Strings

Linguagem de programação
Prof. Allan Rodrigo Leite

Strings

- *Strings* são cadeia (sequências) de caracteres justapostos
  - São fundamentais no desenvolvimento de software
  - Úteis para representação de textos ou similares

- Exemplos de sequências de caracteres
  - Mensagem de e-mail
  - Texto de um programa
  - Nome de clientes, alunos, etc.
  - Endereço em um cadastro
  - Sequencia genética
  - ...

Representação de caracteres

- Os caracteres em C são representados por códigos numéricos
  - São capazes de representar letras, números e símbolos
  - Os códigos compõe a tabela ASCII

- ASCII (American Standard Code for Information Interchange)
  - Padrão para representação de caracteres para sistemas eletrônicos
  - Primeira edição de publicação foi em 1963
  - Embora seja antigo, os esquemas modernos são baseados no ASCII
    - Estes esquemas modernos suportam caracteres adicionais
    - Um exemplo amplamente utilizado de esquema moderno é o UTF-8

Tabela ASCII

Representação de letras, números e símbolos

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Representação de caracteres

- Como caracteres são representações numéricas, a diferença entre caracteres e inteiros está apenas na maneira como são tratados
  - Exemplo

```
char c = 97;
printf("int : %d char : %c\n", c, c); //exibirá 97 e a
```

  - O valor exibido pelo `printf` utilizou dois formatos distintos
    - %d (int)
    - %c (char)

Representação de caracteres

- Evite o uso números diretamente para representar caracteres
  - Esta recomendação visa aumentar a portabilidade do programa
  - Ao invés, usa-se uma constante caractere
    - Isto é, um caractere entre apóstrofe
  - Exemplo

```
char c = 'a';
printf("int : %d char : %c\n", c, c); //exibirá 97 e a
```

  - Constantes de caracteres suportam o uso de operadores lógicos
    - <, <=, ==, >=, >, ! =

Sequência de caracteres

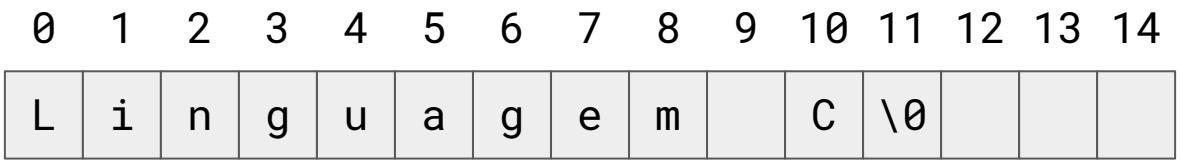
- É possível armazenar um único caractere em uma variável `char`
  - Para vários caracteres utiliza-se uma sequência de caracteres
  - Esta sequência é representada por vetores do tipo `char`

- Estas sequências de caracteres são conhecidas como *strings*
  - Na linguagem C, uma string é representada por
    - Um vetor de caracteres
    - O final da sequência deve conter o caractere nulo (`'\0'`)
    - As *strings* podem ser representadas como literais com aspas duplas

Strings

- Exemplo de uma *string*

```
char disciplina[15] = "Linguagem C";
printf("Disciplina: %s\n", disciplina);
```



- Inicializando uma *string*

```
char disciplina[15] = "Linguagem C";
char disciplina[15] = {'L','i','n','g','u','a','g','e','m',' ','C','\0'};
```

Manipulando *strings*

- A manipulação individual dos caracteres ocorre igual à vetores
  - O acesso aos caracteres são realizados por meio dos índices do vetor
    - Para impressão de uma *string* com `printf`, usa-se o formato %s
  - Exemplo

```
char disciplina[15] = "Linguagem C";
printf("Soletrando -");

for (int i = 0; i < 15; i++) {
     if (disciplina[i] == '\0')
          break;

     printf("%c-", disciplina[i]);
}

printf("\n");
```

Manipulando *strings*

- O C dispõe de diversas funções para manipulação de *strings*
  - Estas funções estão na biblioteca `string.h`

- As principais funções desta biblioteca são:
  - `strcpy(char destino[], char origem[])`
    - Copia o conteúdo da *string* origem para a *string* destino
  - `strlen(char str[])`
    - Retorna o tamanho da string baseado na posição do caractere nulo
  - `strcat(char destino[], char origem[])`
    - Concatena (junta) o conteúdo da *string* origem com a *string* destino
    - O resultado desta junção é armazenado na *string* destino

Exercícios

1. Faça um programa que leia uma cadeia de caracteres e, em seguida, o programa deve contar a quantidade de espaços em branco desta cadeia. Não deve ser utilizada nenhuma função de manipulação de *strings* da biblioteca `string.h`.
   - Exemplo 1, dada a *string* `"abc"`, a saída esperada é `0`.
   - Exemplo 2, dada a *string* `"a  b  c"`, a saída esperada é `3`.

