Cadeias de Caracteres (*Strings*)

- Nesta aula veremos a estrutura de dados homogênea **cadeia de caracteres**
- Essa estrutura pode parecer similar a um vetor de caracteres mas veremos que ela possui um tratamento diferenciado

José Augusto Baranauskas
Departamento de Física e Matemática – FFCLRP-USP
Sala 228 – Bloco P2 – Fone (16) 602-4439
E-mail: augusto@ffclrp.usp.br
URL: http://fmrp.usp.br/augusto

Caractere

- Um **caractere** é considerado um tipo de dado primitivo na maioria dos computadores
- Um tipo de dado é primitivo se o computador possui instruções em linguagem de máquina que permitem a manipulação deste tipo
- Desde que uma cadeia é uma seqüência ordenada de caracteres, o caractere é a entidade fundamental de manipulação de uma cadeia

Caractere

- Um caractere pertence a um conjunto finito de caracteres: um alfabeto
- Um exemplo de alfabeto é o conjunto de letras utilizado na língua inglesa
- Outro exemplo de alfabeto é o conjunto de letras utilizado na língua portuguesa
- Outro alfabeto comum é o conjunto de dígitos decimais
- Através dos anos, muitos alfabetos (conjunto de caracteres) para utilização em computadores foram desenvolvidos

Codificação de Caracteres

- Um caractere pode ser representado na memória como uma seqüência de bits (uma seqüência de zeros e uns), sendo que a cada caractere do conjunto é atribuída uma seqüência distinta de bits, segundo uma convenção escolhida
- Geralmente, adota-se uma codificação do conjunto de caracteres em seqüência de bits de comprimento fixo
  - Cada caractere do conjunto é representado pelo mesmo número de bits
- Uma seqüência de bits de tamanho **n** consegue representar 2^n caracteres
- Por exemplo, com n=7 podemos codificar até 2^7=128; com n=8, podemos representar até 2^8=256

Codificação de Caracteres

- Entre os vários métodos de codificação, os mais populares são
  - código ASCII (7 bits) - *American Standard Code for Information Interchange*
  - código EBCDIC (8 bits) - *Extended Binary Coded Decimal Interchange Code*
  - código UNICODE (8 bits)

Codificação ASCII (7 bits)

- Cada byte armazena um caractere: algarismo, letra, símbolo ou caractere de controle
- Possibilidade de 2^7 representações diversas (128 caracteres)
  - alfabeto inglês em letras minúsculas e maiúsculas (52)
  - caracteres decimais numéricos (10)
  - caracteres especiais e de operação (33)
  - caracteres de controle (33)

ASCII & EBCDIC

dec	hex	binary	ASCII		EBCDIC	
0	0x00	0000 0000	NUL	Null	NUL	Null
1	0x01	0000 0001	SOH	Start of Heading (CC)	SOH	Start of Heading
2	0x02	0000 0010	STX	Start of Text (CC)	STX	Start of Text
3	0x03	0000 0011	ETX	End of Text (CC)	ETX	End of Text
4	0x04	0000 0100	EOT	End of Transmission (CC)	PF	Punch Off
5	0x05	0000 0101	ENQ	Enquiry (CC)	HT	Horizontal Tab
6	0x06	0000 0110	ACK	Acknowledge (CC)	LC	Lower Case
7	0x07	0000 0111	BEL	Bell	DEL	Delete
8	0x08	0000 1000	BS	Backspace (FE)		
9	0x09	0000 1001	HT	Horizontal Tabulation (FE)		
10	0x0A	0000 1010	LF	Line Feed (FE)	SMM	Start of Manual Message
11	0x0B	0000 1011	VT	Vertical Tabulation (FE)	VT	Vertical Tab
12	0x0C	0000 1100	FF	Form Feed (FE)	FF	Form Feed
13	0x0D	0000 1101	CR	Carriage Return (FE)	CR	Carriage Return
14	0x0E	0000 1110	SO	Shift Out	SO	Shift Out
15	0x0F	0000 1111	SI	Shift In	SI	Shift In
16	0x10	0001 0000	DLE	Data Link Escape (CC)	DLE	Data Link Escape
17	0x11	0001 0001	DC1	Device Control 1	DC1	Device Control 1
18	0x12	0001 0010	DC2	Device Control 2	DC2	Device Control 2
19	0x13	0001 0011	DC3	Device Control 3	TM	Tape Mark
20	0x14	0001 0100	DC4	Device Control 4	RES	Restore
21	0x15	0001 0101	NAK	Negative Acknowledge (CC)	NL	New Line
22	0x16	0001 0110	SYN	Synchronous Idle (CC)	BS	Backspace
23	0x17	0001 0111	ETB	End of Transmission Block (CC)	IL	Idle
24	0x18	0001 1000	CAN	Cancel	CAN	Cancel
25	0x19	0001 1001	EM	End of Medium	EM	End of Medium
26	0x1A	0001 1010	SUB	Substitute	CC	Cursor Control
27	0x1B	0001 1011	ESC	Escape	CU1	Customer Use 1

