cadeia de caracteres (string)

INF1005 -- Programação I -- 2016.1
Prof. Roberto Azevedo
razevedo@inf.puc-rio.br

cadeia de caracteres (string)

tópicos

- caracteres
- cadeia de caracteres

referência

- Capítulo 7 do livro

caracteres

tipo char

- tamanho de char:

  = 1byte = 8bits = 256 valores distintos

tabela de códigos

- define correspondência entre caracteres e códigos numéricos
- Exemplo: ASCII
- Alguns alfabetos precisam de maior representatividade (alfabeto chinês tem mais de 256 caracteres)

códigos ASCII de alguns caracteres (sp representa espaço)

	0	1	2	3	4	5	6	7	8	9
30			*sp*	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			

82	**105**	**110**	**32**	**100**	**101**	**32**	**74**	**97**	**110**	**101**	**105**	**114**	**111**
R	i	o		d	e		J	a	n	e	i	r	o

códigos ASCII de alguns caracteres de controle

0	`nul`	*null*: nulo
7	`bel`	*bell*: campainha
8	`bs`	*backspace*: volta e apaga um caractere
9	`ht`	*tab*: tabulação horizontal
10	`nl`	*newline* ou *line feed*: muda de linha
13	`cr`	*carriage return*: volta ao início da linha
127	`del`	*delete*: apaga um caractere

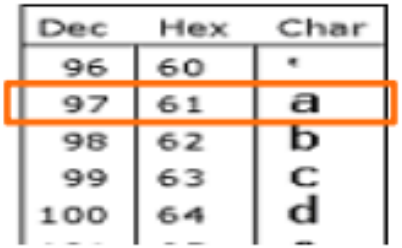
tabela ASCII

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

constante de caractere

- Caractere envolvido com aspas simples:
  - ‘a’ representa uma constante de caractere
  - ‘a’ resulta no valor numérico associado ao caractere a, conforme a tabela ASCII



Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d

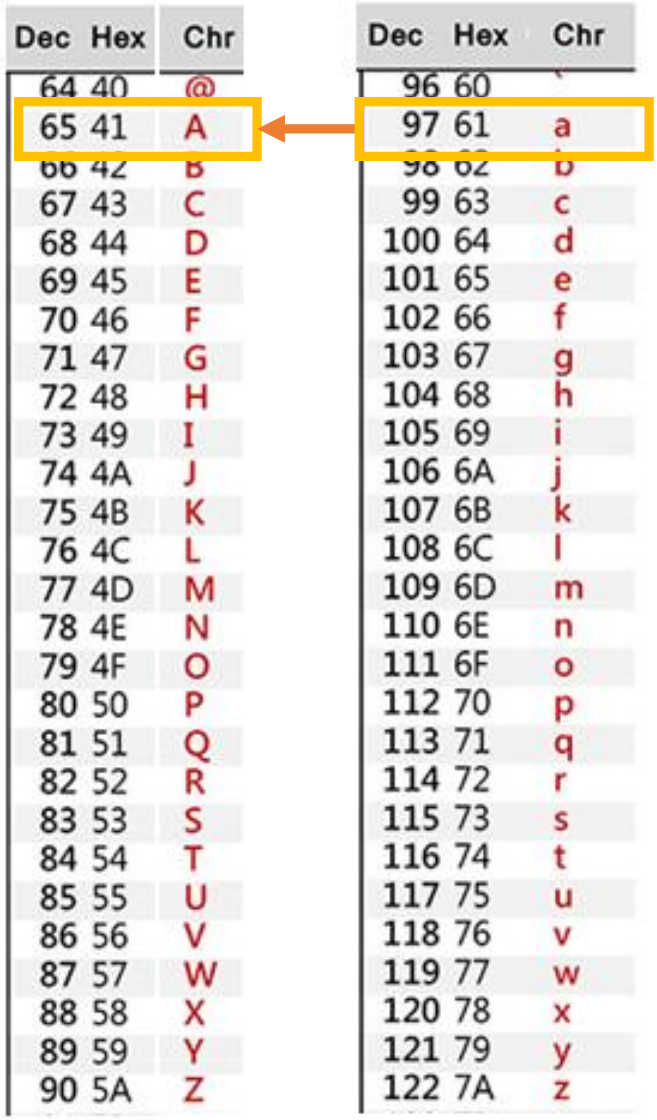
```
char c = 'a';

printf("%d %c\n", c, c);
```

```
97 a
```

exemplo 01

- Considerando a representação das letras na tabela ASCII, como converter um caractere minúsculo no seu correspondente maiúsculo?



