Universidade do Minho

Licenciatura em Engenharia Informática

Processamento de Linguagens

3º Ano, 2º Semestre, Ano lectivo 2012/2013

Linguagem para definição de dados genealógicos

Trabalho de Prático 2

Autores:

André Santos nº60994

Helena Alves nº61000

Pedro Carneiro nº61085

Braga, 2 de Junho de 2013

Conteúdo

1 Introdução

A resolução do trabalho prático descrita no presente relatório refere-se ao desenvolvimento de um processador para uma linguagem referente às relações de parentesco e outros dados de cariz genealógico.

Pretende-se neste trablaho que seja implementada uma nova linguagem para definição de dados genealógicos, uma gramática tradutora, onde o output será uma página em Html, com as informações de cada indivíduo.

Assim, neste trabalho prático pretende-se aplicar os conceitos adquiridos na Unidade Curricular de Processamento de Linguagens associados à utilização das ferramentas Flex e Yacc, para a nálise léxica e sintáctica da linguagem, respectivamente, aliados à linguagem de programação C.

2 Descrição do problema

A linguagem para a definição de dados genealógicos, envolve as diferentes relações de parentesco, como também, outros dados genealógicos, como diferentes histórias e eventos de um dado indivíduo. Desta forma, foi definida uma notação compacta e formal para este tipo de dados. Esta notação cobre elementos, como os próximos exemplos:

- **Nome dos indivíduos**

```
Ana Rita Guimarães Faria
Ana Rita/Guimarães Faria (separação nome aplelido)
Ana Rita Guimarães Faria\%2 (distinção entre 2 elementos com o mesmo nome)
```

- **Datas**

```
*05/1910 - Data de Nascimento (Mês/Ano)
+06/1985 - Data de Morte (Mês/Ano)
+c03/02/2000 - Data de Falecimento (por volta de 03/02/2000)
*c05/09/1942 - Data de Nascimento (por volta de 05/09/1942)
CC 1996 - Data de casamento
```

- **Relações de parentesco**

```
M IndivíduoM - Indivíduo tem como mãe IndividuoM
P IndivíduoP - Indivíduo tem como pai IndividuoP
MM IndivíduoMM - Indivíduo tem como avó materna IndividuoMM
PP IndivíduoPP - Indivíduo tem como avô paterno IndividuoPP
MP IndivíduoMP - Indivíduo tem como avó paterna IndividuoMP
PM IndivíduoPM - Indivíduo tem como avô materno IndividuoPM
CC IndivíduoCC - Casamento com IndivíduoCC
F IndivíduoF - Filho resultante com casamento atrás descrito
```

- **Sexo**

```
S feminino - Indivíduo do sexo feminino
M masculino - Indivíduo do sexo Masculino
```

- **Histórias e Fotos**

```
FOTO foto.jpg - Foto do Indivíduo
HIST historia.txt - Historia do Indivíduo
```

- **Eventos**

```
EVENTO [7] - 7 é o identificador do evento
N "Nome" - Nome do evento
D "Descrição" - Descrição do evento
ev 4 05/09/1930 - Indivíduo foi ao eveneto cujo identificador é 4 em 05/09/1930
```

Importante salientar, que todos os indivíduos possuem um determinado identificador, assim como os eventos, estando este representado entre parêntesis rectos. Toda a informação que se encontra a seguir a um nome de um indivíduo, apenas diz respeito a esse indivíduo, até ao nome do próximo indivíduo. Toda a implemtação desta linguagem, será demonstrada e discutida ao longo do relatório.

3 Implementação

O desenvolvimento deste trabalho prático consistiu em criar um programa que lê especificações de dados geneológicos de vários indivíduos, através de uma linguagem criada de raíz, onde o output final é uma página Html com as informações de cada indivíduo.

O problema pode ser dividido em duas fases: a análise léxica e a análise sintáctica.

3.1 Análise Léxica

Na análise léxica a ferramenta utilizada é o flex. No flex é feito o reconhecimento léxico da linguagem, onde são identificados os símbolos terminais, designados também por tokens, em letras maiúsculas, como o NOME, TEXTO, DATA, entre outros. Um token é uma representação de um símbolo abstracto usado durante a análise sintáctica. O nome dado aos diferentes tokens não segue nenhuma regra, podendo o utilizador atribuir qualquer tipo de nome.

