos de código

Para blocos de código usamos de novo a função *starttag* pois a anotação code é no fundo como as anotações de realçes, apesar de ter maior prioridade na função de nível.

```
``` starttag("code");
```

## 5 Exemplos de Utilização

Explicitamos agora alguns exemplos do antes e depois da passagem de um arquivo pelo analisador sintático que foi desenvolvido.

#H1	<h1>H1</h1>
##H2	<h2>H2</h2>
###H3	<h3>H3</h3>
####H4	<h4>H4</h4>
#####H5	<h5>H5</h5>
#####H6	<h6>H6</h6>
Títulos	
*Asteriscos* ou _underscores_.	<i>Asteriscos*</i> ou <i>underscores</i>.
**Asteriscos** ou __underscores__.	<b>Asteriscos</b> ou <b>underscores</b>.
**Negrito e _Itál_**.	<b>Negrito e <i>Itál</i></b>.
E texto ~~riscado~~	E texto <s>riscado</s>
Realções	
1. Lista Ordenada	<ol>
2. Lista	<li>Lista Ordenada</li>
3. Item	<li>Lista</li>
0. Lista não ordenada	<li>Item</li>
0. Outro elemento	</ol>
1. Lista	<ul>
4. Ordenada	<li>Lista não ordenada</li>
	<li>Outro elemento</li>
	</ul>
	<ol>
	<li>Lista</li>
	<li>Ordenada</li>
	</ol>
Listas	
[I'm an inline-style link](https://www.google.com)	<a href="https://www.google.com">I'm an inline-style link</a>
Hiperligação	

```
    ````  
int addOne(int n)  
{  
    return n+1;  
}  
    ````
```

Blocos de código

```
int addOne(int n)
{
 return n+1;
}
```

## 6 Apreciação Crítica

Durante a realização deste trabalho prático, o segundo desta unidade curricular, várias foram as etapas que tivemos de passar para chegar ao resultado final.

Como era de esperar, com a utilização de *Flex*, aprofundamos os conhecimentos sobre a ferramenta e achamos que no futuro somos capazes de com muito mais facilidade a voltar a utilizar.

Estamos satisfeitos com o código desenvolvido e, como trabalho futuro, iremos continuar a acrescentar funcionalidades a esta ferramenta uma vez que a mesma é útil no dia a dia do nosso desenvolvimento.

Concluímos, no geral que estamos satisfeitos com o trabalho desenvolvido, tendo este se revelado bastante útil no aperfeiçoamento dos nossos conhecimentos.

## A Código Fonte

```
%x LINK CHECKER

%{

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int q;
int st;
char** stack;
char** tmplink;

void puttab(int t){
 int i=0;
 while(i<t){
 printf("\t");
 i++;
 }
}

int getLevel(char* tag){
 if (strcmp(tag,"h1")==0) return 1;
 if (strcmp(tag,"h2")==0) return 1;
 if (strcmp(tag,"h3")==0) return 1;
 if (strcmp(tag,"h4")==0) return 1;
 if (strcmp(tag,"h5")==0) return 1;
 if (strcmp(tag,"h6")==0) return 1;
 if (strcmp(tag,"code")==0) return 2;
 if (strcmp(tag,"ol")==0) return 3;
 if (strcmp(tag,"ul")==0) return 3;
 if (strcmp(tag,"li")==0) return 4;
 if (strcmp(tag,"i")==0) return 5;
 if (strcmp(tag,"b")==0) return 5;
 if (strcmp(tag,"s")==0) return 5;
 return 0;
}

void stackAdd(char* tag){
 int tmp = strlen(tag);
 stack[q] = (char*)malloc(sizeof(char)*(tmp+1));
 strcpy(stack[q],tag);
 q++;
 if(q>st-1){
 stackDouble();
 }
}
```

```

void stackRemove(){
q--;
free(stack[q]);
}

char* stackGet(){
return stack[q-1];
}

char* stackGetList(){
return stack[q-2];
}

void starttag(char* tag){
if(q>0){
if(strcmp(tag,stackGet())==0){
printf("</%s>",tag);
stackRemove();
}
else{
int lvl = getLevel(tag);
while(q>0 && ((getLevel(stackGet()))<lvl)){
printf("</%s>",stackGet());
stackRemove();
}
if(getLevel(stackGet())==1){
puttab(q+1);
}
printf("<%s>",tag);
stackAdd(tag);
}
}
else{
if(getLevel(tag)<2) puttab(q+1);
printf("<%s>",tag);
stackAdd(tag);
}
}
}

void startlist(char* tag){
if(q>0){
if(strcmp(stackGet(),"li") == 0){
if(strcmp(stackGetList(),tag) == 0){
printf("\n");
puttab(q-1);
printf("");
}
else{
printf("\n");
stackRemove();
}
}
}
}

```

```

puttab(q-1);
printf("</%s>\n",stackGet());
stackRemove();
puttab(q);
printf("<%s>\n",tag);
stackAdd(tag);
puttab(q);
printf("");
stackAdd("li");
}
}

}

else{
puttab(q);
printf("<%s>\n",tag);
stackAdd(tag);
puttab(q);
printf("");
stackAdd("li");
}
}

void checkheader(){
if(q>0){
if(((strcmp(stackGet(),"h6")) == 0)
||((strcmp(stackGet(),"h5")) == 0)
||((strcmp(stackGet(),"h4")) == 0)
||((strcmp(stackGet(),"h3")) == 0)
||((strcmp(stackGet(),"h2")) == 0)
||((strcmp(stackGet(),"h1")) == 0)
||((strcmp(stackGet(),"li")) == 0)){
printf("</%s>\n",stackGet());
stackRemove();
}
else if(((strcmp(stackGet(),"ol")) == 0)
||((strcmp(stackGet(),"ul")) == 0)){
puttab(q-1);
printf("</%s>\n",stackGet());
stackRemove();
}
}
else printf("\n");
}

void closetags(){
while(q>0){
printf("</%s>\n",stackGet());
stackRemove();
}
}

```

```

}

void createLink(){
printf("<a href=\"https://www.%s\"%s",tmplink[1],tmplink[0]);
}

%%

\n checkheader();
\[BEGIN LINK;
#{6} starttag("h6");
#{5} starttag("h5");
#{4} starttag("h4");
starttag("h3");
starttag("h2");
starttag("h1");
`` starttag("code");
~~ starttag("s");
__ starttag("b");
/ starttag("b");
_ starttag("i");
* starttag("i");
\n[1-9]+\` startlist("ol");
\n0.\` startlist("ul");
. printf("%s",yytext);

<LINK>{
[^`n]+`\` {tmplink[0]=strdup(yytext);tmplink[0][yyleng-1] = 0;}
\((https:\`/\`)?(www.)?
[A-Za-z]+\`.\` {tmplink[1]=strdup(yytext);tmplink[1][yyleng-1] = 0;createLink(); BEGIN I
}

%%

int yywrap(){
return 1;
}

int main(){
int s;
q = 0;
st = 15;
stack = (char**)malloc(sizeof(char*)*st);
tmplink = (char**)malloc(sizeof(char*)*2);
/*
costum css
padding: 16px;

```

```
background-color: #f6f8fa;
font-family: "SFMono-Regular",Consolas,"Liberation Mono",Menlo,Courier,monospace;
*/
printf("<!DOCTYPE html>\n<html>\n<head>\n<style>\n\tcode{\padding: 16px;}\n</style>\n</head>\n<body>\n</body>\n</html>");

return 0;
}
```