Dec	Hex	Chr
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z

Dec	Hex	Chr
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z

exemplo 01

- conversão para maiúscula se for minúscula

```
char maiuscula(char letra)
{
    /* verifica se é letra minúscula */
    if (letra >= 'a' && letra <= 'z')
    {
        letra = letra - 'a' + 'A';
        return letra;
    }
}
```

como representar palavras?

cadeia de caracteres – representação

vetor do tipo char, terminado pelo caractere nulo (‘\0’)

- {‘R’, ‘i’, ‘o, ‘\0’}

é necessário reservar uma posição adicional no vetor para o caractere de fim de cadeia

- Rio de Janeiro tem 14 letras ou espaços:
- {‘R’, ‘i’, ‘o, ‘ ’, ‘d’, ‘e’, ‘ ‘, ‘J’, ‘a’, ‘n’, ‘e’, ‘i’, ‘r’, ‘o’, ‘\0’} tem 15 char

funções para manipular cadeias de caracteres:

- recebem como parâmetro um vetor de char
- processam caractere por caractere até encontrar o caractere nulo, sinalizando o final da cadeia

cadeia de caracteres – inicialização

- caracteres entre aspas duplas
- caractere nulo é representado implicitamente
- exemplo:
  - variável cidade diensionada e incializada com 4 elementos

```
int main (void)
{
   char cidade[] = "Rio";
   printf("%s \n", cidade);
   return 0;
}
```

≡

```
int main (void)
{
   char cidade[]={'R', 'i', 'o', '\0'};
   printf("%s \n", cidade);
   return 0;
}
```

```
Rio
```

cadeia de caracteres – exemplos

```
char s1[] = "";
char s2[] = "Rio de Janeiro";
char s3[81];
char s4[81] = "Rio";
```

s1 armazena o caractere '\0' (cadeia de caractere é dita vazia)
(vetor s1 tem apenas um elemento)

s2 armazena cadeia de 14 caracteres
(vetor s2 tem 15 elementos)

s3 Armazena cadeis com até 80 caracteres;
dimensionada com 81 elementos, mas não inicializada

s4 armazena cadeias com até 80 caracteres;
primeiros quatro elementos atribuídos na declaração são:
{'R', 'i', 'o', '\0'}

cadeia de caracteres – leitura do teclado

scanf com o especificador de formato %c

- lê o valor de um único caractere fornecido via teclado
- não pula caracteres brancos: espaço (‘ ‘), tabulação ‘\t’

```
char a;
...
scanf("%c", &a);
...
```

```
espaço
> b
/* a recebe o caractere ' ' */
```

- para pular todos os “caracteres brancos”, basta colocar um espaço antes da entrada:

```
char a;
...
scanf(" %c", &a);
...
```

```
espaço
> b
/* a recebe o caractere 'b' */
```

cadeia de caracteres – leitura do teclado

- **scanf** com o especificador **%s**
  - lê uma cadeia de caractere **não brancos**
  - pula os eventuais caractere brancos antes da cadeia

```
char cidade[81];
...
scanf("%s", cidade);
...
```

```
> Rio de Janeiro
```

cidade receberá o quê?

cidade receberá apenas "Rio"

Por que isto está errado?

```
char cidade[81];
...
scanf("%s", &cidade);
...
```

Como a variável cidade foi declarada como vetor, já é o endereço onde os dados lidos devem ser armazenados.

como ler “tudo” até o final?

cadeia de caractere – leia enquanto for ...

scanf com o especificador de formato **%[...]**

%[...] lista entre os colchetes todos os caracteres aceitos na leitura

%[aeiou]

lê sequências de vogais

leitura prossegue até encontrar um caractere que não seja uma vogal

%[^...] lista entre os colchetes todos os caracteres **NÃO** aceitos na leitura

%[^aeiou]

lê sequências de caracteres que não são vogais

leitura prossegue até encontrar um caractere que seja uma vogal

cadeia de caracteres – leitura até o final

como ler uma sequência de caracteres até que seja encontrado o caractere de mudança de linha ('\n')?

- captura linha fornecida pelo usuário até que ele tecle "Enter"
- inclusão do espaço no formato garante que eventuais caracteres brancos que precedam a cadeia de caracteres sejam descartados

```
char cidade[81];
...
scanf(" %80[^\n]", cidade);
...
```

(exemplos de) funções para manipular cadeia de caracteres

cadeia de caracteres

- exemplos de funções para manipular cadeia de caracteres:
  - “exibe”
  - “comprimento”
  - “copia”
  - “concatena”
  - “compara”

cadeia de caracteres – função “exibe”

- exibe uma cadeia de caracteres, caracter por caracter, com uma quebra de linha no final

```
void exibe (char s[]) {
   int i;
   for (i=0; s[i] != '\0'; i++)
      printf("%c", s[i];
   printf("\n");
}
```

```
void exibe (char s[]) {
  printf("%s\n", s);
}
```

cadeia de caracteres – função “comprimento”

