Paradigmas de Programação II
Departamento de Informática
Universidade do Minho

LESI 1º Ano MCC

2002/2003

Ficha Teórico-Prática Nº 2
Programação em C - Representação e Tratamento de Strings

10 de Março de 2003

1 Objectivos

Esta ficha prática aborda o problema da leitura, representação e manipulação de strings em C.

Os exercícios 8 e 10 não serão discutidos formalmente nas aulas teórico-práticas. Estes terão que ser resolvidos pelos alunos e integrarão o relatório do primeiro trabalho prático da disciplina.

2 Instalação do módulo LaTeX listings

O package listings está disponível em `http://www.atscire.de/products/listings/`

Com o package, vem um documento `README`, com um guia de instalação rápida.

3 Representação de strings em C

Na linguagem C, as strings são armazenadas em células de memória contíguas. Para criar o espaço na memória necessário, declara-se um array de células do tipo char. Exemplo:

```
char nome[10];
```

Esta declação reserva 10 células de memória. As células são identificadas como sendo nome[0], nome[1] até nome[9].

Como o tamanho das strings não é exactamente igual ao espaço de armazenamento, usa-se o character cujo código é 0 para marcar o final da string. Assim, se a string nome tiver o conteúdo Matilde, a memória é ocupada da seguinte forma:

M	a	t	i	l	d	e	\0	?	?
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Estão disponíveis diversas funções para manipular strings em C, declaradas em #include <strings.h>. Por exemplo, a invocação n = strlen(nome); atribui a n o valor 7.

3.1 Exercícios

Exercício 1 (Funções de leitura) *Indique as diferenças entre as seguintes funções:*

int getchar(void);

*char *fgets(char *s, int size, FILE *stream);*

*int scanf(const char *format , ...);*

Diga qual o valor retornado pela função scanf.

Exercício 2 *Indique a diferença entre as funções seguintes, escrevendo um pequeno programa que mostra um menu com três opções: uma para ler um número, outra para ler um nome e outra para sair.*

int getchar(void);

int getch(void);

Exercício 3 (Testar se é capicua) *Leia uma string e verifique se a mesma é ou não capicua.*
Capicua brasileira: Socorram-me subi no onibus em marrocos.

Exercício 4 (Conversão de string para inteiro) *Leia uma string e verifique se a mesma representa um número inteiro válido (eventualmente com um sinal no início). Depois escreva uma função com a mesma funcionalidade de int atoi(const char *nptr);, para calcular o valor do inteiro introduzido.*
Nota: Estude a função scanf e a conversão para inteiro com %d.

Exercício 5 *Leia um nome e depois normalize-o segundo as seguintes regras (use as funções tolower/toupper, em #include <ctype.h>):*

- *Todos os nomes começam por maiiúscula, seguindos de letras minúsculas.*
- *Entre os nomes só deve constar um espaço.*
- *No início e no fim do nome, não devem haver espaços.*

Exercício 6 *Leia um nome e depois normalize-o segundo as seguintes regras:*

- *O nome normalizado só deve ter o apelido, seguido de vírgula, seguido do nome próprio.*

Exercício 7 *Leia um nome e depois normalize-o segundo as seguintes regras:*

- *O nome normalizado deve conter o nome próprio, seguido das iniciais dos nomes do meio, seguidos de um ponto, seguido do apelido.*

Exercício 8 (Normalização de datas) *As datas assumem muitas formas na escrita corrente.*
Escreva um programa que aceite um formato e uma data, dê como resultado uma data normalizada, com o formato: YYYY−MM−DD (se a data não for válida de acordo com o formato, deverá dar uma indicação de erro).
A formato da data é uma string qualquer, em que a letra D representa um digito do dia, a letra M um digito do mês e a letra Y representa um digito do ano. Exemplos de formatos válidos:

MM/DD/YY
DD de MM de YYYY
YYYY, DD de MM

Exercício 9 (Cifra por substituição arbitrária) *Considere-se uma cifra (palavra chave) constituída pelas letras "S A P O". Para a cifragem (codificação) da mensagem efectuam-se os seguintes passos:*

1. *retiram-se espaços e caracteres de pontuação;*
2. *convertem-se todas as letras em maiúsculas ou em minúsculas;*
3. *considera-se a seguinte tabela de equivalência:*

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
S	A	P	O	B	C	D	E	F	G	H	I	J	K	L	M	N	Q	R	T	...

   A construcção desta tabela limita-se a colocar a palavra chave no início e completar com o alfabeto, não colocando as letras repetidas.

4. *para a codificação, substitui-se cada um dos caracteres da primeira linha pelo respectivo caracter da seguinda linha.*

Construa um programa que dada uma cifra, codifique a mensagem e um outro que, dada a mensagem e a cifra descodifique a mensagem (claro que neste processo perdemos os espaços e os caracteres de pontuação!).

Exercício 10 (Cifra de Vigenère) *A cifra de César consiste no processo de adicionar a cada caracter um valor entre 0 e 25, ou seja,* $C(x) = (x + n) \bmod 26 \quad 0 \leq n \leq 26$. *Dado que esta codificação é facilmente quebrada, Blaise Vigenère aplicou uma pequena variante que se manteve inquebrável até meados do século XIX.*

O processo de codificação de Vigenère também se baseia numa cifra (palavra chave). Cada uma das letras desta cifra corresponde a um valor inteiro (A=0, B=1, ... Z=25). O processo de codificação consiste nos seguintes passos:

1. *retiram-se espaços e caracteres de pontuação;*
2. *convertem-se todas as letras em maiúsculas ou em minúsculas;*
3. *divide-se a mensagem em blocos do tamanho da cifra e, para cada um destes blocos, aplica-se a cifra de César a cada um dos caracteres, usando para valor n a correspondente letra;*

Considere a palavra chave "A C B" e a mensagem a codificar "sapato". Dividimos "sapato" em grupos de três: "sap" e "ato". Para cada uma das letras de "sap" vamos aplicar a cifra de César ("sap" + "ACB" = "(s+A)(a+C)(p+B)" = "(s+0)(a+2)(p+1)" = "scq"). Aplicando o mesmo algoritmo a "ato" obtemos "avp". Então, a palavra codificada será "scqavp".

Construa um programa que dada uma cifra, codifique a mensagem e um outro que, dada a mensagem e a cifra descodifique a mensagem (claro que neste processo perdemos os espaços e os caracteres de pontuação!).