ASCII & EBCDIC

dec	hex	binary	ASCII		EBCDIC	
40	0x28	0010 1000	(	Left Parenthesis		
41	0x29	0010 1001	)	Right Parenthesis		
42	0x2A	0010 1010	*	Asterisk, "star"	SM	Set Mode
43	0x2B	0010 1011	+	Plus Sign	CU2	Customer Use 2
44	0x2C	0010 1100	,	Comma		
45	0x2D	0010 1101	-	Hyphen, Minus Sign	ENQ	Enquiry
46	0x2E	0010 1110	.	Period, Decimal Point, "dot"	ACK	Acknowledge
47	0x2F	0010 1111	/	Slash, Virgule	BEL	Bell
48	0x30	0011 0000	0	0		
49	0x31	0011 0001	1	1		
50	0x32	0011 0010	2	2	SYN	Synchronous Idle
51	0x33	0011 0011	3	3		
52	0x34	0011 0100	4	4	PN	Punch On
53	0x35	0011 0101	5	5	RS	Reader Stop
54	0x36	0011 0110	6	6	UC	Upper Case
55	0x37	0011 0111	7	7	EOT	End of Transmission
56	0x38	0011 1000	8	8		
57	0x39	0011 1001	9	9		
58	0x3A	0011 1010	:	Colon		
59	0x3B	0011 1011	;	Semicolon	CU3	Customer Use 3
60	0x3C	0011 1100	<	Less-than Sign	DC4	Device Control 4
61	0x3D	0011 1101	=	Equal Sign	NAK	Negative Acknowledge
62	0x3E	0011 1110	>	Greater-than Sign		
63	0x3F	0011 1111	?	Question Mark	SUB	Substitute
64	0x40	0100 0000	@	At Sign	SP	Space
65	0x41	0100 0001	A	A		
66	0x42	0100 0010	B	B		
67	0x43	0100 0011	C	C		

ASCII & EBCDIC

dec	hex	binary	ASCII		EBCDIC	
80	0x50	0101 0000	P	P	&	Ampersand
81	0x51	0101 0001	Q	Q		
82	0x52	0101 0010	R	R		
83	0x53	0101 0011	S	S		
84	0x54	0101 0100	T	T		
85	0x55	0101 0101	U	U		
86	0x56	0101 0110	V	V		
87	0x57	0101 0111	W	W		
88	0x58	0101 1000	X	X		
89	0x59	0101 1001	Y	Y		
90	0x5A	0101 1010	Z	Z	!	Exclamation Point
91	0x5B	0101 1011	[	Opening Bracket	$	Dollar Sign
92	0x5C	0101 1100	\	Reverse Slant	*	Asterisk
93	0x5D	0101 1101	]	Closing Bracket	)	Right Parenthesis
94	0x5E	0101 1110	^	Circumflex, Caret	;	Semicolon
95	0x5F	0101 1111	_	Underline, Underscore		Logical NOT
96	0x60	0110 0000	`	Grave Accent	-	Hyphen, Minus Sign
97	0x61	0110 0001	a	a	/	Slash, Virgule
98	0x62	0110 0010	b	b		
99	0x63	0110 0011	c	c		
100	0x64	0110 0100	d	d		
101	0x65	0110 0101	e	e		
102	0x66	0110 0110	f	f		
103	0x67	0110 0111	g	g		
104	0x68	0110 1000	h	h		
105	0x69	0110 1001	i	i		
106	0x6A	0110 1010	j	j		Logical OR
107	0x6B	0110 1011	k	k	,	Comma

