Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Programação e Desenvolvimento de Software I
1º Semestre de 2020

Trabalho Prático 3

Faça um programa que simule cadastros e consultas em uma árvore genealógica. A árvore genealógica mantém o grau de parentesco entre os ascendentes (pai/avô/bisavô…) e os seus descendentes (filho/neto/bisneto...). Ao representar árvores genealógicas é comum que as pessoas estejam diretamente conectadas somente a seus parentes diretos (filhos ou pais). Porém, é possível alcançar parentes distantes da pela árvore genealógica "caminhando" entre as gerações. A figura a seguir ilustra uma representação de árvore genealógica.

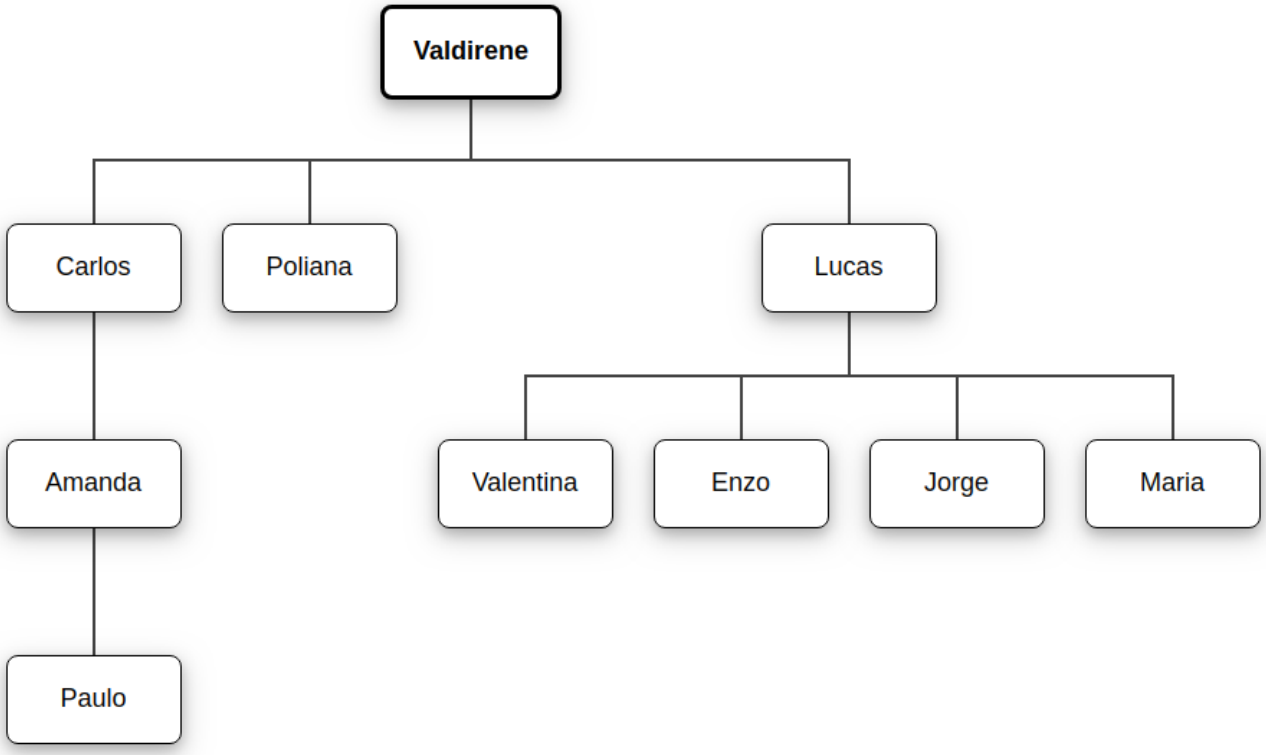

Figura 1 – Árvore Genealógica de exemplo

O "ancestral" mais antigo nesta árvore genealógica é a Valdirene. A Valdirene tem três filhos: Carlos, Poliana e Lucas. Cada um destes filhos pode possuir ou não outros filhos. Carlos possui somente uma filha (Amanda), Poliana não possui filhos, e Lucas possui quatro filhos (Valentina, Enzo, Jorge e Maria). A árvore genealógica permite calcular o nível de parentesco entre as pessoas. Entre Paulo e Amanda há somente um nível de diferença na árvore, porque eles são filho e mãe. Porém, entre Paulo e Valdirene há 3 níveis de diferença na árvore, porque eles são bisneto e bisavó.

Faça um programa que descubra o nível de parentesco entre duas pessoas de uma família. Para isto, uma estrutura (struct) deverá ser criada para representar o tipo Pessoa. O tipo Pessoa deve armazenar valores de acordo com a tabela abaixo:

Nome da estrutura	**Campos da estrutura**
Pessoa	• nome – string de até 50 caracteres • idade – inteiro • filhos – ponteiro do tipo **Pessoa** • numFilhos – inteiro

O atributo filhos será um vetor que armazenará os filhos (descendentes diretos) da pessoa. À medida em que novos filhos forem inseridos pela entrada de dados deve-se incrementar o atributo *numFilhos* e aumentar o tamanho do vetor filhos para caber os novos filhos informados. Para isso, deve-se realocar o vetor filhos usando a função *realloc*, da biblioteca *stdlib.h*.

