Processamento de Linguagens (3º ano de MIEI)

Trabalho Prático nº3
Rede Semântica do Museu do Artista

Relatório de Desenvolvimento

Henrique Faria	João Marques	Nuno Rei
(a82200)	(a81826)	(a81918)

10 de Junho de 2019

Resumo

O presente documento representa o relatório de desenvolvimento do terceiro trabalho prático da unidade curricular *Processamento de Linguagens* do terceiro ano do curso *Mestrado Integrado em Engenharia Informática*. Este trabalho tem como objectivo o desenvolvimento de um processador de linguagem segundo o método da *tradução dirigida pela sintaxe* suportado numa gramática tradutora. Vai ser especificada uma **GIC**, *gramática indepedente de contexto*, e utilizado o par **flex/yacc** de *geradores de compiladores* para criar o processador.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O *YACC* foi escrito por *Stephen C. Johnson* nos inícios de 1970 e é utilizado para gerar analisadores léxicos. O ficheiro yacc possui um conjunto de produções e respetivas ações, isto é, a uma produção está associada uma ação, sendo a produção um conjunto de valores por uma ordem especifica definida pelo programador, a ação é realizada em código *C*. O *FLEX* já referido no trabalho prático número 1 vai ser importante neste trabalho, pois para filtrar os documentos será utilizado o flex para este retornar para o yacc apenas os valores pretendidos.

O terceiro trabalho prático desta unidade curricular consiste na utilização do par **flex/yacc** para a criação de um reconhecedor léxico e sintático. O **enunciado a ser resolvido é o número um**, *Rede Semântica do Museu do Artista*, no qual é pretendido que se filtre os documentos com informações sobre os artistas, suas obras e eventos em que participaram, como também as relações que estes possam ter com outros artistas, sendo estas informações depois utilizadas para criar grafos orientados e interagivéis de forma a mostrar mais ou menos informação.

Este documento vai relatar detalhadamente todas as decisões tomadas para a resolução do problema, desde o raciocínio até à implementação da solução, em que será apresentada a codificação e alguns testes que mostram o seu funcionamento.

O documento começa pela Análise e Especificação, no Capítulo 2, onde se faz uma análise detalhada do problema proposto de modo a compreender o que será lido pelo filtro de texto e formalizar uma ideia sobre os resultados que devem ser obtidos. De seguida no Capítulo 3, Concepção/desenho da Resolução, irão ser apresentados os algoritmos que foram pensados e que serão utilizados para a resolução. No Capítulo 4, Codificação e Testes, vão ser apresentadas todas as decisões tomadas para implementação, bem como problemas surgidos e possíveis alternativas para aquilo que foi codificado, e ainda testes realizados com os respectivos resultados de forma a mostrar o funcionamento da solução apresentada. Por fim no Capítulo 5 termina-se o relatório com uma síntese do que foi dito, conclusões tiradas, possíveis melhorias para trabalho futuro e principais dificuldades encontradas e como poderiam ser superadas.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

No problema descrito no enunciado **número um**, *Rede Semântica do Museu do Artista* é pedido que seja especificada uma **GIC**, no âmbito do ramo artístico musical (escolhido pelo grupo), que permita a criação de nodos do tipo **artista, obra, evento**, bem como ligações entre eles tal como *produziu* (ligar artista a obra), *participou* (ligar artisa a evento) ou ainda *ensinouCom/aprendeuCom/colaborouCom* (para ligar artista com artista). No final deve ser gerado um grafo que permita clicar em nodos para passar para uma página com informações sobre esse nodo.

2.2 Especificação dos Requisitos

2.2.1 Dados

Para este enunciado não foi fornecido um ficheiro de dados com uma gramática concreta de entrada, e como tal o grupo especificou uma da seguinte forma:

```
artista:
nome: NOME-ARTISTA
idade: IDADE-ARTISTA
cidade: CIDADE-ARTISTA
bibliografia: BIBLIOGRAFIA-ARTISTA#
colaborou: COLABOROU-COM-ARTISTA
ensinou: ENSINOU-O-ARTISTA
aprendeu: APRENDEU-COM-ARTISTA
obra:
nome: NOME-OBRA
tipo: TIPO-OBRA
tempo: TEMPO-OBRA
letra: LETRA-OBRA#
evento:
nome: NOME-EVENTO
tipo: TIPO-EVENTO
data: DATA-EVENTO
```

;

Os parâmetros principais do ficheiro de texto são *artista, obra e evento*, pois estes são os parâmetros sem atributos e que definem quais os tipos de parâmetros que tem de ser processados a seguir.