ASCII & EBCDIC

dec	hex	binary	ASCII		EBCDIC	
120	0x78	0111 1000	x	x		
121	0x79	0111 1001	y	y		
122	0x7A	0111 1010	z	z	:	Colon
123	0x7B	0111 1011	{	Opening Brace	#	Number Sign, Octothorp, "pound"
124	0x7C	0111 1100	|	Vertical Line	@	At Sign
125	0x7D	0111 1101	}	Closing Brace	'	Apostrophe, Prime
126	0x7E	0111 1110	~	Tilde	=	Equal Sign
127	0x7F	0111 1111	DEL	Delete	"	Quotation Mark
128	0x80	1000 0000		Reserved		
129	0x81	1000 0001		Reserved	a	a
130	0x82	1000 0010		Reserved	b	b
131	0x83	1000 0011		Reserved	c	c
132	0x84	1000 0100	IND	Index (FE)	d	d
133	0x85	1000 0101	NEL	Next Line (FE)	e	e
134	0x86	1000 0110	SSA	Start of Selected Area	f	f
135	0x87	1000 0111	ESA	End of Selected Area	g	g
136	0x88	1000 1000	HTS	Horizontal Tabulation Set (FE)	h	h
137	0x89	1000 1001	HTJ	Horizontal Tabulation with Justification (FE)	i	i
138	0x8A	1000 1010	VTS	Vertical Tabulation Set (FE)		
139	0x8B	1000 1011	PLD	Partial Line Down (FE)		
140	0x8C	1000 1100	PLU	Partial Line Up (FE)		
141	0x8D	1000 1101	RI	Reverse Index (FE)		
142	0x8E	1000 1110	SS2	Single Shift Two (1)		
143	0x8F	1000 1111	SS3	Single Shift Three (1)		
144	0x90	1001 0000	DCS	Device Control String (2)		
145	0x91	1001 0001	PU1	Private Use One	j	j
146	0x92	1001 0010	PU2	Private Use Two	k	k
147	0x93	1001 0011	STS	Set Transmit State	l	l

ASCII & EBCDIC

dec	hex	binary	ASCII		EBCDIC	
160	0xA0	1010 0000				
161	0xA1	1010 0001	~	Tilde		
162	0xA2	1010 0010	s	s		
163	0xA3	1010 0011	t	t		
164	0xA4	1010 0100	u	u		
165	0xA5	1010 0101	v	v		
166	0xA6	1010 0110	w	w		
167	0xA7	1010 0111	x	x		
168	0xA8	1010 1000	y	y		
169	0xA9	1010 1001	z	z		
170	0xAA	1010 1010				
171	0xAB	1010 1011				
172	0xAC	1010 1100				
173	0xAD	1010 1101				
174	0xAE	1010 1110				
175	0xAF	1010 1111				
176	0xB0	1011 0000				
177	0xB1	1011 0001				
178	0xB2	1011 0010				
179	0xB3	1011 0011				
180	0xB4	1011 0100				
181	0xB5	1011 0101				
182	0xB6	1011 0110				
183	0xB7	1011 0111				
184	0xB8	1011 1000				
185	0xB9	1011 1001	`	Grave Accent		
186	0xBA	1011 1010				
187	0xBB	1011 1011				

ASCII & EBCDIC

dec	hex	binary	ASCII		EBCDIC	
200	0xC8	1100 1000			H	H
201	0xC9	1100 1001			I	I
202	0xCA	1100 1010				
203	0xCB	1100 1011				
204	0xCC	1100 1100				
205	0xCD	1100 1101				
206	0xCE	1100 1110				
207	0xCF	1100 1111				
208	0xD0	1101 0000			}	Closing Brace
209	0xD1	1101 0001			J	J
210	0xD2	1101 0010			K	K
211	0xD3	1101 0011			L	L
212	0xD4	1101 0100			M	M
213	0xD5	1101 0101			N	N
214	0xD6	1101 0110			O	O
215	0xD7	1101 0111			P	P
216	0xD8	1101 1000			Q	Q
217	0xD9	1101 1001			R	R
218	0xDA	1101 1010				
219	0xDB	1101 1011				
220	0xDC	1101 1100				
221	0xDD	1101 1101				
222	0xDE	1101 1110				
223	0xDF	1101 1111				
224	0xE0	1110 0000			\	Reverse Slant
225	0xE1	1110 0001				
226	0xE2	1110 0010			S	S
227	0xE3	1110 0011			T	T

