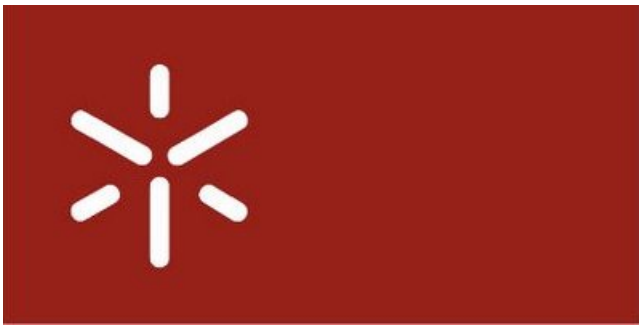

Processador de Inglês Corrente
Processador de Named Entities

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens
2º Semestre\2016-2017

A70676 Marcos de Morais Luís
A71625 Nelson Arieira Parente

18 de Abril de 2017
Braga

Processamento de Linguagens

Conteúdo

Capítulo 1

Introdução

Neste segundo trabalho existem como objetivos previamente definidos novamente um foco no aumento da experiência em ambiente Linux e algumas ferramentas de apoio à programação, a pratica de desenvolvimento de expressões regulares , e o desenvolvimento de filtros que transformem textos com base nos conceito de regras de produção, Condição-Acção. Serão nesta fase utilizados filtros de texto com o gerador FLEX. Foram disponibilizados seis problemas em concreto dando aos alunos a opção de escolher um dos quatro para realizar, o grupo como especificado na capa do relatório decidiu em elaborar uma resolução para dois problemas em concreto, sendo eles o processador de inglês corrente e o processador de named entities.

Capítulo 2

Processador de Inglês Corrente

Este problema em concreto consiste em encontrar certas expressões no texto fornecido e expandi-las mais concretamente contrações que existem na língua inglesa.

2.1 Análise do texto-fonte

O texto fornecido para a realização desta tarefa tem na sua estrutura alguns padrões que após analisados conclui-se que iriam beneficiar e influenciar a estratégia utilizada. De seguida encontra-se um excerto do texto-fonte como ilustração e suporte para a estratégia adotada.

```
[ He'd ] forgotten all about the people in cloaks until he passed a group of
them next to the baker's. He eyed them angrily as he passed. He [ didn't ]
know why, but they made him uneasy. This bunch were whispering
excitedly, too, and he [ couldn't ] see a single collecting tin. It was on
his way back past them, clutching a large doughnut in a bag, that he
caught a few words of what they were saying.
```

Neste enxerto de input conseguimos identificar todas as contrações que podem ser expandidas, mas não encontramos nenhuma correlação entre elas, logo a estratégia passará por identificar cada caso especifico.

2.2 Ações Semânticas

As ações semânticas que se acharam necessárias após a análise do texto fonte são simplesmente duas a eliminação do apóstrofe e a substituição da abreviação da palavra que se segue pela palavra completa.

2.3 Estruturas de Dados Globais

Para o desenvolvimento deste problema decidiu-se recorrer a estrutura AVL por duas principais razões primeiro vai assim deste modo existir uma ordenação natural , a segunda razão é que automaticamente já faz uma filtração de elementos repetidos, não tendo que haver posteriormente uma preocupação pela parte do grupo de eliminar essas redundâncias. Para tal foi utilizado o modulo de AVLs da GNU já creditado em cadeiras anteriores. Com a devida utilização desse modulo foi implementado um modulo de nome verbos.c em que implementamos a nossa própria versão de AVL com recurso ao modulo da GNU. Apresentamos de seguida a API relativa à estrutura implementada para o armazenamento de dados retirados na filtragem do ficheiro fonte.

- API:

```
//Arvore Binaria Balanceada usada para guardar os verbos encontrados
struct myAvl{
AVL entradas ;
};

//Função de comparação utilizida na inserção de elementos na myAVL
int compara(const void * , const void * , void *)

//Construtor Vazio da Estrutura
myAVL new_myAvl()

//Retorna o número de elementos da estrutura myAVL
int contagem(myAVL)

//Insere um novo elemento na estrutura myAVL
myAVL insere(myAVL , char*)

//Retorna uma lista com todos os elementos presentes na estrtura myAVL
char** retorna_elementos(myAVL)

//Função para segundo elemento de uma string
char* str_exp(char*)

//Função de tratamento de String para o terceiro elemento da mesma
char* str_other(char*)

//Cria um ficheiro HTML com os elementos recolhidos na estrutura MyAVL
void cria_html(myAVL)
```

Através da API fornecida podemos concluir que todas as funções básicas de avl foram implementadas tais como o inserir um novo elemento, a contagem de elementos , o retornar uma lista de elementos. Foi também codificada um método que nos cria a partir de uma AVL, o HTML tal como é solicitado no enunciado do trabalho prático.