Estes símbolos terminais (tokens) são identificados graças ao uso de expressões regulares. Quando uma expressão regular é identificada, é feito um return do token encontrado, de modo a informar o yacc que tal token foi encontrado. Quando é necessário que o yacc tenha acesso ao texto que foi processado, como por exemplo, quando é encontrado um nome ou um número, o flex escreve esse mesmo conteúdo para uma union, definida no yacc, mas que é partilhada por ambos. A cada tipo de dados diferentes tem de corresponder um campo diferente na union, tomando o exemplo de um nome, para este terá de haver um char*, enquanto que para um número terá de haver um int.

Deste modo, todo o parser do input é efectuado no flex.

3.2 Análise Sintáctica

A ferramenta utilizada na análise sintáctica é o yacc. No yacc é efectuado o reconhecimento sintáctico da linguagem, através das produções elaboradas. Uma produção pode ser definida como uma sequência de símbolos terminais, de outras produções ou apenas ser vazia. Deste modo, é possível saber o que é ou não aceitável em determinado contexto da linguagem, seguindo apenas a produção.

Uma produção pode ou não ter um tipo associado. Por exemplo, a produção Nome é do tipo char*. Significa, então, que a produção Indíviduo, que é do tipo Individuo*, pode fazer directamente $$->nome = $1, onde $1 é o resultado da produção Nome, sendo também o primeiro campo da produção Indivíduo. O mesmo acontece com as outras produções.

A função chave é a 'indivRec', que cria um novo indivíduo a partir de dois indivíduos recebidos como parâmetro, fazendo a união das informações entre eles. Por exemplo, a produção Informacao é do tipo Individuo*. Se esta produção derivar numa FOTO ESPACOS TEXTO, a produção retornará um indivíduo que terá como informação só um elemento na sua lista de fotos. Assim, a produção ListaInf (Lista de informações de um indivíduo), recursivamente, processará uma informação do indivíduo e juntá-la-à ao indivíduo resultante da acção recursiva.

Deste modo, é possível concluir que o utilizador pode definir diferentes tipo de acções a serem invocadas quando uma regra é reconhecida. Estas acções têm a capacidade de retornar valores e fazer uso de valores retornados por outras acções.

4 Estrutura de dados

Os diferentes tipos de dados foram guardados em diferentes estruturas de dados. Antes de mais, importante referir que foi utilizada a biblioteca GLib para todas as estruturas de dados.

Deste modo, foi definida uma tabela de hash para guardar as informações relativamente aos indivíduos, onde a chave da respectiva tabela diz respeito ao identificador do indivíduo. Caso o individuo não possua identificador, o programa atribui-lhe um automaticamente.

```
typedef struct sIndividuo{
    int id;
    char* nome;
    int sexo;                        // 0-ND 1-M 2-F
    char* data_nascimento;
    char* data_morte;
    char* data_casamento;
    char* foto;

  int idPai ;
  int idMae ;
  int idConjugue ;

  GList* historias;                // Glist de char*
  GList* eventos;                  // Glist de Evento*
  GList* parentescos;              // GList de Relacoes por resolver*

}Individuo;
```

Esta estrutura guarda o identificador e o sexo do indivíduo, representados por um inteiro; guarda o nome, foto, data de nascimento, data de falecimento e data de casamento, representados por uma string.

De maneira a saber quem são o pai, a mãe e o conjugue de um dado indivíduo, também são guardados os identificadores dos mesmos.

Também foram definidas listas ligadas responsáveis por guardar os eventos e histórias de cada indivíduo. A lista ligada de histórias corresponde a uma lista ligada de strings, enquanto que a lista ligada responsável por guardar os eventos de um indivíduo, corresponde

a uma lista ligada do tipo Evento. No mesmo contexto, também foi definida outra lista ligada, responsável por guardar as relações de parentesco incompletas, lista ligada esta do tipo ParentPorResolver, que é apresentada de seguida:

```
typedef struct sParentPorResolver {


  char* parentesco ;
  int id ;
} ParentPorResolver ;
```

Relativamente aos eventos de um dado indivíduo, estes foram guardados numa lista ligada. Cada evento possui um identificador, assim como o nome do evento e uma descrição.

```
typedef struct sEvento{
    int id;
    char* nome;
    char* descricao;
    char* data;
}Evento;
```

5 Execução

De seguida é apresentado um exemplo de um indivíduo dos nossos dados teste:

```
Joaquim Manuel Santos Oliveira *21/01/1940 +10/10/2012 [10]
S Masculino
M Maria dos Anjos/Santos *10/02/1915 +08/06/1980 [2]
P António Manuel Ferreira Oliveira +c09/09/1987 [449]
MM [14]
MP [16]
FOTO "../Dados/Fotos/10.jpg"
HIST "../Dados/Historico/festa.txt"
CC 06/06/1965 [11]
F André Ricardo Oliveira Faria *13/04/1960 [62] {
    S Masculino
FOTO "../Dados/Fotos/62.jpg"
HIST "../Dados/Historico/latim.txt"
HIST "../Dados/Historico/pequeno.txt"
ev 6 24/10/1984
ev 7 1988
}
```

A partir deste dado teste, é possível verificar que se trata do indivíduo de nome Joaquim Manuel Santos Oliveira, que nasceu em 21/01/1940 e faleceu em 10/10/2012 e cujo identificador é o número 10.

