Prova 2

Algoritmos e Estruturas de Dados I - turma TM2

Professor: Pedro O.S. Vaz de Melo

29 de outubro de 2013

Nome: _____

escrevendo o meu nome eu juro que seguirei o código de honra

Código de Honra para este exame (baseado no *Honor Code* da Universidade de Stanford):

- Não darei ajuda a outros colegas durante os exames, nem lhes pedirei ajuda;
- não copiarei nem deixarei que um colega copie de mim;
- não usarei no exame elementos de consulta não autorizados.

Informações importantes:

- Em questões que pede um **programa**, este deve ser completo, com bibliotecas (incluindo, quando necessário, a biblioteca `prova2.h`), função main, etc. Se deve ser feita uma **função**, somente a função é suficiente. Se deve ser feito um **procedimento**, somente o procedimento é suficiente.
- A interpretação das questões da prova faz parte do critério de avaliação. Caso tenha dúvida sobre a sua interpretação de uma determinada questão, escreva as suas suposições na resolução da mesma.

Referências:

Função/Operador	Descrição	Exemplo
`rand()`	gera um número aleatório inteiro entre 0 e `RAND_MAX`	rand() pode gerar 41
`RAND_MAX`	o maior número possível que pode ser gerado por `rand()`	`RAND_MAX = 32767`

1. (*6 points*) Uma rede social de amizades pode ser representada por uma matriz de adjacência $n \times n$ de n colunas e n linhas. Cada linha (ou coluna) i contém as relações da pessoa n_i. Considere a matriz de adjacência abaixo:

id	n_0	n_1	n_2	n_3	n_4
n_0	0	1	1	0	1
n_1	1	0	0	1	0
n_2	1	0	0	0	0
n_3	0	1	0	0	1
n_4	1	0	0	1	0

Esta matriz representa uma rede social entre 5 pessoas: n_0, n_1, n_2, n_3 e n_4. Além disso, quando a posição (i, j) da matriz é 1, então as pessoas n_i e n_j são amigas entre si. Caso a posição (i, j) da matriz é 0, então n_i e n_j não são amigas. Observe que a pessoa n_0 é amiga das pessoas n_1, n_2 e n_4, mas não é amiga da pessoa n_3.
Assim, implemente uma **função** que recebe uma matriz de adjacência M e o número de pessoas n contidas nela e que retorna o 1 se há nessa rede social uma pessoa sem amigos e 0 caso contrário. Na matriz exemplo, a sua função deve retornar 0, pois todas as pessoas tem amigos. Além disso, considere que existe uma definição para o número máximo de pessoas que a matriz comporta, chamado `MAX_PESSOAS`. O protótipo dessa função deve ser:

```
int existeIsolados(int M[ ][MAX_PESSOAS], int n);
```

2. (*15 points*) Neste exercício, você deve criar um protótipo de um sistema de batalha entre guerreiros de um jogo. Para isso, implemente os itens a seguir.

a. (*2 pts*) Defina um novo tipo de dados chamado `Guerreiro` com os seguintes campos: `ataque` (inteiro), `defesa` (inteiro), `pontos_vida` (inteiro) e `id_jogador` (inteiro).

b. (*2 pts*) Escreva uma função de nome `rolaDados` que simula a rolagem de três dados de seis faces tradicionais (1 a 6) e retorna a soma dessas rolagens. Note que somar os valores resultantes da rolagem de três dados de seis faces é diferente de rolar um dado que retorna um número entre 3 e 18.

c. (*3 pts*) Escreva um procedimento de nome `criaGuerreiro` que recebe um `Guerreiro` por passagem de parâmetro por referência e que atribui valores aos seus campos de batalha. Cada um dos seus campos de batalha (`ataque`, `defesa` e `pontos_vida`) deve receber um valor inteiro da função `rolaDados`.

d. (*4 pts*) Escreva um procedimento de nome `ataca` que recebe dois `Guerreiro`s por passagem de parâmetro por referência e simula um ataque do primeiro guerreiro no segundo. O ataque é dado da seguinte maneira:

- O primeiro guerreiro rola três dados e soma os seus valores com o seu campo `ataque`. Essa soma é o valor do `golpe` do primeiro guerreiro.
- O segundo guerreiro rola três dados e soma os seus valores com o seu campo `defesa`. Essa soma é o valor do `escudo` do segundo guerreiro.
- Faça `dano` = `golpe` − `escudo`. Se o `dano` for maior que zero, reduza `dano` dos `pontos_vida` do segundo guerreiro.

e. (*4 pts*) Escreva um programa que simula a batalha até a morte entre dois guerreiros. Para isso, crie dois guerreiros, um com `id_jogador` 1 e outro com `id_jogador` 2. Depois, atribua valores aleatórios para os seus campos de batalha a partir da função `criaGuerreiro` e inicie ataques intercalados entre esses guerreiros, ou seja, comece com o guerreiro 1 atacando o 2, depois o 2 atacando o 1, depois o 1 atacando o 2 e assim por diante. Para simular um ataque, use a função `ataca`. A batalha deve acabar quando um dos jogadores, o perdedor, alcançar 0 ou menos `pontos_vida`. Imprima na tela o identificador do guerreiro vencedor.

3. (*6 points*) Escreva um procedimento RECURSIVO de nome `imprimeNaturais` para imprimir todos os números naturais de 0 até n em ordem crescente. O procedimento não deve fazer uso de estruturas de repetição (while, for etc) nem de variáveis globais.