2.4 Filtro de Texto - Gerador FLEX

Depois da análise efetuada através da analise de padrões e estudo de tratamento da informação e o seu armazenamento procede-se então ao desenvolvimento de um filtro de texto utilizando o gerador FLEX.

2.4.1 Expansão de Expressões

Na primeira das duas questões relativas ao processador de inglês corrente é nos solicitado que a partir de um ficheiro de texto fonte, seja retornado um ficheiro com o mesmo texto mas com todas as contrações expandidas.

- **Função auxiliar** : Função desenvolvida de nome , str exp , que expande uma expressão que se encontra na linha passada como argumento, a expansão que é feita depende do inteiro que se encontra no segundo argumento da função.
- **Definições**: Série de definições codificadas para cada uma das possibilidades de contrações que existem na gramática inglesa, especificando assim cada caso existente.
- **Regras** : Em termos das regras a estratégia utilizada foi nesta fase ligar cada contração encontrada com o inteiro que irá ser passado a a função auxiliar havendo assim uma correlação entre o numero de regras e o numero de contrações identificas.
- **Main**: Por final a main é bastante simples caso haja dois argumentos sendo o primeiro o executável e o segundo o ficheiro a ser scanizado procede-se à filtragem.

Implementação:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* str_exp(char* linha, int op){
    ...

}

%}

%option noyywrap
NOUN [Ii]|[Yy]ou|[Hh]e|[Ss]he|[Ii]t|[Tt]hey|[Ww]e
AM  {NOUN}'m
ARE     {NOUN}'re
IS {NOUN}'s
WILL {NOUN}'ll
WOULD {NOUN}'ld|{NOUN}'d
DONT    [Dd]on't
CANNOT [Cc]an't
WASNOT [Ww]asn't
DIDNOT [Dd]idn't
THATS [Tt]hat's
HADNT [Hh]adn't
COULD [Cc]ouldn't
HAVE  {NOUN}'ve

%%
{AM}  {printf("{%s}" , str_exp(yytext,1));}
{ARE}  {printf("{%s}" , str_exp(yytext,2));}
{IS}  {printf("{%s}" , str_exp(yytext,3));}
{WILL}  {printf("{%s}" , str_exp(yytext,4));}
{WOULD}  {printf("{%s}" , str_exp(yytext,5));}
{CANNOT}  {printf("{%s}" , str_exp(yytext,6));}
{WASNOT}  {printf("{%s}" , str_exp(yytext,7));}
{DIDNOT}  {printf("{%s}" , str_exp(yytext,8));}
{THATS}  {printf("{%s}" , str_exp(yytext,9));}
{HADNT}  {printf("{%s}" , str_exp(yytext,10));}
{COULD}  {printf("{%s}" , str_exp(yytext,11));}
```

```
{DONT}  {printf("{%s}" , str_exp(yytext,12));}
{HAVE} {printf("{%s}" , str_exp(yytext,13));}
.|\n { ECHO ; }

%%

int main(int argc, char* argv[]){
 if ( argc == 2 )
 yyin = fopen(argv[1], "r");
 yylex();
 return 0;
}
```

Output:

```
{He would} forgotten all about the people in cloaks until he passed a group of
them next to the baker's. He eyed them angrily as he passed. He {did not}
know why, but they made him uneasy. This bunch were whispering
excitedly, too, and he {could not} see a single collecting tin. It was on
his way back past them, clutching a large doughnut in a bag, that he
caught a few words of what they were saying.
```

2.4.2 Verbos

Nesta segunda questão o objetivo é recolher todos os verbos que se encontrem no infinito não flexionado considerando também a forma interrogativa e retornar um HTML com essa lista ordenada.

Implementação:

```
    %{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "verbos.h"

myAVL verbos = NULL ;
%}

%option noyywrap
VERBO [A-Za-z]+
SUJ [Ii]|[Yy]ou|[Hh]e|[Ss]he|[Ii]t|[Ww]e|[Tt]hey
CAN [Cc]an(\ {VERBO})
COULD [Cc]ould(\ {VERBO})
SHALL [Ss]hall(\ {VERBO})
WILL [Ww]ill(\ {VERBO})
WOULD [Ww]ould(\ {VERBO})
MAY [Mm]ay(\ {VERBO})
MIGHT [Mm]ight(\ {VERBO})
OTHER  ([Dd]id|[Dd]o|[Dd]oes)(\ ){SUJ}(\ ){VERBO}

%%
{CAN}|{COULD}|{SHALL}|{WILL}|{WOULD}|{MAY}|{MIGHT} {
 if(!verbos) verbos = new_myAvl() ;
```

```
 char* aux = strdup(str_exp(yytext)); ;
 insere(verbos , aux ) ;
 }
{OTHER}  {
 if(!verbos) verbos = new_myAvl() ;
 char* aux = strdup(str_other(yytext)); ;
 insere(verbos , aux ) ;
 }
.+|\n {}
%%


int main(int argc, char* argv[]){
 if ( argc == 2 )
 yyin = fopen(argv[1], "r");
 yylex();
 cria_html(verbos) ;
 return 0;
 }
```

Output:

Verbos

- to arrive
- to ask
- to be
- to do
- to jump
- to know
- to need
- to swim

Capítulo 3

Processador de Named Entities

O principal objectivo deste probelma é a elaboração de um indíce HTML com as pessoas, países e cidades referidas num ficheiro ENAMEX.

3.1 Análise do texto-fonte

Um ficheiro ENAMEX é um documento anotado num dialeto XML, isto permite através de etiquetas identificar entidades referidas no texto. Tendo isto a pesquisa através das etiquetas de identificação é fundamental para a criação do indíce HTML.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<ENAMEX> Vivia </ENAMEX> este hero i no <ENAMEX TYPE="CITY"> Rio de
Janeiro </ENAMEX>, <ENAMEX TYPE="COUNTRY">Brasil</ENAMEX> e era
professor na˜o se sabe de que doutrinas no ano de
<TIMEX TYPE="DATE"> 1710 </TIMEX>, quando os franceses comandados por <ENAMEX TYPE="PERSON"> Duclerc </ENAMEX
O governador portou-se mal, e permanecia numa deplor avel inac c˜ao, quando <ENAMEX TYPE="PERSON"> Bento do A
e os agressores ficassem prisioneiros.
<ENAMEX> N˜ao </ENAMEX> tardou <ENAMEX TYPE="PERSON"> Dugua-Trouin </ENAMEX> a vir tomar a desforra.
</document>
```

Facilmente identifica-mos as diversas etiquetas presentes no excerto, o passo seguinte passou por definir a sua estrutura em cada uma das entidades necessárias para o nosso índice.

3.2 Ações Semânticas

As acções semânticas necessárias foram relativamente simples, sendo necessário em primeiro lugar eliminar as etiquetas que envolvem e perante o resultado eliminar os espaços antes e depois de modo a limpar a string resultante.

3.3 Estruturas de Dados Globais

Relativamente a estruturas de dados globais recorremos novamente a Àrvores Binárias Balanceadas usando as API's da GNU relativas a este tipo de àrvores. Para além destas foi necessário a implementação de um ficheiro C para ajudar na resolução do problema.

"enamex.h"

```
// Àrvore Binária Balanceada usada para guardar as entidades encontradas
typedef struct myAvl* myAVL ;
```

```
// Função de Comparação das estrutura myAVL
int compara(const void * avl_a , const void * avl_b , void * param) ;

// Construtor Vazio da estrutura myAVL
myAVL new_myAvl() ;

// Retorna o número de Elementos da estrutura myAVL
int contagem(myAVL avl) ;

// Insere elemento na Estrutura myAVL
myAVL insere(myAVL avl , char* entrada ) ;

// Retorna todas as strings presentes numa estrutura do tipo myAVL
char** retorna_elementos(myAVL avl) ;


// Recebendo uma string devolve uma nova sem as tags ENAMEX presentes na recebida
char* elimina_tags(char* linha) ;

// Cria o ficheiro HTML com os elementos recolhidos nas diversas estruturas myAVL
void cria_html(myAVL listaPessoas , myAVL listaPaises , myAVL listaCidades ) ;
```

3.4 Filtros de Texto - Gerador FLEX

Depois de efectuadas as análises necessárias aos padrões presentes no texto-fonte avançamos para o desenvolvimento de um filtro de texto em FLEX que dado um ficheiro ENAMEX cria um indíce HTML com as entidades referidas neste mesmo.