O símbolo ';' representa o final da informação sobre um artista, enquanto que o símbolo '#' é utilizado para saber onde termina a bibliografia e a letra da obra, pois estes dois campos contêm carateres especiais que necessitam ser coletados e para tal teve de ser difinido um símbolo final para o texto.

2.2.2 Pedidos

De seguida apresentam-se os requisitos propostos no enunciado que a solução deve implementar:

a) Criar uma linguagem específica à escolha do grupo que englobe três nodos principais (*Artista, Obra e Evento*) de forma a definir o conhecimento do MvA (Museu virtual do Artista);

b) A linguagem deve permitir a criação de arcos para ligar uns nodos com os outros, existem 3 tipos de arcos: Artista-Artista, Artista-Obra e Artista-Evento;

c) A linguagem tem de incluir ações semânticas de forma a ser possível criar um grafo orientado utilizando o *GraphViz/WebDot* para que seja também possível a navegação ao longo do grafo.

Capítulo 3

Concepção/desenho da Resolução

3.1 Algoritmos

Para a resolução do problema proposto foram criados dois programas, um em *FLEX* e outro em *YACC*. De forma a ser mais eficiente o *Flex* faz o parse do ficheiro e encaminha para o ficheiro *Yacc* apenas a informação pertinente para depois serem criados os ficheiros *DOT e HTML*.

3.1.1 Flex

Para a utilização do *Flex* para o parsing do documento de texto foram primeiros definitos 3 *start condition*, ARTISTA para informações sobre artistas, OBRA para informações sobre as obras e EVENTO para informações sobre os eventos. Para inicializar as start conditions são procuradas as seguintes expressões (todas as expressões não fazem diferenciação entre carateres *UPCASE e DOWNCASE*):

- "(?i:artista):" - expressão regular necessária para inicializar a start condition ARTISTA;
- "(?i:obra):" - expressão regular necessária para inicializar a start condition OBRA;
- "(?i:evento):" - expressão regular necessária para inicializar a start condition EVENTO.

Depois de inicializado uma start condition é possível trocar para uma das outras se forem apanhadas as expressões.

Quando dentro de uma start condition existem certos padrões a encontrar conforme a start condition em que se está, sendo que quando dentro da start condition ARTISTA as expressões a fazer match são (do mesmo modo que as anteriores não fazem diferenciação entre *UPCASE e DOWNCASE*):

- "(?i:nome):.*" - Apanha o nome do artista para depois encaminhar para o Yacc;
- "(?i:idade):.*" - Apanha a idade do artista para depois encaminhar para o Yacc;
- "(?i:cidade):.*" - Apanha a cidade onde nasceu o artista para depois encaminhar para o Yacc;
- "(?i:bibliografia):[^ #]*" - Apanha a bibliografia do artista para depois encaminhar para o Yacc;
- "(?i:colaborou):.*" - Apanha o nome do artista que colaborou com o artista que está a ser processado para depois encaminhar para o Yacc;
- "(?i:aprendeu):.*" - Apanha o nome do artista com quem o artista que está a ser processado aprendeu a sua arte para depois encaminhar para o Yacc;

- "(?i:ensinou):.*" - Apanha o nome do artista que foi ensinado pelo artista que está a ser processado para depois encaminhar para o Yacc;
- ; - Apanha o final das informações sobre o artista, suas obras e eventos.

As expressões criadas para a start condition OBRA são demonstradas abaixo:

- "(?i:nome):.*" - Apanha o nome da obra para encaminhar para o Yacc;
- "(?i:tipo):.*" - Apanha o tipo/géreno da obra em questão e encaminha para o Yacc;
- "(?i:tempo):.*" - Apanha a duração da obra (a obra tem duração por se tratar de música) para encaminhar para o Yacc;
- "(?i:letra):[^ #]*" - Apanha a letra da obra e encaminha para o Yacc;
- ; - Apanha o final das informações sobre o artista, suas obras e eventos.