Este indivíduo é do sexo masculino e tem como mãe, Maria dos Anjos/Santos (Nome: Maria dos Anjos, Apelido:Santos), que nasceu em 10/02/1915 e faleceu em 08/06/1980, cujo identificador é o número 2. Relativamente ao pai do indivíduo, este chama-se António Manuel Ferreira Oliveira, não possui data de nascimento nem de falecimento, mas sabe-se que casou em 09/09/1987 e o identificador é o 449. Desta forma, nas informações do indivíduo em causa, nomeadamente, nos identificadores da mãe e do pai, são colocados os números 2 e 449, respectivamente. Assim como, a restante informação vista até agora.

Relativamente à avó materna, identificada por 'MM [14]', só se tem referência ao identificador da mesma, representado pelo número 14. Isto quer dizer que o indivíduo, cujo identificador é o 14 já existe, ou seja, já foi definido. Desta forma, é colocado nas informações

da mãe do indivíduo em causa, nomeadamente, no identificador da mãe, o número 14. Em relação à avó paterna, 'MP [16]', o procedimento é análogo ao da avó materna.

É possível também verificar que o indivíduo em causa possui uma história e uma foto. Este indivíduo casou-se em 06/06/1965 com o indivíduo, cujo identificador é o número 11. Assim, é colocado nas informações do indivíduo, ou seja, no identificador do conjugue o identificador número 11.

Para finalizar, este indivíduo possui um filho, de nome André Ricardo Oliveira Faria, que nasceu em 13/04/1960 e cujo identificador é o número 62. Este indivíduo possui algumas informações, informações estas que se encontram entre '{ }'. Podemos observar que se trata de um indivíduo do sexo masculino, possui uma foto e duas histórias. Esteve presente em dois eventos, um realizou-se em 24/10/1958 e é identificado pelo número 6, enquanto que o outro possui como identificador o número 7, e realizou-se no ano de 1988. Estes dois eventos já foram definidos, uma vez que também já possuem identificador (campo obrigatório), e são apresentados de seguida:

```
EVENTO [6]
N "Corrida Solidariedade"
D "Corrida no porto da caritas para ajudar os mais desfavorecidos"

EVENTO [7]
N "Visita ao estádio"
D "Visita ao estádio do Jamor para sessão de autografos dos jogadores do
Guimarães"
```

Os eventos possuem um identificador representado entre parêntesis rectos, um nome e uma decrição.

Relativamente a alguns factos importantes, destacam-se as datas, onde estas podem apresentar três formatos: dia/mês/ano, mês/ano ou apenas ano. Por outro aldo, quando se faz referência a um dado indivíduo a partir do seu identificador, como por exemplo, 'MM [14]' ou 'CC 06/06/1965 [11]', estes indivíduos têm obrigatorimaente de existir. Em casos em que indivíduos possuem avós paternas ou maternas, é obrigatório que esse indivíduo tenha referência ao pai e à mãe, respectivamente.

Após todos estes procedimentos descritos ao longo relatório, é necessário uma makefile para executar o programa. A partir desta makefile, são possíveis os seguintes comandos:

- **make install:** instala o programa no sistema e coloca o manual acessível;
- **make clean:** apaga os ficheiros gerados pelo yacc e pelo flex, como também o executável;
- **make uninstall:** desinstala o programa e manual deixa de estar disponível;
- **make test:** executa o programa com os nossos dados teste, gerando o respectivo html;

De seguida, segue o manual de utilização:

Figura 1: Manual de utilização

6 Output

Como já foi referido anteriormente o output do problema é uma página Html, onde são apresentadas todas as informações de cada indivíduo, como mostra a figura seguinte:

Figura 2: Informações do Indivíduo

7 Conclusão

Como forma de conclusão pode-se referir que o trabalho prático foi, no entender do grupo, realizado com sucesso. As funcionalidades propostas pelo enunciado foram implementadas com sucesso, juntamente com a adição de algumas funcionalidades extras.

De uma forma geral, os resultados produzidos foram satisfatórios, de modo a obter uma solução simples e eficiente.