- **Funções auxiliares** : A função desenvolvida de nome , elimina_tags , tem como objectivo eliminar as etiquetas presentes antes de depois de uma entidade referida, esta função juntamente com a função trim devolve a string correspondente à entidade de uma forma limpa, isto deve-se ao facto de apenas um espaço antes ou depois da entidade faz com que seja duplamente inserida na estrutura correspondente.
- **Definições**: Série de definições codificadas para cada uma das possibilidades de etiquetas procuradas pelo ficheiro, especificando assim cada caso existente.
- **Regras** : Sempre que uma dada etiqueta é encontrada a entidade nesta referida é limpa e a string resultante inserida na estrutura de dados relativa a esse tipo de entidade. No final da leitura do ficheiro é criado o índice HTML atráves das diversas entidades recolhidas.
- **Main**: Por final a main é bastante simples caso haja dois argumentos sendo o primeiro o executável e o segundo o ficheiro a ser scanizado procede-se à filtragem.

Implementação:

```
        %{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "avl.h"
#include "enamex.h"

myAVL pessoas = NULL ;
myAVL paises = NULL ;
myAVL cidades = NULL ;
```

```
%}

%option noyywrap
letra [a-zA-ZáÁàÀãÃâÂéÉèÈêÊíÍìÌîÎóÓòÒõÕôÔúÚùÙûÛçÇ]
calendario [0-9]{4}|[0-9]+\/[0-9]+\/[0-9]+
entidade {letra}({letra}|[\.\-]|(\ ))*

pessoa \<ENAMEX(\ )+TYPE=\"PERSON\"(\ )*\>(\ )*{entidade}(\ )*\<\/ENAMEX\>
pais \<ENAMEX(\ )+TYPE=\"COUNTRY\"(\ )*\>(\ )*{entidade}(\ )*\<\/ENAMEX\>
cidade \<ENAMEX(\ )+TYPE=\"CITY\"(\ )*\>(\ )*{entidade}(\ )*\<\/ENAMEX\>
data \<TIMEX(\ )+TYPE=\"DATE\"(\ )*\>(\ )*{calendario}(\ )*\<\/TIMEX\>

%%
{pessoa}  { if(!pessoas) pessoas = new_myAvl() ;
char* aux = strdup(trim(elimina_tags(yytext))) ;
insere(pessoas , aux ) ;
}

{pais}  {
if(!paises) paises = new_myAvl() ;
char* aux = strdup(trim(elimina_tags(yytext))) ;
insere(paises , aux ) ;
}

{cidade}  {
if(!cidades) cidades = new_myAvl() ;
char* aux = strdup(trim(elimina_tags(yytext))) ;
insere(cidades , aux ) ;
}

.|\n {}
%%


int main(int argc, char* argv[]){
if ( argc == 2 )
yyin = fopen(argv[1], "r");
yylex();
cria_html(pessoas , paises , cidades ) ;
return 0;
}
```

Output:

Pessoas

- Bento do Amaral
- Duclerc
- Dugua-Trouin
- Francisco da Maia

Paises

- Brasil
- Mo

Cidades

- Rio de Janeiro

Capítulo 4

Conclusão

O objetivo deste trabalho prático foi aplicar de uma forma global os conhecimentos adquiridos nas aulas práticas ao longo do semestre relativamente ao uso de filtros de texto em FLEX. Nisto o grupo escolheu dois dos exercícios possíveis , sendo estes o Processador de Inglês corrente e Processador de Named Entities.

A das principal dificuldade prendeu-se no uso da biblioteca GLIB, estando esta dificuldade presente o grupo optou por transferir o source code do código relativo a àrvores binárias balanceadas desta mesma biblioteca alterando apenas o seu nome de modo a facilitar a implementação.