ASCII & EBCDIC

dec	hex	binary	ASCII	EBCDIC	
240	0xF0	1111 0000		0	0
241	0xF1	1111 0001		1	1
242	0xF2	1111 0010		2	2
243	0xF3	1111 0011		3	3
244	0xF4	1111 0100		4	4
245	0xF5	1111 0101		5	5
246	0xF6	1111 0110		6	6
247	0xF7	1111 0111		7	7
248	0xF8	1111 1000		8	8
249	0xF9	1111 1001		9	9
250	0xFA	1111 1010			
251	0xFB	1111 1011			
252	0xFC	1111 1100			
253	0xFD	1111 1101			
254	0xFE	1111 1110			
255	0xFF	1111 1111			

Codificação de Caracteres

- Vamos analisar a representação de caracteres utilizando a codificação ASCII
- A representação do caractere 'a' é a seqüência de bits: 0110 0001

0	1	1	0	0	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

onde o peso de cada dígito binário é dado abaixo daquele dígito

- O valor decimal equivalente da representação para o caractere 'a' é

$$0\times2^7+1\times2^6+1\times2^5+0\times2^4+0\times2^3+0\times2^2+0\times2^1+1\times2^0$$
$$=64+32+1=97$$

Codificação de Caracteres

- Vimos que a representação decimal do caractere 'a' é 97
  - De forma similar, podemos determinar que o valor associado ao caractere 'b' é 98, 'b' é 99, ..., 'z' é 122
- Assim, para efeito de comparação temos
  - 'a' < 'b'< ...< 'y' < 'z'
- Note que a representação decimal de
  - 'A' é 65, 'B' é 66, ..., 'Z' é 90
  - '0' é 48, '1' é 49, ... '9' é 57
- Assim, para efeito de comparação temos
  - 'A' < 'B' < ... 'Y' < 'Z'
  - '0' < '1' < ...< '8' < '9'
- Observe também que
  - '0' < '1' < ...< '8' < '9' < 'A' < 'B' < ... 'Y' < 'Z' < 'a' < 'b'< ...< 'y' < 'z'
- Note também que o caractere branco (espaço) tem valor 32 e é menor do que qualquer letra (maiúscula ou minúscula) ou dígito decimal

Caracteres em C++

- Constantes do tipo caractere são escritas entre apóstrofos
  - 'a', 'b', ...
  - 'A', 'B', ...
- Alguns caracteres podem ser representados por uma seqüência de escape, além da sua representação decimal:
  - '\0' (caractere nulo)
  - '\n' (newline)
  - '\t' (tabulação)
  - '\a' (alert = BEL)
  - '\'' (apóstrofo)
  - '\"' (aspas)
  - '**ooo**' (caractere correspondente ao número octal **ooo**)
  - '\x**hhh**' (caractere correspondente ao número hexadecimal **hhh**)

Caracteres em C++

- Variáveis do tipo caractere são declaradas como **char**
- `char c1, c2;`
  - Declara duas variáveis **c1** e **c2** do tipo caractere
- `char x='a', y='\n';`
  - Declara duas variáveis **x** e **y** do tipo caractere cujos valores iniciais são o caractere 'a' e o caractere de nova linha, respectivamente
- `char apostrofo='\'';`
  - Declara variável contendo um apóstrofo
- `char w='\x61';`
  - Declara variável **w** contendo caractere 'a' ($61_{16}=97_{10}$)

Caracteres em C++

- `#include <cctype>`
- Por serem originalmente funções C, **bool** não é utilizado. Ao invés, zero é **false** e não-zero é **true**

Função	Descrição
isalnum(**c**)	true se **c** é uma letra ou dígito
isalpha(**c**)	true se **c** é uma letra
isblank(**c**)	true se **c** é espaço ou tabulação
isdigit(**c**)	true se **c** é um dígito
islower(**c**)	true se **c** é uma letra minúscula
isupper(**c**)	true se **c** é uma letra maiúscula
isspace(**c**)	true se **c** é um caractere (espaço, tab, formfeed, carriage return ou newline)
tolower(**c**)	retorna a versão minúscula do caractere **c**, se houver uma; caso contrário retorna o caractere **c** sem alteração
toupper(**c**)	retorna a versão maiúscula do caractere **c**, se houver uma; caso contrário retorna o caractere **c** sem alteração