Para a start condition relacionada com os eventos, EVENTO, as expressões criadas para recolher informações sobre os mesmos são:

- "(?i:nome):.*" - Apanha o nome do evento e passa para o Yacc;
- "(?i:tipo):.*" - Apanha o tipo de evento de que se trata e passa para o Yacc;
- "(?i:data):.*" - Apanha a data na qual se realizou o evento e passa a informação para o Yacc;
- ; - Apanha o final das informações sobre o artista, suas obras e eventos.

3.1.2 Yacc

De seguida apresenta-se a gramática definida para a realização do trabalho.

```
T  = {NOME, IDADE, CIDADE, BIBLIOGRAFIA, TIPO, TEMPO, LETRA, DATA, COLABOROU, APRENDEU,
    ENSINOU}
NT = {prog, grafos, grafo, ligacoes, artista, musicaOuEvento,lista}

prog           -> grafos
grafos         -> grafo ';' grafos
                | &
grafo          -> artista ligacoes
                | &
ligacoes       -> musicaOuEvento ligacoes
                | &
artista        -> NOME IDADE CIDADE BIBLIOGRAFIA lista
musicaOuEvento -> NOME TIPO TEMPO LETRA
                | NOME TIPO DATA
lista          -> COLABOROU lista
                | APRENDEU lista
                | ENSINOU lista
                | &
```

O yacc é o responsável por gerar o ficheiro grafo.dot em *DOT* bem como todos os ficheiros *HTML*, um para cada artista, obra e evento sobre o qual existam dados.

Na criação do ficheiro dot a cada nodo é associado o endereço do ficheiro html correspondente, esta correspondência é feita usando o nome do artista, da obra ou do evento. Para maior facilidade em distinguir os vários tipos de nodos os artistas sobre os quais se possuem informações estão em octogonos roxos, as obras em retângulos amarelos e os eventos em elipses vermelhas.

Consideremos o seguinte exemplo:
artista:
nome:Tiago Miranda
idade:25
cidade:Lisboa
bibliografia: informação do autor
aprendeu:Artur Batalha

obra:
nome:Telemóveis
tipo:pop
tempo:3.10
letra: Eu parti o telemóvel etc
;

Árvore de derivação originada:

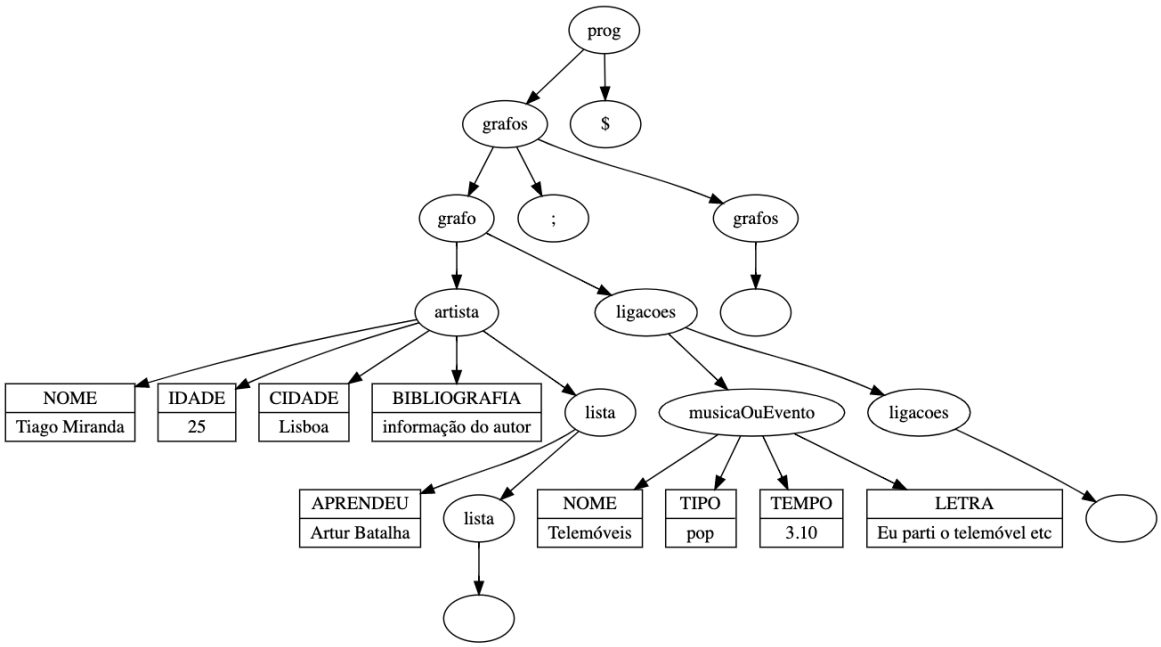

Figura 3.1: Imagem do grafo resultante da divisão gramatical sobre o input

3.2 Compilação e Execução

Para facilitar a compilação do trabalho prático foi criada uma *makefile*, sendo o comando por omissão (make) a compilação do programa flex com as flags *-f e -8* para este ser mais rápido a processar o documento de texto quando executado e para suportar o *utf-8*, compilação do programa yacc com a flag *-v* para a criação do ficheiro *y.output* com informações da gramática definida e gerando também os conflitos e ambiguidades da mesma. De seguida compila o programa *y.tab.c* gerado com a instrução anterior para gerar o executável, depois corre o executável passando lhe o documento de texto com as informações a processar para serem geradas as páginas *HTML* com as informações dos artistas, obras e eventos, gerando ainda também um documento *DOT* para ser processado por um último comando para se criar o grafo final que possa ser aberto num WebBrowser de forma a ser clicável. Para além do comando por omissão foi criado o comando *make mac* e *make linux* que fazem todas as instruções anteriores mais um comando especifíco para abrir o grafo no WebBrowser, apesar de ambos abrirem o grafo no mesmo WebBrowser o comando para tal é diferente.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Para a realização do trabalho foram criados os dois ficheiros para o processamento (um em Flex outro em Yacc) em que o ficheiro Flex processa o ficheiro de input apanhando as expressões regulares nele definidas e passando os valores ao ficheiro Yacc para que com a sua gramática construísse os ficheiros *HTML e DOT*. Para tal foi definido que as informações de um artista (eventos, obras, artista com quem colaborou, etc.) teriam de vir todas seguidas para a gramática funcionar corretamente.

Quando se estão a processar as relações de um artista com outros artista foi necessário guardar no ficheiro Flex temporariamente o nome do artista que se está a processar para concatenar com o nome do artista com quem este tem uma relação, isto foi necessário devido ao Yacc funcionar de modo *BOTTOM-UP*, apesar de o Flex processar primeiro o nome do artista e só depois as suas relações o Yacc utilizava primeiro os valores das relações e só depois o nome do artista o que fazia com que se criasse uma relação de *null* para um artista. Esta medida levou a um extra no processamento para depois separar os nomes dos artistas no lado do Yacc.

Caso um artista tenha uma relação com outro que não exista informação sobre ele no ficheiro de input, o nodo deste é criado na mesma para representar o grafo total da informação, mas este não terá associado a ele uma página *HTML* como os restantes nodos, ficando ainda com um aspeto diferente dos nodos dos outros artistas para criar uma distinção entre nodos com informação e nodos sem informação.

4.2 Testes realizados e Resultados

Mostra-se a seguir o teste realizado e os respectivos resultados obtidos:

No primeiro caso usamos o comando *make* para compilar e executar o programa por nós. Sendo as instruções do comando *make* as seguintes:

1) flex -f -8 museuVirtualArtista.fl

2) yacc -v museuVirtualArtista.y

3) cc -o MvA y.tab.c

4) ./MvA < texto.txt

5) dot -Tsvg grafo.dot -o grafo.svg

Desta forma este mostra-nos a imagem final do grafo, como podemos ver de seguida.

