Ficha Prática n^{o}3

vectores e strings

Programação III

Paul Crocker e Simão Melo de Sousa
Departamento de Matemática/Informática da UBI

23 de Março de 2001

Exercício 1: (*Soma de Vectores*)
Escreva um programa, com as devidas funções, que permita inicializar dois vectores de inteiros e efectuar a soma - elemento por elemento.

Exercício 2: (*Operações Básicas em Vectores*)

1. Defina uma função (Quais são os seus parâmetros?) que encontra o maior e o menor elemento de um qualquer vector de inteiros.
2. Escrever um programa C que lê 10 números inteiros num vector e que depois procura o maior e o mais pequeno valor introduzido.
   (a) Usando o formalismo vector
   (b) Usando o formalismo apontador

Exercício 3: (*Procura do segundo maior valor*)
Escreva um programa que inicializa um vector de inteiros, e que permita mostrar o segundo maior valor do vector percorrendo uma só vez o vector. Qual é o número de comparações entre elementos do vector que efectua a função de procura;

Exercício 4: (*Merge de Vectores Ordenados*)
Escreva uma função que permita juntar dois vectores ordenados num terceiro vector. Defina também a versão recursiva desta função.

Exercício 5: (*Ordenação de vectores*)
Simula os algoritmos de ordenação seguintes com um exemplo da sua escolha e implementa-los em C:

1. **Bubble Sort**: O método do Bubble Sort pode ser resumido pelo algoritmo seguinte:

   Requisitos: tab: vector de inteiros preenchidos
   com N valores
   Para i **A Variar de** N **Até** segunda posição **Fazer**
     Para j **A Variar de** primeira posição **Até** i-1 **Fazer**
       Se tab[j] > tab[j+1]
         Então Trocá-los entre si.

2. **Quick Sort**. O método do Quick Sort pode ser resumido pelo algoritmo seguinte:

   Requisitos: tab: vector de inteiros, ini: índice da primeira posição
   "de trabalho"de tab, fim: índice da última posição de "trabalho"de tab
   Se ini $\leq$ fim
     Então
       Escolher um pivot p (valor presente no vector entre a posição
         ini e a posição fim).
       Particionar tab de tal forma que todos os elementos menores do que p
         estajam a esquerda de p e os maiores à direita.
         Seja i o índice de p nesta partição.
       QuickSort(tab,ini,i-1)
       QuickSort(tab, i+1,fim)

Exercício 6: (*Produto de Matrizes*)
Escreva um programa, com as devidas funções, que permita inicializar duas matrizes A, B de inteiros de respectivo tamanho (m, p) e (p, n), dimensão máxima MAXN e efectuar o produto numa matriz C (de tamanho (m, n)). Relembramos que $C[i, j] = \sum_{k=0}^{p} A[i, k]B[k, j]$.

Exercício 7: (*A Sorte Grande*)
Escreva um programa que permita simular o jogo do totoloto usando funções, com matrizes e/ou vectores. Pretende-se que o programa seja o mais realista possível (e não o mais eficiente possível) pedindo ao utilizador uma aposta simples (pede 6 números) e arquivando-a numa grelha de 49 números, pretende-se também que o programa peça os 6 números, mais o complementar, que foram sorteados e que devolva então a quantidade de números certos da aposta do utilizador.

Exercício 8: (*Procura de motivo*)
Escreva uma função C seguinte:

```
int procura(char s[], char t[]);
```

que permita devolver a posição da primeira occorência de `t` em `s`. A função devolve -1 se `t` não occorer em `s`.

Exercício 9: (*Peso duma palavra*)
Define-se inductivamente sobre o comprimento duma palavra a noção de peso começando pela noção de peso de caracteres:

$$\begin{array}{lclcl} \text{peso('a')} & = & \text{peso('A')} & = & 1 \\ \text{peso('b')} & = & \text{peso('B')} & = & 2 \\ & & \vdots & & \\ \text{peso('z')} & = & \text{peso('Z')} & = & 26 \end{array}$$

$$\text{e } \forall\, x \notin \{'a', \ldots, 'z', 'A', \ldots, 'Z'\} \text{ peso}(x)\text{=}0$$

Seja c um caracter, S uma palavra e S' a palavra resultante da concatenação de c e de S. Representamos por a palavra vazia. Assim sendo, o peso duma palavra é definida por:

$$\begin{array}{lcl} \text{PESO()} & = & 0 \\ \text{PESO(S')} & = & \text{peso(c)+PESO(S)} \end{array}$$

Resolva de forma recursiva e iterativa as alíneas seguintes:

1. Escreva uma função que devolve o peso duma string.
2. Escreva uma função que compara duas palavras pelo respectivo peso.