Conceitos e Terminologia de Cadeias

- Uma cadeia de caracteres é uma seqüência ordenada de caracteres, cada um dos quais pertence a um alfabeto
  - Por exemplo, usando o alfabeto {'X','Y','Z'} podemos construir cadeias tais como "X", "XY", "XXYYZ" e "ZYX"
- Uma cadeia pode não conter caracteres
  - É indicada por ""
  - Denominada **cadeia vazia**
- Utilizaremos o símbolo □ para indicar o caractere espaço quando não existir outra forma clara de mostrar que ele faz parte da cadeia
- Note que "□" contém o caractere branco e não deve ser confundida com a cadeia vazia

Concatenação

- A operação de concatenação une duas cadeias de caracteres formando uma só
- Geralmente é indicada pelo operador **+**
- Por exemplo, se temos duas cadeias "Mona" e "lisa", o resultado da operação "Mona"+"lisa" é a nova cadeia "Monalisa"
- A cadeia vazia é o elemento neutro da concatenação
  - x+" " = " "+x = x
- Concatenação é associativa
  - (x+y)+z = x+(y+z) = x+y+z

Atribuição

- Se cidade é uma variável tipo cadeia, então o comando
  - cidade ← "Brasil" + "ia"
- atribuirá "Brasilia" à variável cidade
- Qualquer variável tipo cadeia pode assumir um valor tipo cadeia cujo comprimento é um número finito
- O comprimento de uma cadeia é o número de caracteres dessa cadeia
- Algumas linguagens de programação requerem que um comprimento máximo seja especificado para cada cadeia declarada
- Em nossa notação algorítmica não utilizaremos esta limitação
- Uma expressão com cadeias pode conter variáveis tipo cadeia assim como constantes tipo cadeia
  - a ← "Ciencia"
  - b ← "Computacao"
  - c ← a + "□da□" + b
  - As variáveis a,b e c são do tipo cadeia e o valor resultante c é "Ciencia da Computacao"

Comparação

- Dadas duas cadeias $x=x_1x_2...x_n$ e $y=y_1y_2...y_m$
- A condição x = y é considerada verdadeira se
  - x e y têm o mesmo número de caracteres (n=m)
  - $x_i=y_i$ para todo 1 <= i <= n
  - A condição "Maria" = "Maria" é verdadeira enquanto "Brilha"= "Brilhar" é falsa
- A relação de desigualdade é a negação da igualdade
  - As condições "Maria□" ≠ "Maria" e "Brilha" ≠ "Brilhar" são ambas verdadeiras

Comparação

- É fácil estender a comparação para incluir outros operadores relacionais tais como <, <=, > e >=
- O significado dessas comparações está baseado na codificação binária adotada
- A ordem é similar àquela encontrada em um dicionário ou lista telefônica
  - "brilha"<"brilhar", "ana"<"joana", "joaquim">= "bonifacio" são condições verdadeiras
  - "trem"<"blusa", "computador"<"ciencia", "bonifacio"<="alan" são condições falsas
- A condição é testada executando uma comparação seqüencial de caracteres da esquerda para a direita
- Note que a presença de qualquer caractere (mesmo um espaço) é sempre considerada maior do que a omissão de um caractere
  - "ciencia□" > "ciencia" é uma condição verdadeira

Outras Operações

- Existem outras operações básicas, além da concatenação, atribuição e comparação
- Geralmente, as linguagens de programação incluem operações para
  - Determinar o tamanho de uma cadeia (número de caracteres)
  - Pesquisa de uma sub-cadeia dentro de uma cadeia
  - Substituição de uma sub-cadeia dentro de uma cadeia

Cadeias em C++

- Para utilizar cadeias de caracteres (strings) em C++ é necessário incluir o arquivo *header*
  - **#include <string>**
- Strings são declaradas da mesma forma os tipos de dados primitivos, tais como inteiros ou reais, utilizando o tipo **string**
  - `string s1, s2, s3;`
    - ❖declara s1, s2 e s3 como variáveis do tipo string
- É possível declarar uma string e atribuir seu valor inicial junto ao comando de declaração
  - `string s1="Maria", s2="Pedro", s3="";`