Tendo isto a dificuldade seguinte foi a interpertação de algumas particularidades dos exercícios, no final podemos considerar que as dificuldades foram ultrapassadas. Posto isto, podemos verificar que todas as tarefas a realizar nestes exercícios foram concluídas com sucesso e que deixou-nos com um conhecimento mais aprofundado de FLEX , construção de filtros de texto e expressões regulares.

Capítulo 5

Ficheiros

Processador de Inglês Corrente

5.1 verbos.c

```
    #include <stdio.h>
#include <stdlib.h>
#include "verbos.h"
#include "avl.h"
#include <string.h>



struct myAvl{
AVL entradas ;
};

int compara(const void * avl_a , const void * avl_b , void * param) {
char* a = (char*) avl_a ;
char* b = (char*) avl_b ;
return strcmp(a,b) ;
}


myAVL new_myAvl() {
int i ;
myAVL new = (myAVL) malloc(sizeof(struct myAvl)) ;
new->entradas = (AVL)malloc(sizeof(AVL)) ;
new->entradas= avl_create(compara,NULL,NULL) ;
return new ;
}

int contagem(myAVL avl){
int i=0,r=0 ;
r+=(avl_contador(avl->entradas));
return r ;
}

myAVL insere(myAVL avl , char* entrada ) {
int i ;
```

```
myAVL aux = avl ;
char* temp = (char*)malloc(sizeof(char)*124) ;
strcpy(temp,entrada);
avl_insert(avl->entradas,temp) ;
return avl;
}

char** retorna_elementos(myAVL avl) {
int k=contagem(avl);
char** new = (char**)malloc(sizeof(char*)*k);
char* aux=(char*)malloc(sizeof(char)*128);
int i=0;
for(i=0;i<k;i++)
new[i]=(char*)malloc(sizeof(char)*128);
i=0;
strcpy(aux,"aaaaa");
TVS a = avl_t_alloc(avl->entradas);
while(avl_t_next(a)!=NULL){
avl_t_prev(a);
strcpy(aux,avl_t_next(a));
strcpy(new[i],aux);
i++;
}
return new ;
}


char* str_exp(char* linha){
char* aux = (char*)malloc(sizeof(char)*64) ;
strcpy(aux,"to ") ;
int i ;
char* token = (char*)malloc(sizeof(char)*64) ;
token = strtok(linha, " ") ;
token = strtok(NULL," ") ;
strcat(aux,token) ;
aux[strlen(aux)]='\0' ;
return aux;
}

char* str_other(char* linha){
char* aux = (char*)malloc(sizeof(char)*64) ;
strcpy(aux,"to ") ;
int i ;
char* token = (char*)malloc(sizeof(char)*64) ;
token = strtok(linha, " ") ;
token = strtok(NULL," ") ;
token = strtok(NULL," ") ;
strcat(aux,token) ;
aux[strlen(aux)]='\0' ;
return aux;
}
```

```
void cria_html(myAVL verbos ){

int k ;
int i ;

FILE *fich = fopen("verbos.html" , "w") ;

fprintf(fich, "<html> <head> <meta charset='UTF-8'/> </head> <body>") ;

if(verbos){
fprintf(fich,"<h1> <b>Verbos</b> \n </h1>") ;
k=contagem(verbos);
char** new1 = (char**)malloc(sizeof(char*)*k);
char* aux1=(char*)malloc(sizeof(char)*128);
for(i=0;i<k;i++)
new1[i]=(char*)malloc(sizeof(char)*128);
new1 = retorna_elementos(verbos) ;
for(int i = 0 ; i< k ; i++)
fprintf(fich,"<li>  %s \n </li>" , new1[i]) ;
}

fprintf(fich , "</body> </html>") ;

}
```

5.2 verbos.h

```
    typedef struct myAvl* myAVL ;
myAVL new_myAvl() ;
int contagem(myAVL avl) ;
int compara(const void * avl_a , const void * avl_b , void * param)  ;
myAVL insere(myAVL avl , char* entrada )  ;
char** retorna_elementos(myAVL avl) ;
char* str_exp(char* linha) ;
void cria_html(myAVL verbos ) ;
char* str_other(char* linha) ;
```