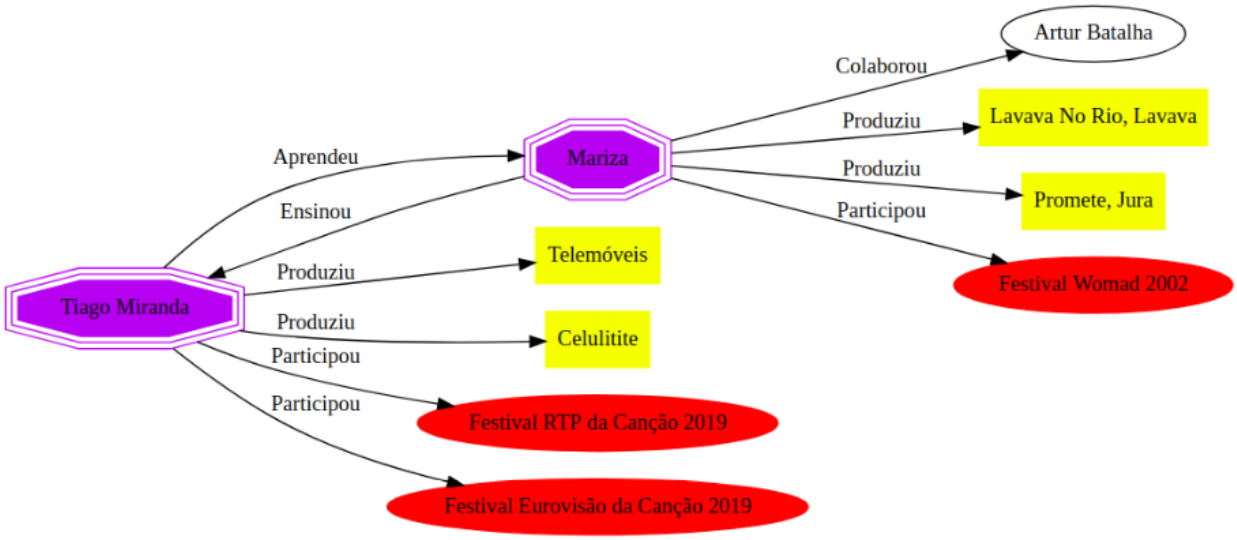

Figura 4.1: Imagem do grafo resultante do comando make

De forma a facilitar a visualização do grafo foi definido que os nodos dos diferentes tipos seriam representados com diferentes formas e diferentes cores, pelo que as obras são representadas por retangulos de cor amarela, os eventos por ovais de cor vermelhas e os artistas por octognos de cor roxa. Existe a peculiaridade de um nodo aparecer com a forma oval e sem cor, isto deve-se ao facto de se tratar de um artista do qual não possuimos informação, apenas sabemos que um outro artista colaborou com este ou aprendeu com outro ou ensinou outro.

Todos os nodos são clicáveis excetuando os nodos sem informação (ovais sem cor), pelo que se o nodo "Mariza" do exemplo for clicado aparecerão as informações do artista, sendo estas as vizualizadas a seguir:

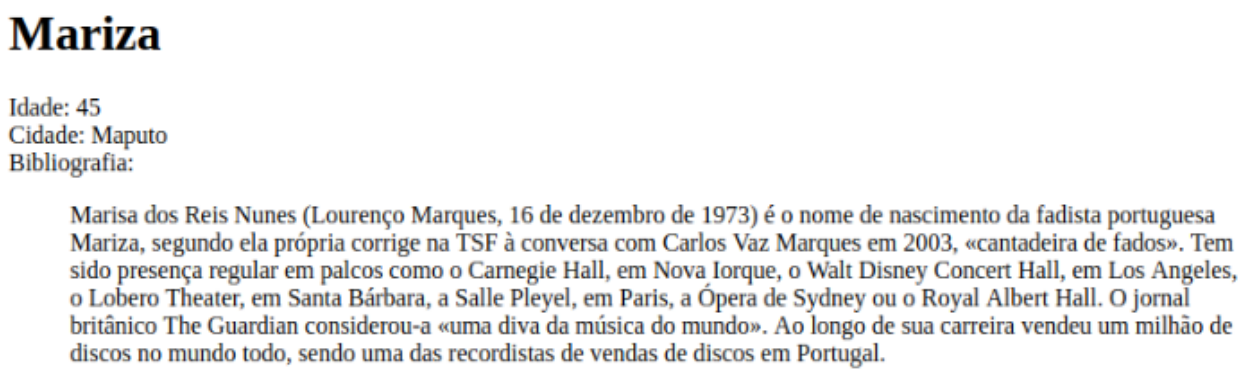

Figura 4.2: Resultado de clicar no nodo "Mariza"