Exercício 10: (*Strings*)
Escreva as funções seguintes:

1. `int lers(char s[], int lim);` que lê uma string `s` de tamanho máximo `lim`;
2. `int strlen(char s[]);` que devolve o comprimento duma string `s`
3. `void strcpy(char s[], char t[]);` que copia `t` em `s`
4. `void strcat(char s[], char t[]);` que concatena `t` ao fim de `s`
5. `int strcmp(char s[], char t[]);` que devolva 0 se `t=s` , -1 se `s>t` e 1 se `t>s`.
6. `void tolower(char s[]);` que converte todas as maiúsculas de `s` em minúsculas.
7. `void remove(char c, char s[]);` que remove o caracter `c` de `s`

Exercício 11: (*Aritmetica sobre inteiros vistos como Vectores*)
Propomos neste exercício que defina os tipos e escreva as funções C que permitam trabalhar com os inteiros vistos como vectores. Por exemplo 12243 é visto como um vector de inteiro em que cada célula contem um algarismo do número 12243. Sugere-se que guarde na posição 0 desse vector o comprimento (o número de algarismos) do numero arquivado (no nosso exemplo 5 estará arquivado na posição 0 e representa o comprimento de 12243).

1. Defina o tipo dos inteiros como vectores `vinteiro`
2. Escreva uma função que permita a introdução de um valor inteiro para um desses vectores
   ```
   void intro(vinteiro num);
   ```
3. Escreva a função `void soma(vinteiro a, vinteiro b, vinteiro c);` que permite somar em `c` as duas variáveis de tipo `vinteiro`, `a` e `b`.

Exercício 12: (*Capicuas por Inversão*)
É possivel criar números capicuas com o chamado algoritmo de "inversão"que passamos a descrever:

1. O método começa por considerar um número inicial que podemos chamar "semente";
2. Baseando-se neste número é criado o seu inverso, invertendo simplesmente a ordem dos algarismos que consituem o número;
3. soma-se o número e o inverso;
4. Repete-se o processo tomando como nova semente o resultado da soma, se esse não for uma capicua e que não seja demasiado grande.

Exemplo para o número inicial 87
$87 + 78 = 165$
$165 + 561 = 726$
$726 + 627 = 1353$
$1343 + 3531 = 4884$: que é capicua

Devido ao facto de alguns números iniciais resultarem em números muito superiores ao `MAXINT` da máquina e também de não podermos usar o tipo float e arriscar arredondamentos, temos que representar um número inteiro por um vector de dígitos 0-9. Um bom limite inicial para o tamanho de tal vector pode ser 51.

Nota: utilizar : `typdef unsigned short vinteiro[MAX]`, para poupar espaço. Assim, podemos utilizar e incluir funções criadas no Exercício anterior, devidamente alteradas para usar o typdef acima referido, para, por exemplo, fazer a soma de dois vectores de dígitos.

Escreva um programa que, baseado neste processo, lista todas as capicuas relacionadas com os números iniciais de três digitos: 100-999. O programa deverá fazer o seguinte:

1. Para o número inicial 876 mande para o Output o número capicua resultante deste algoritmo de inversão;

2. Guarde numa Matriz todos os números iniciais que precisam pelo menos de 6 operações de inversão para dar uma capicua, para cada um deles o número de operações utilizadas e o número de dígitos da capicua;

3. Guarde num vector todos os números iniciais que não dão um número capicua no limite especificado;

4. No fim do programa mande para o Output os resultados encontrados nas partes 2 e 3

Experimente alterando o tamanho máximo de 51 para milhares! Corra o programa e redireccione o output para um ficheiro para que esse possa ser examinado à vontade por si e entregue juntamente com o seu programa devidamente anotado com comentários.
Acredita-se (embora não esteja provado) que todos os números darão um número capicua com este processo. Todos os números $<$ 10000 já deram um número capicua, alguns com milhões de dígitos, com a excepção do número 196!