- retorna o comprimento de uma cadeia de caracteres de entrada s.
  - Conta o número de caracteres até encontrar o caractere **nulo ‘\0’** (o caractere nulo não é contado).

```
int comprimento (char s[])
{
   int i;
   int n = 0;  /* contador */
   for (i=0; s[i] != '\0'; i++)
      n++;
   return n;
}
```

```
int comprimento (char s[])
{
   int i;
   for (i=0; s[i] != '\0'; i++)
      ;
   return i;
}
```

testando a função comprimento

```c
#include <stdio.h>

int comprimento (char s[])
{
   int i;
   int n = 0;  /* contador */
   for (i=0; s[i] != '\0'; i++)
      n++;
   return n;
}

int main (void)
{
   int tam;
   char cidade[] = "Rio de Janeiro";
   tam = comprimento(cidade);
   printf("A string \"%s\" tem %d caracteres\n", cidade, tam);
   return 0;
}
```

exercício 01

- Com base na função anterior, escreva uma função que receba uma cadeia de caracteres e exiba-a de trás para frente. Por exemplo:

  exibe_rev (“Rio de Janeiro”); -> exibe **orienaJ ed oiR**

cadeia de caracteres – função “copia”

copia os elementos de uma cadeia de origem (**orig**) para uma cadeia de destino (**dest**)

dest deverá ter espaço suficiente

```
void copia (char dest[], char orig[])
{
   int i;
   for (i=0; orig[i] != '\0'; i++)
      dest[i] = orig[i];
   /* "fecha" a cadeia copiada */
   dest[i] = '\0';
}
```

cadeia de caracteres – função “concatena”

copia os elementos de uma cadeia de origem (**orig**) para o final da cadeia de destino (**dest**)

dest deverá ter espaço suficiente

```
void concatena (char dest[], char orig[])
{
  int i = 0;       /* indice usado na cadeia destino, inicializado com zero */
  int j;           /* indice usado na cadeia origem */
  /* acha o final da cadeia destino */
  while (dest[i] != '\0')
    i++;
  /* copia elementos da origem para o final do destino */
  for (j=0; orig[j] != '\0'; j++)
  {
    dest[i] = orig[j];
    i++;
  }
  /* "fecha" cadeia destino */
  dest[i] = '\0';
}
```

cadeia de caracteres – função “compara”

- compara, caractere por caractere, duas cadeias dadas

  Usa os códigos numéricos associados aos caracteres para determinar a ordem relativa

- valor de retorno da função

  -1 se a primeira cadeia preceder a segunda (e.g. “ar” vs. “pé”)

  1 se a segunda cadeia preceder a primeira (e.g. “pé” vs “ar”)

  0 se ambas as cadeias tiverem a mesma sequência

```c
int compara (char s1[], char s2[])
{
   int i;
   /* compara caractere por caractere */
   for (i = 0; s1[i] != '\0' && s2[i] != '\0'; i++) {
      if (s1[i] < s2[i])
         return -1;
      else if (s1[i] > s2[i])
         return 1;
   } /* termina o for quando pelo menos uma das cadeias terminar */
   /* compara se cadeias têm o mesmo comprimento */
   if (s1[i] == s2[i])
      return 0;          /* as duas cadeias terminaram (com '\0'):
                            cadeias iguais */
   else if (s2[i] != '\0')
      return -1;         /* s1 é menor, pois tem menos caracteres */

   else
      return 1;          /* s2 é menor, pois tem menos caracteres */
}
```

cadeia de caracteres – biblioteca string.h

strlen "comprimento"

```
int strlen (char* s);
```

strcpy "copia"

```
char* strcpy (char* destino, char* origem);
```

strcat "concatena"

```
char* strcat (char* destino, char* origem);
```

strcmp "compara"

```
int strcmp (char* s, char *t);
```

constante cadeia de caracteres

representada por uma sequência de caracteres delimitada por aspas duplas

comporta-se como uma expressão constante, cuja avaliação resulta no ponteiro para onde a cadeia de caracteres está armazenada

```
#include <string.h>

int main ( void )
{
   char cidade[4];
   strcpy (cidade, "Rio" );
   printf ( "%s \n", cidade );
   return 0;
}
```

constante cadeia de caracteres – exemplo

```
#include <string.h>

int main ( void )
{
   char cidade[4];
   strcpy (cidade, "Rio");
   printf ("%s \n", cidade);
   return 0;
}
```

```
int main (void)
{
   char *cidade;
   cidade = "Rio";
   printf ("%s \n", cidade);
   return 0;
}
```

quando a cadeia “Rio” é encontrada:

- uma área de memória é alocada com a sequência
  - {‘R’, ‘i’, ‘o’, ‘\0’}
- o ponteiro para o primeiro elemento desta sequência é devolvido

função strcpy recebe dois ponteiros de cadeias:

- o primeiro aponta para o espaço associado à variável cidade
- o segundo aponta para a área onde está armazenada a cadeia constante

cadeia de caracteres

exemplos:

```
char s1[] = "Rio de Janeiro";
```

s1 é um vetor de char, inicializado com a cadeia **Rio de Janeiro**, seguida do caractere nulo.

s1 ocupa **15 bytes** de memória

é válido escrever **s1[0] = 'X'**, alterando o conteúdo da cadeia para **Xio de Janeiro**, pois **s1** é um vetor.

```
char* s2 = "Rio de Janeiro";
```

s2 é um ponteiro para **char**, inicializado com o endereço da área de memória onde a constante **Rio de Janeiro** está armazenada
s2 ocupa **4 bytes** (espaço de um ponteiro)
não é válido escrever **s2[0]='X'**, pois **não é possível alterar um valor constante**