Capítulo 5

Conclusão

Este trabalho prático que consistia no desenvolvimento de um processador de linguagem, permitiu aprimorar conhecimento, e aplicar algum já adquirido, acerca da utilização do par **flex/yacc** para a *geração de compiladores*.

Todos os requisitos foram cumpridos, e no final foi obtido um grafo clicável que permite uma navegação intuitiva para páginas com infomação acerca da entidade, quando um dos **três** tipos de nodos é clicado.

Contudo poderiam ser feitas melhorias na **GIC** especificada, de modo a que esta ficasse mais legível, e futuramente poderá ser adicionada mais informação às entidades de modo a obter uma enciclopédia mais informativa, o que apenas implicaria ligeiras mudanças na *gramática concreta de entrada* e consequentemente no par **flex/yacc**.

Apêndice A

Código do Programa

A.1 Flex

```
%{
    #include <stdio.h>
    char* myartista;
    int comp;
%}
%option noyywrap

%x ARTISTA OBRA EVENTO

%%

(?i:artista):             {BEGIN ARTISTA;}
(?i:obra):                {BEGIN OBRA;}
(?i:evento):              {BEGIN EVENTO;}

<ARTISTA>{
    (?i:nome):.*          {yylval.c = strdup(yytext+5); myartista = yylval.c;
                           comp = strlen(myartista); return NOME;}
    (?i:idade):.*         {yylval.n = atoi(strdup(yytext+6)); return IDADE;}
    (?i:cidade):.*        {yylval.c = strdup(yytext+7); return CIDADE;}
    (?i:bibliografia):[^#]* {yylval.c = strdup(yytext+13); return BIBLIOGRAFIA;}
    (?i:colaborou):.*     {char* tmp = malloc(sizeof(char)*(comp+2+strlen(yytext+10)));
                           sprintf(tmp,"%s/%s",strdup(yytext+10),myartista);
                           yylval.c = strdup(tmp); return COLABOROU;}
    (?i:aprendeu):.*      {char* tmp = malloc(sizeof(char)*(comp+2+strlen(yytext+9)));
                           sprintf(tmp,"%s/%s",strdup(yytext+9),myartista);
                           yylval.c = strdup(tmp); return APRENDEU;}
    (?i:ensinou):.*       {char* tmp = malloc(sizeof(char)*(comp+2+strlen(yytext+8)));
                           sprintf(tmp,"%s/%s",strdup(yytext+8),myartista);
                           yylval.c = strdup(tmp); return ENSINOU;}
    ;                     {return yytext[0];}
    (?i:obra):            {BEGIN OBRA;}
```

```
    (?i:evento):          {BEGIN EVENTO;}
    .|\n                  {;}
}


<OBRA>{
    (?i:nome):.*          {yylval.c = strdup(yytext+5); return NOME;}
    (?i:tipo):.*          {yylval.c = strdup(yytext+5); return TIPO;}
    (?i:tempo):.*         {yylval.x = atof(strdup(yytext+6)); return TEMPO;}
    (?i:letra:)[^#]*      {yylval.c = strdup(yytext+6); return LETRA;}
    ;                     {return yytext[0];}
    (?i:artista):         {BEGIN ARTISTA;}
    (?i:evento):          {BEGIN EVENTO;}
    .|\n                  {;}
}

<EVENTO>{
    (?i:nome):.*          {yylval.c = strdup(yytext+5); return NOME;}
    (?i:tipo):.*          {yylval.c = strdup(yytext+5); return TIPO;}
    (?i:data):.*          {yylval.c = strdup(yytext+5); return DATA;}
    ;                     {return yytext[0];}
    (?i:artista):         {BEGIN ARTISTA;}
    (?i:obra):            {BEGIN OBRA;}
    .|\n                  {;}
}
(.|\n)                    {;}

%%
```