2. Altere o programa do exercício 1 para considerar os espaços adjacentes apenas como uma unidade na contagem. Também não deve ser utilizado nenhuma função de manipulação de *strings* contida na biblioteca `string.h`.
   - Exemplo 1, dada a *string* `"abc"`, a saída esperada é `0`.
   - Exemplo 2, dada a *string* `"a  b  c"`, a saída esperada é `2`.

3. Faça um programa que leia uma cadeia de caracteres e, em seguida, o programa deve retornar a mesma cadeia de caracteres com todas as letras em maiúsculo. Não deve ser utilizado nenhuma função de manipulação de *strings* da biblioteca `string.h`.
   - Exemplo 1, dada a *string* `"abc"`, a saída esperada é `"ABC"`.
   - Exemplo 2, dada a *string* `"AbC"`, a saída esperada é `"ABC"`.

Exercícios

4. Faça um programa que leia uma cadeia de caracteres e, em seguida, o programa deve retornar `true` se a cadeia de caracteres forma um número inteiro, caso contrário deve retornar `false`. A verificação deve considerar a existência ou não de separadores de milhar para o idioma português. Não deve ser utilizado nenhuma função de manipulação de *strings* da biblioteca `string.h`.
    - Exemplo 1, dada a *string* `"-150"`, a saída esperada é `true`.
    - Exemplo 2, dada a *string* `"-150.0"`, a saída esperada é `false`.

5. Faça um programa que leia duas cadeia de caracteres e, em seguida, o programa deve retornar `true` se a segunda cadeia estiver contida na primeira (*substring*), caso contrário deve retornar `false`. Não deve ser utilizado nenhuma função de manipulação de *strings* da biblioteca `string.h`.
    - Exemplo 1, dadas as *strings* `"abcd"` e `"cd"`, a saída esperada é `true`.
    - Exemplo 2, dadas as *strings* `"abcd"` e `"cb"`, a saída esperada é `false`.

6. Faça um programa que leia uma cadeia de caracteres e, em seguida, o programa deve retornar `true` se a cadeia forma um palíndromo, caso contrário deve retornar `false`.
    - Exemplo 1, dada a *string* `"abcd"`, a saída esperada é `false`.
    - Exemplo 2, dada a *string* `"abba"`, a saída esperada é `true`.

Exercícios

7. *Run-length encoding* (RLE) é uma técnica simples para compressão de textos. A ideia desta técnica é representar caracteres repetidos sucessivamente com um contador seguido pelo caractere. Dada uma *string*, retorne o texto resultante da aplicação da técnica RLE.
    - Exemplo, dada a *string* `"AAAABBBCCDAA"`, a saída compactada deve ser `"4A3B2C1D2A"`.

8. Distância de edição é uma medida para indicar o quão próximas são duas *strings*. Esta medida é calculada a partir do número mínimo de operações necessárias para transformar uma *string* na outra. As operações válidas são: inserção, deleção ou substituição de um caractere. Faça um programa que leia duas cadeias de caracteres e, em seguida, o programa deve retornar o número de operações necessárias para transformar a primeira cadeia na segunda. A solução proposta deve implementar o algoritmo de Levenshtein.
    - Exemplo 1, dadas as *strings* `"exercício"` e `"exército"`, a saída esperada é 4.
        - `"exercício"`
        - `"exército"`
        - `"__s__ssr" //_: igual, s: substituição, r: remoção`

Exercícios

9. Dada uma *string* de parênteses, gere uma nova *string* com os parênteses balanceados produzida com o menor número de operações, considerando inserção ou deleção. Se houver mais de uma solução, retorne a primeira encontrada
    - Exemplo 1, dada a *string* " (( ) ", uma possível saída será " (( )) ".
    - Exemplo 2, dada a *string* ")) ( ) (", uma possível saída será " ( ) ( ) ( ) ( ) ".

Strings

Linguagem de programação
Prof. Allan Rodrigo Leite