universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Programação I

Folha Exercícios 11
Arrays bidimensionais, de strings e de registos

António J. R. Neves
João Rodrigues
Osvaldo Pacheco
Arnaldo Martins

2018/19

Folha exercícios 11 - Arrays bidimensionais, de strings e de registos

Resumo:

- arrays bidimensionais
- arrays de Strings
- arrays de registos (classes)

Vimos até aqui que podíamos armazenar numa variável do tipo array vários valores do mesmo tipo. Podemos, no entanto, também ter arrays de tipos referência, sendo que numa variável deste tipo podemos armazenar várias referências de um certo tipos de dados: arrays de Strings arrays de registos (classes) e arrays bidimensionais (arrays de arrays).

A criação deste tipo de arrays necessita de uma declaração semelhante à dos arrays de tipos primitivos, mas cada elemento do array necessita depois de ser "criado" segundo as regras correspondentes ao tipo de dados em causa. (No caso de arrays bidimensionais a criação pode ser simultânea.)

Problemas para resolver

Exercício 11.1

Escrever um programa que calcule a multiplicação de duas matrizes. O programa começa por pedir as dimensões e o conteúdo das duas matrizes, e depois mostra a matriz resultado. Deve garantir que as matrizes têm dimensões compatíveis.

> Em matemática, o produto de duas matrizes é definido somente quando o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz. Se *A* é uma matriz m×n (A também pode ser denotada por $A_{m,n}$) e *B* é uma matriz n×p, então seu **produto** é uma matriz m×p[1] definida como *AB* (ou por $A \cdot B$). O elemento de cada entrada c_{ij} da matriz AB (o qual denotaremos por $(AB)_{ij}$) é dado pelo produto da i-ésima linha de A com a j-ésima coluna de B[2], ou seja,
>
> $$(AB)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$
>
> para cada par *i* e *j* com $1 \le i \le m$ e $1 \le j \le p$.

Exercício 11.2

Tendo como base o programa desenvolvido no exercício 8.2, nomeadamente todas as funções implementadas, acrescente a funcionalidade de imprimir no final os pontos ordenados por ordem crescente da sua distância à origem. Para isso deverá ir armazenando num array todos os pontos introduzidos.

Assim sendo, deverá desenvolver um programa (função main) que peça ao utilizador pontos até à introdução do ponto (0, 0). O programa deve contar quantos pontos foram introduzidos e calcular a soma das suas distâncias à origem (ponto (0, 0)). O programa deverá também determinar qual o ponto mais afastado da origem. No final deverão ser impressos todos os pontos introduzidos ordenados por ordem crescente da sua distância à origem.

Exercício 11.3

Pretende-se escrever um programa que leia do teclado uma lista de frases, até um máximo de 10 frases ou até ser lida a frase "fim" e escreva no monitor as frases lidas pela ordem inversa com que foram introduzidas. Não só deve escrever primeiro a última frase introduzida, como a frase deve aparecer ao contrário. Exemplo de utilização do programa:

```
Frase 1: ola
Frase 2: aveiro
Frase 3: P1
Frase 4: fim

Resultado:
1P
orieva
alo
```

Exercício 11.4

O programa `ProcessImage.java` permite ler uma imagem, num ficheiro em formato Plain PGM, e produz uma nova imagem modificada, que grava no ficheiro out.pgm. Compile e experimente esse programa sobre uma das imagens fornecidas. Pode visualizar as imagens com o programa eog (Eye-of-gnome) ou outro (ex: IrfanView no Windows).

Crie uma função que rode a imagem de 90º no sentido dos ponteiros do relógio. Modifique o programa para fazer essa operação.

Note: O formato Plain PGM é pouco eficiente e por isso é raramente usado. Num sistema Linux pode usar o comando convert (do pacote ImageMagik) para converter entre formatos de imagem.

Exercício 11.5

Faça um programa para gerir uma agenda de contactos. A agenda aceita um máximo de 100 contactos e cada contacto deve ter a seguinte informação:

```
Nome - texto livre;
Morada - texto livre;
Telefone - número inteiro;
email - endereço de email.
```

O programa deve funcionar de forma repetitiva com base num menu de opções que a seguir se apresenta:

```
Gestão de uma agenda:
I - Inserir um contacto
P - Pesquisar contacto por nome
E - Eliminar um contacto
M - Mostrar os contactos
```

```
O - Mostrar contactos Ordenados pelo nome
V - Validar endereços de email
A - Apagar a agenda
T - Terminar o programa
Opção ->
```

O programa deverá permitir as seguintes operações:

1) Introdução de um novo contacto na agenda. Não é necessário fazer validações na introdução dos dados.
2) Mostrar informação sobre um contacto com base no nome, parcial ou completo, pedido ao utilizador. Deve informar o utilizador se o aluno não existir. Sugere-se a utilização da função **indexOf** da classe String.
3) Eliminar um contacto da agenda com base no número de telefone, pedido ao utilizador. Deve informar o utilizador se o contacto não existir.
4) Mostrar ao utilizador os contactos da agenda pela ordem com que foram introduzidos.
5) Mostrar ao utilizador os contactos ordenados pelo nome. Nesta operação deve ter o cuidado de manter a base de dados original, isto é, deve manter a ordem original de introdução.
6) Verificação dos endereços de email existentes na agenda e, em caso de erro, pedir ao utilizador novo email para o contacto em causa. Um email válido só pode conter caracteres alfanuméricos, um símbolo '@' e os símbolos '.' e '_'.
7) Apagar a agenda.