Processador de Named Entities

5.3 enamex.c

```
    #include <stdio.h>
#include <stdlib.h>
#include "enamex.h"
#include "avl.h"
#include <string.h>


struct myAvl{
AVL entradas ;
};
```

```
int compara(const void * avl_a , const void * avl_b , void * param) {
char* a = (char*) avl_a ;
char* b = (char*) avl_b ;
return strcmp(a,b) ;
}


myAVL new_myAvl() {
int i ;
myAVL new = (myAVL) malloc(sizeof(struct myAvl)) ;
new->entradas = (AVL)malloc(sizeof(AVL)) ;
new->entradas= avl_create(compara,NULL,NULL) ;
return new ;
}

int contagem(myAVL avl){
int i=0,r=0 ;
r+=(avl_contador(avl->entradas));
return r ;
}

myAVL insere(myAVL avl , char* entrada ) {
int i ;
myAVL aux = avl ;
char* temp = (char*)malloc(sizeof(char)*124) ;
strcpy(temp,entrada);
avl_insert(avl->entradas,temp) ;
return avl;
}

char** retorna_elementos(myAVL avl) {
int k=contagem(avl);
char** new = (char**)malloc(sizeof(char*)*k);
char* aux=(char*)malloc(sizeof(char)*128);
int i=0;
for(i=0;i<k;i++)
new[i]=(char*)malloc(sizeof(char)*128);
i=0;
strcpy(aux,"aaaaa");
TVS a = avl_t_alloc(avl->entradas);
while(avl_t_next(a)!=NULL){
avl_t_prev(a);
strcpy(aux,avl_t_next(a));
strcpy(new[i],aux);
i++;
}
return new ;
}


char* elimina_tags(char* linha){
char* aux = (char*)malloc(sizeof(char)*128) ;
int i ;
for(i=0 ; linha[i]!='>' ; i++) ;
```

```
for(; linha[i]>='A' && linha[i]<='Z' ; i++) ;
i++ ;
int j ;
for(j=0 ; linha[i]!='<' ; j++)
aux[j]=linha[i++] ;
aux[j]='\0' ;
return aux ;
}



void cria_html(myAVL pessoas , myAVL paises , myAVL cidades ){

int k ;
int i ;

FILE *fich = fopen("enamex.html" , "w") ;

fprintf(fich, "<html> <head> <meta charset='UTF-8'/> </head> <body>") ;

// Pessoas

if(pessoas) {
fprintf(fich,"<h1> <b>Pessoas</b> \n </h1>") ;
k=contagem(pessoas);
char** new1 = (char**)malloc(sizeof(char*)*k);
char* aux1=(char*)malloc(sizeof(char)*128);
for(i=0;i<k;i++)
new1[i]=(char*)malloc(sizeof(char)*128);
new1 = retorna_elementos(pessoas) ;
for(int i = 0 ; i< k ; i++)
fprintf(fich,"<li>  %s \n </li>" , new1[i]) ;
}


// PAISES


if(paises) {
fprintf(fich,"<h1> <b>Paises</b> \n </h1>") ;
k=contagem(paises);
char** new2 = (char**)malloc(sizeof(char*)*k);
char* aux2=(char*)malloc(sizeof(char)*128);
for(i=0;i<k;i++)
new2[i]=(char*)malloc(sizeof(char)*128);
new2 = retorna_elementos(paises) ;
for(int i = 0 ; i<k ; i++)
fprintf(fich,"<li>  %s \n </li>" , new2[i]) ;
}

// Cidades

if(cidades) {
```

```
fprintf(fich,"<h1> <b>Cidades</b> \n </h1>") ;
k=contagem(cidades);
char** new3 = (char**)malloc(sizeof(char*)*k);
char* aux3=(char*)malloc(sizeof(char)*128);
for(i=0;i<k;i++)
new3[i]=(char*)malloc(sizeof(char)*128);
new3 = retorna_elementos(cidades) ;
for(int i = 0 ; i<k ; i++)
fprintf(fich,"<li>  %s \n </li>" , new3[i]) ;
}


fprintf(fich , "</body> </html>") ;

}
```

5.4 enamex.h

```
#include <stdlib.h>
typedef struct myAvl* myAVL ;
int compara(const void * avl_a , const void * avl_b , void * param) ;
myAVL new_myAvl() ;
int contagem(myAVL avl) ;
myAVL insere(myAVL avl , char* entrada ) ;
char** retorna_elementos(myAVL avl) ;
char* elimina_tags(char* linha) ;
void cria_html(myAVL listaPessoas , myAVL listaPaises , myAVL listaCidades ) ;
```