PRÉ-PROCESSADOR DE HTML

O enunciado que nos foi atribuído segundo o sistema definido foi o enunciado 3 **"**Pré-processador de HTML**"**. Este consiste da criação de uma linguagem de anotação do género do sistema Wiki, para a construção de páginas de HTML. A linguagem criada deve permitir a abreviação da escrita de formatação de texto e de lista de tópicos de forma a minimizar o tempo gasto e tornar o processo mais eficiente.

Para além disso, foi necessária a construção de um analisador léxico, usando a ferramenta Flex, que permitisse a expansão das abreviaturas encontradas.

Assim, consideramos que a melhor opção para a resolução deste problema era o uso de Start Conditions, mais concretamente, inclusivas já que é possível haver, por exemplo, texto em bold e itálico ao mesmo tempo, isto é, podem ser detetados padrões exteriores a uma start condition enquanto esta se encontra ativa. Estas foram, então inicializadas com a opção %s para sinalizar esta sua propriedade.

```
%s BOLD
%s ITALIC
%s UNDERLINED
%s HEADING1
%s HEADING2
%s HEADING3
%s HEADING4
%s HEADING5
%s HEADING6
%s UNORDLIST
%s LISTELEM
%s ORDLIST
%s DESCLIST
%s DESCTERM
%s DEFTERM
```

Ainda tendo em vista o facto e a necessidade de serem inclusivas, uma vez que cada tag de abertura tem que ser fechada com a respetiva tag de fecho, decidimos usar a stack do Flex para este processo.

```
%option stack
```

O flex, providencia também funções de manuseio da stack, nomeadamente yy_push_state e yy_pop_state. Estas foram utilizadas da seguinte forma: quando um padrão era detetado, em vez de fazer BEGIN da start condition necessária, é feito push da mesma para a stack. Quando

aparece um padrão que leva ao fecho da mesma, em vez de BEGIN 0, é feito pop da stack. Isto garante que no caso de haverem tags a abrirem e a fecharem dentro de outras, a primeira a abrir é a última a fechar, mantendo a ordem correta de fecho e abertura.

```
yy_push_state(start_condition);
yy_pop_state();
```

Um exemplo concreto do uso desta técnica encontra-se na imagem a seguir, com a formatação de texto bold.

```
    /* negrito */
"#b"     {fprintf(output, "<b>"); yy_push_state(BOLD);}
<BOLD>"##"   {fprintf(output, "</b>"); yy_pop_state();}
```

Por outro lado, no que concerne à linguagem criada, definimos todos os padrões a começar com um cardinal (por exemplo #b para bold, #i para itálico, etc.). Para finalizar a secção de texto com uma determinada propriedade (pseudo tag de fecho), são utilizados dois cardinais.

Atentado no exemplo, seguinte. O seguinte texto com a nossa anotação

```
#b texto em bold ##

#u texto sublinhado ##

#u #b texto a bold e sublinhado ## ##
```

após a transformação com o analisador léxico ficaria em html como mostra na imagem abaixo

```
<b> texto em bold </b>

<u> texto sublinhado </u>

<u> <b> texto a bold e sublinhado </b> </u>
```

e utilizando o firefox para pré-visualizar o resultado final:

texto em bold
texto sublinhado
texto a bold e sublinhado

Esta foi a estrutura utilizada para os casos que resolvemos definir. Nomeadamente, a nível de formatação, negrito, itálico e sublinhado. A nível de títulos, criamos regras para headings desde nível 1 ao nível 6.

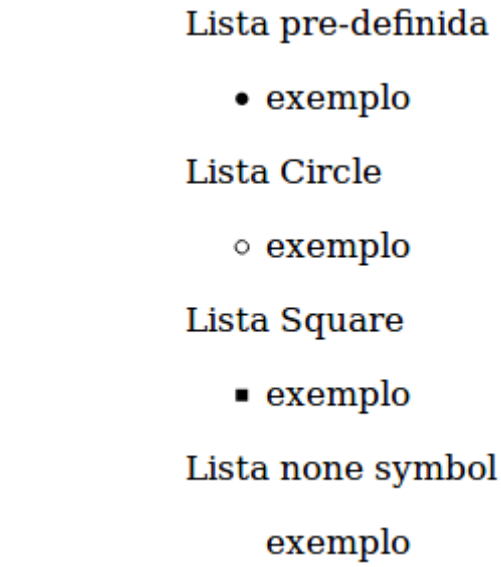
Por fim, no que toca a listas, definimos para unordered, ordered e description lists. Começando pelas unordered (isto é, não numeradas) criamos regras também para os casos em que a o simbolo que antecede cada ítem seja personalizado (círculo, quadrado, ou nenhum) bem como para o caso pré-definido.

```
Lista pre-definida
#ul
#li exemplo ##
##
Lista Circle
#ulc
#li exemplo ##
##
Lista Square
#uls
#li exemplo ##
##
Lista none symbol
#uln
#li exemplo ##
##
```