A.2 Yacc

```
%{
    #define _GNU_SOURCE
    #include <stdio.h>
    #include <string.h>
    #include <math.h>
    #include <unistd.h>
    void yyerror(char *s);
    char* prepareString(char* str);
    int yylex();
    FILE* fd1,*fd2,*fd3,*fd4;
    char* myArtista;
%}

%token NOME IDADE CIDADE TIPO TEMPO DATA COLABOROU APRENDEU ENSINOU PARTICIPOU
            PRODUZIU BIBLIOGRAFIA LETRA
```

```
%union{
    int n;
    float x;
    char* c;
}

%type <n> IDADE
%type <x> TEMPO
%type <c> grafos grafo artista ligacoes lista musicaOuEvento NOME CIDADE TIPO DATA
            COLABOROU APRENDEU ENSINOU PARTICIPOU PRODUZIU BIBLIOGRAFIA LETRA

%%

prog    : grafos                        {fd1 = fopen("grafo.dot","w");
                                         fprintf(fd1,"digraph {\nrankdir=LR;\n%s\n}",$1);
                                        }
        ;

grafos  : grafo ';' grafos              {asprintf(&$$,"%s\n%s",$1,$3);}
        |                               {$$="";}
        ;

grafo   : artista ligacoes              {asprintf(&$$,"%s\n%s",$1,$2);}
        |                               {$$="";}
        ;

ligacoes: musicaOuEvento ligacoes       {asprintf(&$$,"%s\n%s",$1,$2);}
        |                               {$$="";}
        ;

artista : NOME IDADE CIDADE BIBLIOGRAFIA lista
                                      {myArtista = strdup($1);
                                       char* f = malloc(sizeof(char)*strlen($1)+6);
                                       sprintf(f,"%s.html",$1);
                                       // aqui escreve para dot
                                       asprintf(&$$,"\"%s\"[URL=\"%s\",shape=tripleoctagon,
                                         color=purple,style=filled];\n%s",$1,f,$5);
                                       fd2=fopen(f,"w");
                                       // a partir daqui para html
                                       fprintf(fd2,"<html> \n\t<head> \n\t<h1>\n\t %s \n\t
                                         </h1> \n\t</head> \n\t<body> \n\t Idade: %d <br>
                                         \n\t Cidade: %s <br>\n\t Bibliografia: <blockquote
                                         style=\"height:300px; overflow: hidden; min-height:
                                         300 px;\"> %s </blockquote> \n\t</body> \n</html>",
                                         $1,$2,$3,prepareString($4));
                                      }
         ;
```

```
musicaOuEvento: NOME TIPO TEMPO LETRA
                                    {char* f = malloc(sizeof(char)*strlen($1)+6);
                                     sprintf(f,"%s.html",$1);
                                     // aqui escreve para dot
                                     asprintf(&$$,"\"%s\"[shape=box,color=yellow,
                                       style=filled];\n\"%s\"[URL=\"%s\"];\n\"%s\" ->
                                       \"%s\"[label=\"Produziu\"];",$1,$1,f,myArtista,$1);
                                     fd3=fopen(f,"w");
                                     // a partir daqui para html
                                     fprintf(fd3,"<html>\n\t <head> \n\t<h1> %s \n\t</h1>
                                       \n\t</head> \n\t<body> \n\t Tipo: %s <br>\n\t Tempo:
                                       %.2f <br><br>\n\t Letra: <br> <p> %s </p>\n\t</body>
                                       \n</html>",$1,$2,$3,prepareString($4));
                                    }
              | NOME TIPO DATA      {char* f = malloc(sizeof(char)*strlen($1)+6);
                                     sprintf(f,"%s.html",$1);
                                     // aqui escreve para dot
                                     asprintf(&$$,"\"%s\"[color=red,style=filled];\n\"%s\"
                                       [URL=\"%s\"];\n\"%s\" -> \"%s\"[label=\"Participou\"
                                       ];",$1,$1,f,myArtista,$1);
                                     fd4=fopen(f,"w");
                                     // a partir daqui para html
                                     fprintf(fd4,"<html>\n\t <head> \n\t<h1> %s \n\t</h1>
                                       \n\t</head> \n\t<body> \n\t Tipo: %s <br>\n\t Data:
                                       %s \n\t</body> \n</html> ",$1,$2,$3);