Exercício 11.6

Crie um programa que permita gerir a avaliação de uma turma de Programação I com um máximo 20 alunos. Para cada aluno deverá ser guardada a seguinte informação:

```
Número mecanográfico - valor inteiro;
Nome do aluno - texto livre;
Notas de 3 testes – valores inteiros no intervalo [0 ...
20];
Nota final - valor real no intervalo [0,0 … 20,0].
```

O programa deve funcionar de forma repetitiva com base num menu de opções que a seguir se apresenta:

```
Gestão de uma turma:
1 - Inserir informação da turma
2 - Mostrar informação de um aluno
3 - Alterar informação de um aluno
4 - Listar os alunos ordenados por nº mec.
5 - Listar os alunos ordenados por nota final
```

```
6 - Média das notas finais
7 - Melhor aluno
8 - Inserir pesos dos vários testes
0 - Terminar o programa
Opção?
```

O programa deverá permitir as seguintes operações:

1) Introdução da informação associada aos alunos, terminando com a introdução do nº mec. 0. Toda a informação deverá ser pedida ao utilizador com a exceção da nota final que deverá ser calculada pelo programa de acordo com os pesos dos vários testes (opção 8). Nesta opção, a turma deve ser preenchida desde o início, ignorando os dados previamente introduzidos.
2) Mostrar informação sobre um aluno com base no nº mec., pedido ao utilizador. Deve informar o utilizador se o aluno não existir.
3) Alterar notas de um aluno cujo nº mec. é pedido ao utilizador. Se o aluno não existir, deve ser acrescentado à turma no caso de ainda não se encontrar preenchida.
4) Mostrar a informação sobre os alunos, ordenada por nº mec.
5) Mostrar a informação sobre os alunos, ordenada por nota final.
6) Calcular e imprimir a média das notas finais da turma.
7) Mostrar ao utilizador a informação sobre o melhor aluno.
8) Introduzir os pesos dos vários testes.

Exercício 11.7

Modifique o exercício 11.6 para permitir gerir várias turmas simultaneamente (no máximo 10 turmas). O menu deve ser semelhante, mas cada opção deve começar por pedir a turma (um número entre 1 e 10). Por exemplo:

```
Gestão de turmas:
1 - Inserir informação numa turma
2 - Mostrar informação de um aluno
...
0 - Terminar o programa
Opção? 1
Turma? 8
Introduza informação para a turma 8:
NMec? ...
```

Sugestão: Criar um array de turmas, sendo cada turma um array de alunos (do exercício anterior). As funções originais, que operam sobre turmas individuais, não devem precisar de ser alteradas. Se foram bem feitas, deve bastar alterar os argumentos passados nas suas invocações. Mas se (ab)usou de variáveis “globais”, vai ser mais difícil!

Exercício 11.8

Crie um programa para gerir uma biblioteca. Assume-se que existe um máximo de 200 livros e que se pretende manter para cada um deles a seguinte informação:

```
Cota - código com 20 caracteres alfanuméricos;
Autor - texto com um máximo de 40 caracteres;
Título - texto com um máximo de 60 caracteres;
Data de aquisição - constituída por dia, mês e ano;
Estado do livro -  carácter (requisitado 'R', livre 'L').
```

O programa deve funcionar de forma repetitiva com base num menu de opções que a seguir se apresenta:

```
Gestão de uma biblioteca
1 - Introduzir livro
2 - Remover um livro
3 - Apagar a base de dados
4 - Verificação de cotas repetidas
5 - Alterar o estado de um livro
6 - Listar os livros requisitados
7 - Listar os livros ordenados pela cota
8 - Listar os livros ordenados pela data
9 - Terminar o programa
Opção ->
```

O programa deverá permitir as seguintes operações:

1) Introdução de um livro na biblioteca. São pedidos os dados do livro e acrescenta-se o registo à base de dados. Assuma que o utilizador introduz uma data válida e garanta apenas a validação dos restantes campos, segundo o enunciado.
2) Remover um livro com base na cota, pedida ao utilizador. Deve informar o utilizador se o livro não existir.
3) Apagar a base de dados.
4) Verificação das cotas existentes na base de dados e em caso de cotas repetidas pedir ao utilizador nova cota para o livro em causa. A ordem com que as cotas são analisadas não é relevante. No final desta ação devemos garantir que não temos dois livros com a mesma cota.
5) Para alterar o estado de um livro, começa-se por pedir a cota e, caso exista, em função do seu estado atual o livro pode ser libertado (se requisitado ou condicionado), requisitado (se livre) ou colocado em acesso condicionado (se livre).
6) Mostrar ao utilizador os livros requisitados.
7) Mostrar ao utilizador os livros ordenados por cota.
8) Mostrar ao utilizador os livros ordenados por data.