exemplo de texto com a nossa linguagem

```
Lista pré-definida
<ul>
<li> exemplo </li>
</ul>
Lista Circle
<ul style="list-style-type:circle">
<li> exemplo </li>
</ul>
Lista Square
<ul style="list-style-type:square">
<li> exemplo </li>
</ul>
Lista none symbol
<ul style="list-style-type:none">
<li> exemplo </li>
</ul>
```

exemplo após o analisador léxico



exemplo pré-visualizado no firefox

No que toca às ordered lists, a estratégia usada foi a mesma. As regras extra criadas foram para a numeração dos ítens com letras ou numeração romana, em maiúsculas e em minúsculas. Isto para além do caso pré-definido, a numeração com números árabes usuais.

```
Lista pre-definida
#ol
#li exemplo ##
##
Lista letras maiusculas
#olul
#li exemplo ##
##
Lista letras minúsculas
#olll
#li exemplo ##
##
Lista numeração romana maiúsculas
#olur
#li exemplo ##
##
Lista numeração romana minúsculas
#ollr
#li exemplo ##
##
```

texto

```
Lista pre-definida
<ol>
<li> exemplo </li>
</ol>
Lista letras maiusculas
<ol type="A">
<li> exemplo </li>
</ol>
Lista letras minúsculas
<ol type="a">
<li> exemplo </li>
</ol>
Lista numeração romana maiúsculas
<ol type="I">
<li> exemplo </li>
</ol>
Lista numeração romana minúsculas
<ol type="i">
<li> exemplo </li>
</ol>
```

no terminal

Lista pre-definida

1. exemplo

Lista letras maiusculas

A. exemplo

Lista letras minusculas

a. exemplo

Lista numeracao romana maiusculas

I. exemplo

Lista numeracao romana minusculas

i. exemplo

Por fim, criamos regras para as descriptions lists, não esquecendo as marcas de definição e descrição de termos.

```
    /* The <dl> tag defines the description list */
"#dl"    {fprintf(output, "<dl>"); yy_push_state(DESCLIST); }
<DESCLIST>"##"   {fprintf(output, "</dl>"); yy_pop_state();}

    /* the <dt> tag defines the term (name) */
"#dt"    {fprintf(output, "<dt>"); yy_push_state(DESCTERM); }
<DESCTERM>"##"   {fprintf(output, "</dt>"); yy_pop_state();}

    /* the <dd> tag describes each term */
"#dd"    {fprintf(output, "<dd>"); yy_push_state(DEFTERM); }
<DEFTERM>"##"    {fprintf(output, "</dd>"); yy_pop_state();}
```

Para testar o nosso analisador léxico, criamos ficheiros com a linguagem de anotação e corremos o programa de modo a verificar a correção dos resultados.

Com vista a elevar a qualidade do nosso trabalho, decidimos a criação de um menu (definido na função main) que permita ao utilizador escolher entre visualizar o resultado no terminal, ou criar um ficheiro output.html que poderá eventualmente abrir com o browser. Para tal, inicializamos o FILE *output e recorrendo às funções fopen e getchar construímos o menu.

```
int main(int argc, char* argv[]){

    if (argc==1){
            printf("ERRO. Não introduziu ficheiro de input.\n");
            return 0;
        }

    printf("\n  Escolha umas das seguintes opções:\n");
    printf("      1  -> Imprimir output no terminal\n");
    printf("      2  -> Criar ficheiro com output\n");
    printf("      3  -> Sair\n");

    c = getchar(); printf("\n");

    yyin = fopen(argv[1], "r");

    if (c == '1')
        output = stdout;

    if (c == '2')
        output = fopen("output.html","w");

    if (c == '3')
        return 0;

    yylex();

    fclose(output);

    return 0;
}
```

main

Na imagem, podemos ver que o FILE output, é o stdout caso a opção “Imprimir output no terminal” seja escolhida, ou então é criado o fhicheiro output.html no caso contrário. Esta funcionalidade só foi possível concretizar recorrendo ao fprint em vez do tradicional printf, e ainda devido à criação de duas regras para o texto não ficar desformatado no ficheiro (quando encontra

algo que não os padrões, imprime, quer sejam caracteres quer sejam tabs, espaços, etc).

```
[ \n\r\t]    fprintf(output, "%s",yytext);
.            fprintf(output, "%s",yytext);
```