Cadeias em C++

- Strings podem ser lidas utilizando o comando **cin**
  - `string s;`
  - `cin >> s;`
- Strings lidas por meio do comando **cin** são delimitadas por caracteres separadores (por exemplo, um espaço)
- Assim para ler uma string que contenha espaço é conveniente utilizar a função **getline** cujo delimitador é um caractere de nova linha ('\n')
  - `string s;`
  - `getline(cin,s);`

Cadeias em C++

- Concatenação
  - `string s1="Ciencia", s2="da", s3, s4;`
  - `s3 = "Computacao";`
  - `s4 = s1 + "␣" + s2 + "␣" + s3;`
  - A variável s4 conterá o valor "Ciencia da Computacao"
- Atribuição
  - O operador utilizado é o mesmo para operações aritméticas (**=**)
- Comparação
  - Os operadores relacionais podem ser utilizados (==, !=, <, <=, >, >=)

Cadeias em C++

- Comprimento
  - `int n;`
  - `string s;`
  - `s = "Ciencia";`
  - `n = s.length();`
  - A variável n conterá o valor 7
- Acesso a caracteres: Cada caractere k de uma string s pode ser acessado utilizando s[k], para 0<= k <= s.length()-1
  - `string s = "Ciencia";`
  - `s[0] = 'T';`
  - A variável s resultante conterá o valor "Tiencia"

Cadeias em C++

- Sub-Cadeia
  - `string x,s = "O aeroplano decolou";`
  - `x = s.substr(6,5);`
  - A variável x conterá a sub-cadeia "plano" que inicia no índice 6 e consiste de 5 caracteres
- Pesquisa por sub-cadeia
  - `int pos;`
  - `string s = "O aeroplano decolou";`
  - `pos = s.find("de");`
  - A variável pos conterá o valor 12 que é o índice onde a sub-cadeia "de" foi encontrada
  - Se a sub-cadeia não é encontrada, o sistema retorna a constante **string::npos**

Cadeias em C++

- Apagando caracteres
  - `string s = "O aeroplano decolou";`
  - `s.erase(11);`
  - A variável s conterá a sub-cadeia "O aeroplano"; todos os caracteres desde o índice 11 (inclusive) até o final da cadeia são apagados
- Substituindo caracteres
  - `string s = "O aeroplano decolou";`
  - `s.replace(2,9,"aviao");`
  - A variável s conterá a cadeia "O aviao decolou"; todos 9 caracteres que iniciam no índice 2 foram trocados por "aviao"
- Existem outros métodos (funções e procedimentos) de manipulação de strings em C++

Exercícios

1. Elabore um algoritmo que leia uma cadeia de caracteres, escrevendo-a de trás para frente
2. Elabore um algoritmo que leia uma cadeia de caracteres, converta todas as letras minúsculas em maiúsculas e todas as letras maiúsculas em minúsculas, escrevendo a cadeia resultante
3. Elabore um algoritmo que leia uma cadeia de caracteres e verifique se é um palíndromo ou não. Um palíndromo é uma palavra ou sentença que é lida da mesma forma, tanto da direita para esquerda, como da esquerda para a direita. Por exemplo, “arara” é um palíndromo enquanto “araras” não é
4. Elabore um algoritmo para calcular a freqüência de letras em um texto

Solução Exercício 1

```
#include <iostream>
#include <string>
using namespace std;
//-------------------------------------
// Retorna string s de tras para frente
string reverso(string s)
{ int i;
  string r="";

  for(i=s.length()-1; i>=0 ; i--)
    r = r + s[i];
  return r;
}
//-------------------------------------
int main()
{ string frase;

  cout << "Frase?";
  getline(cin,frase);
  cout << "Frase fornecida: " << frase << endl
       << "Frase reversa..: " << reverso(frase) << endl;
  return 0;
}
```