Exercício 13: (*Xadrez*)
No jogo de xadrez, representa-se cada posição pelas suas cordenadas (i, j), a célula mais à esquerda e em baixo terá as coordenadas $(0, 0)$. Num tal tabuleiro, numa só deslocação, o cavaleiro pode-se deslocar da posição (i, j) para 8 posições possíveis(se nenhuma delas sair fora do tabuleiro, claro... isto é se as duas cordenadas resultantes forem maiores ou iguais a 0 e menores ou iguais a 7): **1:** $(i-2, j+1)$, **2:** $(i-1, j+2)$, **3:** $(i+1, j+2)$, **4:** $(i+2, j+1)$, **5:** $(i+2, j-1)$, **6:** $(i+1, j-2)$, **7:** $(i-1, j-2)$ e **8:** $(i-2, j-1)$.

1. Escreva um programa que calcula as posições de todas as células do tabuleiro accessíveis pelo cavaleiro em p lances a partir de uma dada célula (i_0, j_0).

   Para esse efeito sugerimos que considera os tipos, as variáveis e as funções seguintes:

```
/* nmax representa o numero maximo de jogadas que nos autorizamos
fazer para atingir todas as celulas do tabuleiro digamos que nmax
=70 (de facto nmax pode ser menor...) */ #define nmax 70

/* ncel representa o numero de celulas do tabuleiro ou seja 8*8 */
#define ncel 64

/*uma celula e  constituida por duas cordenadas, seja celula o
tipo destas cordenadas*/ typedef int celula[2];

/*jogadas[i] representa o conjunto das celulas atingidas em i+1
jogadas e jogadas[i][j] representa a j-esima dessas celulas. Uma
restricao para satisfazer e i+1 ser o menor numero de jogadas
possivel para atingir as celulas arquivadas na linha i*/ celula
jogadas[nmax][ncel];

/*para cada numero de jogada i(+1) e arquivada o numero de celula
atingidas, ncelula[i] corresponde ao cardinal da linha jogadas[i]
```



```
(numero de celulas ai arquivadas)*/ int ncelula[nmax];

/*esta funcao retorna o facto da celula proposta em parametro ser
uma celula valida ou nao. devolve 1 se o for, 0 senao */ int
ok(celula c) {
 if ((c[0]>= 0) && (c[0] <=7) && (c[1]>= 0) && (c[1] <=7))
      return 1;
 else return 0;
}
```

   relembramos que qualquer que seja o número de lances nunca se conseguirá atingir mais do que `ncel` células. `jogadas[i][j]` representa a j-ésima das células atingíveis em exactamente i+1 jogadas[1] e `ncelula[i]` representa o número de celulas que se pode atingir em i+1 jogadas(e que não se pode atingir em menos jogadas).

   Sugere-se que escreva as funções seguintes:

```
/* acessivel e a funcao principal de resolucao do problema, o seu
papel e essencialmente permitir o calculo da lista de todas
celulas acessiveis em menos de njogadas (incluido) a partir da
celula (i0,j0) */ void acessivel(int i0,int j0,int njogadas);
```

   que utiliza

```
/* esta funcao devolve a posicao cseg que corresponde ao movimento
k (das 8 possiveis) do cavaleiro calculada a partir da celula c */
void celula_seguinte(celula c, int k, celula cseg);

/* esta funcao devolve verdade se a celula c ja foi atingida por
uma jogada inferior a pcorente, falso senao */ int ja_existe(
celula c, int pcorente);
```

2. Quais modificações sugere para que o programa anterior (e as suas funções) indique se todas as celulas do tabuleiro podem ser atingidas a partir duma dada posição (i_0, j_0).

---

[1] Um vector em C começa na posição 0, ora as jogadas começam em 1, por isso a posição 0 refere a jogada 1 e assim successivamente...