O programa deve funcionar da seguinte forma:

1. Inicialmente deve-se armazenar em um inteiro **n** o número de pessoas a serem cadastradas;

2. Em seguida deve-se ler os dados destas **n** pessoas. A leitura dos dados deve ser feita da seguinte forma:

   2.1 Cada uma das **n** linhas seguintes representa uma pessoa diferente;

   2.2 Cada linha possui três dados: o nome da pessoa, a idade da pessoa, e o nome do seu **ascendente direto** (nome do pai ou da mãe);

   2.3 Repare que não é necessário criar um atributo para o "ascendente direto" na struct Pessoa. **PORÉM**, o ascendente direto deve ter seus filhos e filhas cadastrados no atributo "filhos". Para encontrar o ascendente direto, deve-se usar a função **buscaPessoa** descrita no item 5;

   2.4 Repare também que não é informado o número de descendentes que a Pessoa tem. O ideal é realocar o atributo "filhos" usando a função realloc sempre que um novo filho é informado pela entrada de dados.

3. Após o cadastro das pessoas, deve armazenar em um inteiro **m** o número de consultas de nível de parentesco a serem feitas;

4. Em seguida deve-se ler os parâmetros das **m** consultas. A leitura das consultas deve ser feita da seguinte forma:

   4.1 Cada linha representa uma consulta diferente;

   4.2 Cada consulta é representada por um par de nomes, sendo que o primeiro nome é o do descendente (filho/neto/bisneto…) e o segundo nome é o do ascendente (pai/avô/bisavô…);

   4.3 Para cada consulta, deve-se imprimir o nível de parentesco entre o par de nomes mencionado na consulta. Para isso, deve-se usar a função buscaPessoa descrita a seguir passando o parâmetro imprimeNivel com valor 1.

5. Deve-se implementar a função **recursiva** "buscaPessoa (pessoaAsc, nomeProcurado, nivel, imprimeNivel);" que descobre **recursivamente** e imprime o nível de parentesco entre as pessoas mencionadas nas consultas. Caso o ascendente seja pai ou mãe do descendente, a função deve imprimir 1. Caso o ascendente seja avô ou avó, a função deve imprimir 2 e assim sucessivamente. A chamada da função é apresentada a seguir.

struct Pessoa* **buscaPessoa** (struct Pessoa *pessoaAsc, char nomeProcurado[], int nivel, int imprimeNivel);

O primeiro parâmetro, "pessoaAsc", é um ponteiro para o antecessor mais distante a ser considerado na busca. Quando esta função é usada somente para encontrar o struct de uma pessoa (necessário para o item 2.3) deve-se passar para o parâmetro "imprimeNivel" o valor 0 (informando que desta vez não se deve imprimir o nível) e passar para o parâmetro "pessoaAsc" a primeira pessoa informada na entrada de dados. Por padrão dos casos de teste do Moodle, a primeira pessoa informada na entrada de dados sempre será o antecessor mais distante da família (o equivalente à Valdirene no exemplo da Figura 1). Dessa forma, a busca considerará todas as pessoas da árvore genealógica.

O segundo parâmetro é o nome da pessoa que está sendo procurada, conforme o informado na entrada de dados. Após encontrar a pessoa procurada, a função buscaPessoa deve retornar um ponteiro para a struct da pessoa encontrada. Contudo, antes de retornar a pessoa encontrada, deve-se testar o valor do parâmetro "imprimeNivel". Se "imprimeNivel" for igual a 1, imprima o nível de parentesco entre o ascendente e o descendente envolvido na consulta. Se "imprimeNivel" é igual a zero, não imprima o nível e somente retorne a struct da pessoa encontrada. Dessa forma, a função buscaPessoa pode ser

usada sempre que você tem o nome da pessoa, mas não tem a struct da pessoa. Neste caso, basta utilizar o parâmetro "imprimeNivel" com valor igual a 0 que a função irá buscar e retornar o struct da pessoa sem imprimir outras informações.

O terceiro parâmetro é o nível de parentesco entre o ascendente e o descendente da consulta. Este nível de parentesco deve ser incrementado através de **recursividade** à medida em que se caminha pela árvore genealógica.

O quarto parâmetro informa se o nível de parentesco deve ser impresso ou não. O nível de parentesco **não deve ser impresso** quando se está somente procurando uma pessoa pela árvore genealógica (por exemplo, para cumprir com o item 2.3). E o nível de parentesco **deve ser impresso** quando se está executando as consultas solicitadas pela entrada de dados, conforme descrito no item 4.3.

6. Por fim, deve-se liberar a memória alocada usando o comando "free()".

Um exemplo da execução do código é apresentado a seguir.

Entrada:	5 João 50 NULL Maria 27 João Joana 6 Maria Carlos 30 João Gisele 2 Maria 3 Gisele João Carlos João Joana Maria
Saída:	2 1 1

Nos casos de teste do Moodle é garantido que, com exceção da primeira Pessoa da entrada de dados, todas próximas pessoas são descendentes de alguém já previamente informado. Não serão incluídos casos de teste sobre tio ou tia. E não serão incluídos casos de teste com duas ou mais pessoas com o mesmo nome.