Solução Exercício 2

```
#include <iostream>
#include <string>
using namespace std;
//-------------------------------------
// Troca letra maiúscula por minúscula
// e vice-versa
char troca(char c)
{ char resultado;

  if('a' <= c && c <= 'z')
    resultado = c - 'a' + 'A';
  else
    if('A' <= c && c <= 'Z')
      resultado = c - 'A' + 'a';
    else
      resultado = c;
  return resultado;
}
//-------------------------------------
```

```
// Troca todas as letras maiúsculas
// por minúsculas e vice-versa
string MaiusculasMinusculas(string s)
{ int i;
  string s2="";

  for(i=0; i<s.length(); i++)
    s2 = s2 + troca(s[i]);
  return s2;
}
//-------------------------------------
int main()
{ string s;

  cout << "Frase?";
  getline(cin,s);
  cout << "Frase fornecida: " << s
       << endl
       << "Frase final....: "
       << MaiusculasMinusculas(s)
       << endl;

  return 0;
}
```


Solução Exercício 3

```
#include <iostream>
#include <string>
using namespace std;
//-----------------------------
// Verifica se string é
// palindromo
bool palindromo(string s)
{ int i,j;

  i = 0;
  j = s.length()-1;
  while (i < j)
  { if(s[i] != s[j])
      return false;
    i++;
    j--;
  }
  return true;
}
```

```
//-----------------------------
int main()
{ string s;

  cout << "Frase?";
  getline(cin,s);
  if(palindromo(s))
    cout << s
         << " palindromo"
         << endl;
  else
    cout << s
         << " nao palindromo"
         << endl;
  return 0;
}
```


Solução Exercício 4

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

int main()
{ const string ALFABETO =
    "abcdefghijklmnopqrstuvwxyz";
  const int TAMANHO=ALFABETO.length();
  int total_letras = 0, tamanho_texto,
    i, j;
  float frequencia[TAMANHO];
  char caractere;
  string texto;

  cout << "Texto: ";
  getline(cin,texto);
  tamanho_texto = texto.length();

  for(i=0; i<TAMANHO; i++)
    frequencia[i] = 0;
```

```
  for(i=0; i<tamanho_texto; i++)
  { caractere = tolower(texto[i]);
    j = ALFABETO.find(caractere);
    // encontrou?
    if (j != string::npos)
    { frequencia[j]++;
      total_letras++;
    }
  }

  cout << "Total de letras: "
       << total_letras
       << endl;
  for(i=0; i<TAMANHO; i++)
    cout << ALFABETO[i] << ": "
         << frequencia[i]
         << " ("
         << frequencia[i]/total_letras
         << ")" << endl;

  return 0;
}
```


Cadeias em C

- Strings em C são declaradas como um vetor de caracteres (char [])
- Toda string em C é terminada com o caracter ´\0´ (diferentemente do que ocorre em C++)
- Atenção
  - Constantes caractere são delimitadas por apóstrofos (´)
  - Constantes string são delimitadas por aspas (“)
- Por exemplo, para declarar uma string **nome** capaz de armazenar até 5 caracteres:
  - char nome[6];
  - Lembrar que o tamanho da string deve também contar o ´\0´ final (6 = 5 caracteres + ´\0´ final)

Cadeias em C

- É possível declarar uma string e dar o seu valor inicial junto ao comando de declaração
- Por exemplo:
  - char nome[7] = “Maria”;
  - char s[6] = “a bcde”;

	0	1	2	3	4	5	6
nome	M	a	r	i	a	\0	

	0	1	2	3	4	5
s	a		b	c	d	\0

Cadeias em C

- Problema:
  - char x[10], y[10];
  - x = “Ola”;
  - x = y; /* Não faça isso! */
  - x = x + y; /* Não faça isso! */
  - if (x == y)... /* Não faça isso! */
- Strings não podem ser atribuídas ou comparadas diretamente
- Strings são normalmente manipuladas por intermédio de funções de biblioteca
  - #include <string.h>

Cadeias em C

- Strings são normalmente manipuladas por meio de funções da biblioteca #include <string.h>:
  - gets(nome_da_string);
    - lê uma string do teclado, incluindo espaços em branco até encontrar <ENTER>
  - strcpy(string_destino,string_origem);
    - copia a string-origem para a string-destino
  - strcat(string_destino,string_origem);
    - a string de origem permanecerá inalterada e será anexada ao fim da string de destino
  - strlen(string);
    - retorna o comprimento da string fornecida
  - strcmp(string1,string2);
    - compara a string1 com a string2, retorna -1 se string1 < string2, zero se string1 == string2 e +1 se string1 > string2. O compilador diferencia maiúsculas de minúsculas bem como de letras com e sem acento.
  - stricmp(string1,string2);
    - compara a string1 com a string2, retorna -1 se string1 < string2, zero se string1 == string2 e +1 se string1 > string2. O compilador não diferencia maiúsculas de minúsculas mas diferencia letras com e sem acento.
- Existem outras funções de manipulação de strings em C

Cadeias em C

- Solução
  - char x[10], y[10];

No lugar de:	**Utilize:**
x = “Ola”;	strcpy(x,”Ola”);
x = y;	strcpy(x,y);
x = x + y;	strcat(x,y);
if(x == y)...	if(strcmp(x,y) == 0)...
if(x < y)...	if(strcmp(x,y) == -1)...
if(x > y)...	if(strcmp(x,y) == 1)...

Exemplos