                                    }
              ;

lista   : COLABOROU lista           {char* colab = malloc(sizeof(char)*strlen($1));
                                     int i = 0,j = 0, index;
                                     for(;$1[i]!='/';i++) colab[i] = $1[i];
                                     index = i;
                                     char* artista = malloc(sizeof(char)*strlen($1));
                                     i++;
                                     for(; $1[i]!='\0';j++,i++) artista[j] = $1[i];
                                     artista[j] = colab[index] = '\0';
                                     char* f = malloc(sizeof(char)*strlen($1)+6);
                                     sprintf(f,"%s.html",colab);
                                     if(fopen(f,"r")){
                                         asprintf(&$$,"\"%s\" -> \"%s\"[URL=\"%s\", label=
                                           \"Colaborou\"];\n%s",artista,colab,f,$2);
                                     } else{
                                          asprintf(&$$,"\"%s\" -> \"%s\"[ label=\"Colaborou
                                            \"];\n%s",artista,colab,$2);
                                     }
                                    }
```

```
        | APRENDEU lista              {char* apren = malloc(sizeof(char)*strlen($1));
                                      int i = 0,j = 0, index;
                                      for(;$1[i]!='/';i++) apren[i] = $1[i];
                                      index = i;
                                      char* artista = malloc(sizeof(char)*strlen($1));
                                      i++;
                                      for(; $1[i]!='\0';j++,i++) artista[j] = $1[i];
                                      artista[j] = apren[index] = '\0';
                                      char* f = malloc(sizeof(char)*strlen($1)+6);
                                      sprintf(f,"%s.html",apren);
                                      if(fopen(f,"r")){
                                           asprintf(&$$,"\"%s\" -> \"%s\"[URL=\"%s\", label=
                                             \"Aprendeu\"];\n%s",artista,apren,f,$2);
                                      } else{
                                           asprintf(&$$,"\"%s\" -> \"%s\"[ label=\"Aprendeu
                                             \"];\n%s",artista,apren,$2);
                                      }
                                     }
        | ENSINOU lista             {char* ensi = malloc(sizeof(char)*strlen($1));
                                      int i = 0,j = 0, index;
                                      for(;$1[i]!='/';i++) ensi[i] = $1[i];
                                      index = i;
                                      char* artista = malloc(sizeof(char)*strlen($1));
                                      i++;
                                      for(; $1[i]!='\0';j++,i++) artista[j] = $1[i];
                                      artista[j] = ensi[index] = '\0';
                                      char* f = malloc(sizeof(char)*strlen($1)+6);
                                      sprintf(f,"%s.html",ensi);
                                      if(fopen(f,"r")){
                                          asprintf(&$$,"\"%s\" -> \"%s\"[URL=\"%s\", label=
                                            \"Ensinou\"];\n%s",artista,ensi,f,$2);
                                      } else{
                                           asprintf(&$$,"\"%s\" -> \"%s\"[ label=\"Ensinou
                                             \"];\n%s",artista,ensi,$2);
                                      }
                                     }
        |                           {$$="";}
        ;

%%
#include "lex.yy.c"

void yyerror(char *s){fprintf(stderr,"ERRO:%s\nLine:%d\n",s,yylineno);}

int main() {
    yyparse();
    return 0;
}
```

```
 char* prepareString(char* str){
    int x,y;
    char* res = malloc(sizeof(char)*strlen(str)*2);
    for(x=0,y=0;str[x] != '\0';x++,y++){
        if(str[x] == '\n'){
            res[y] = '<';
            res[y+1] = 'b';
            res[y+2] ='r';
            res[y+3] = '>';
            y+=3;
        } else res[y] = str[x];
    }
    res[y] = '\0';
    return res;
}
```