```
char x[10]=”abcde”,
   y[10]=”xyzw”;

cout <<”x=”<<x<<” e y=”<<y;
cout << ”Tamanhos: x=“ <<
  strlen(x)<<“,y=”<<strlen(y);
strcat(x,y);
cout <<”x=”<<x<<” e y=”<<y;
cout << ”Tamanhos: x=“ <<
  strlen(x)<<“,y=”<<strlen(y);
strcpy(x,”aaaa”);
strcpy(y,”AAAA”);
cout <<”x=”<<x<<” e y=”<<y;
cout <<”Comparar “<<x<<“ e ”<<
   y << “=“ strcmp(x,y);
cout <<”Comparar “<<y<<“ e ”<<
   x << “=“ strcmp(y,x);
cout <<”Comparar “<<x<<“ e ”<<
   x << “=“ strcmp(x,x);
```

Resultados

```
x=abcde e y=xyzw
Tamanhos: x=5, y=4

x=abcdexyzw e y=xyzw
Tamanhos: x=9, y=4

x=aaaa e y=AAAA
Comparar aaaa e AAAA=1

Comparar AAAA e aaaa=-1

Comparar aaaa e aaa=0
```


Exemplo gets

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{ char n[100];

  cout << "Digite o seu nome: ";
  gets(n);
  cout << "\nOla " << n << endl;
  return 0;
}
```

Exemplo strcpy

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{ char str1[100],str2[100],str3[100];

  cout << "Entre uma string: ";
  gets(str1);
  strcpy(str2,str1);
  strcpy(str3,"Voce digitou a string ");
  cout << "\n\n" << str3 << str2;
  return 0;
}
```


Copiando Cadeias

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{ char a[100],b[100];

  gets(a);// Ler string

  // Copiar a para b
  strcpy(b,a);

  cout << "Copia = " <<b << endl;
  return 0;
}
```


Exemplo strcat

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{ char str1[100],str2[100];

  cout << "Entre uma string: ";
  gets(str1);
  strcpy(str2,"Voce digitou a string ");
  strcat(str2,str1);
  cout << str2 << endl;
  return 0;
}
```


Exemplo strlen

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{ char str[100];
  int tamanho;

  cout << "Entre uma string: ";
  gets(str);
  tamanho = strlen(str);
  cout << "\nA string tem tamanho = "
       << tamanho << endl;
  return 0;
}
```


Exemplo strcmp

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{ char str1[100],str2[100];

  cout << "Entre uma string: ";
  gets(str1);
  cout << "\nEntre outra uma string: ";
  gets(str2);
  switch(strcmp(str1,str2))
  { case -1: cout << "\n1a. String < 2a. String";
             break;
    case  0: cout << "\n1a. String = 2a. String";
             break;
    case +1: cout << "\n1a. String > 2a. String";
             break;
  }
  return 0;
}
```


Resumo

- Nesta aula vimos alguns conceitos básicos sobre strings
- Strings podem ser manipuladas como objetos (tipo **string** em C++) ou como um vetor de caracteres (tipo **char [ ]** em C)
- Note que há um tratamento especial para vetores de caracteres em C, uma vez que ele pode ser lido (cin, gets) ou escrito (cout) diretamente, o que não ocorre com outros tipos de vetores
- É interessante saber o comportamento de ambos os tipos de dados devido à existência de código utilizando vetores de caracteres
- Entretanto, é recomendável que o programador escolha um dos tipos e trabalhe somente com ele de modo a melhorar a consistência do trabalho