



Strings

Flavius Gorgônio
flavius@dct.ufrn.br



Agenda

- Representação interna de caracteres (Tabela ASCII)
- Cadeias de caracteres em linguagem C
- Leitura e escrita de cadeias de caracteres
- Funções úteis
- Vetores de cadeias de caracteres

Representação interna de caracteres

A linguagem C não dispõe de um tipo específico para armazenar cadeias de caracteres (strings)

Ao invés disso, oferece o tipo **char** que armazena um único caractere, com a possibilidade de criação de vetores de **char** para armazenar um conjunto maior de caracteres

Uma variável do tipo char pode armazenar valores entre 0 e 255, que estão relacionados aos caracteres de uma tabela interna de códigos do computador (em geral, usa-se a Tabela ASCII)

Tabela ASCII

	00	16	32	48	64	80	96	112
0	NUL DLE			0	@	P	`	p
1	SOH DC1	!	1	A	Q	a	q	
2	STX DC2	"	2	B	R	b	r	
3	ETX DC3	#	3	C	S	c	s	
4	EOT DC4	\$	4	D	T	d	t	
5	ENQ NAK	%	5	E	U	e	u	
6	ACK SYN	&	6	F	V	f	v	
7	BEL ETB	'	7	G	W	g	w	
8	BS CAN	(8	H	X	h	x	
9	HT EM)	9	I	Y	i	y	
10	LF SUB	*	:	J	Z	j	z	
11	VT ESC	+	;	K	[k	{	
12	FF FS	,	<	L	\	l		
13	CR GS	-	=	M]	m	}	
14	SO RS	.	>	N	^	n	~	
15	SI US	/	?	O	_	o	DEL	

ASCII	Caracteres de controle
0	null (nulo)
7	bell (campainha)
8	backspace (retorna e apaga um caractere)
9	tab (tabulação horizontal)
10	newline (avança para a próxima linha)
13	carriage return (retorna ao início da linha)
127	delete (apaga um caractere)

Tabela ASCII Estendida

Regular ASCII Chart (character codes 0 - 127)															
000	<nul>	016	<dle>	032	sp	048	0	064	@	080	P	096	`	112	p
001	<soh>	017	<dc1>	033	!	049	1	065	A	081	Q	097	a	113	q
002	<stx>	018	<dc2>	034	"	050	2	066	B	082	R	098	b	114	r
003	<etx>	019	<dc3>	035	#	051	3	067	C	083	S	099	c	115	s
004	<eot>	020	<dc4>	036	\$	052	4	068	D	084	T	100	d	116	t
005	<enq>	021	<nak>	037	%	053	5	069	E	085	U	101	e	117	u
006	<ack>	022	<syn>	038	&	054	6	070	F	086	V	102	f	118	v
007	<bel>	023	<etb>	039	'	055	7	071	G	087	W	103	g	119	w
008	<bs>	024	<can>	040	<	056	8	072	H	088	X	104	h	120	x
009	<tab>	025		041	>	057	9	073	I	089	Y	105	i	121	y
010	<lf>	026	<eof>	042	*	058	:	074	J	090	Z	106	j	122	z
011	<vt>	027	<esc>	043	+	059	;	075	K	091	[107	k	123	<
012	<np>	028	<fs>	044	,	060	<	076	L	092	\	108	l	124	!
013	<cr>	029	<gs>	045	-	061	=	077	M	093]	109	m	125	>
014	<so>	030	<rs>	046	.	062	>	078	N	094	^	110	n	126	~
015	<si>	031	<us>	047	/	063	?	079	O	095	_	111	o	127	Δ

Extended ASCII Chart (character codes 128 - 255)															
128	Ç	143	À	158	×	172	¼	186	¶	200	Ù	214	í	228	õ
129	ü	144	É	159	÷	173	½	187	§	201	Ú	215	î	229	ö
130	é	145	Æ	160	á	174	¾	188	¶	202	Û	216	ï	230	÷
131	â	146	Œ	161	í	175	»	189	§	203	Ü	217	í	231	µ
132	ä	147	ô	162	ó	176	¼	190	¥	204	Ý	218	ï	232	þ
133	à	148	ö	163	ú	177	½	191	£	205	Þ	219	ü	233	ú
134	ä	149	ò	164	ñ	178	¾	192	£	206	ß	220	ü	234	ò
135	ç	150	û	165	Ñ	179	¼	193	£	207	À	221	ü	235	ù
136	è	151	ù	166	é	180	½	194	£	208	Á	222	ü	236	ý
137	ë	152	ÿ	167	ê	181	¾	195	£	209	Â	223	ü	237	ÿ
138	è	153	ö	168	ë	182	¼	196	£	210	Ã	224	ó	238	ÿ
139	ï	154	Ü	169	®	183	½	197	£	211	Ä	225	ø	239	ÿ
140	î	155	Ö	170	¯	184	¾	198	£	212	Å	226	ö	240	ÿ
141	ì	156	£	171	¼	185	¼	199	£	213	Ä	227	ö	241	±
142	ä	157	Ø											242	=
														243	¼
														244	¶
														245	§
														246	÷
														247	ò
														248	ò
														249	ü
														250	ü
														251	ü
														252	ü
														253	ü
														254	ü
														255	ü

Imprimindo caracteres

```
#include <stdio.h>

int main(void) {
    char letra;

    letra = 's';
    printf("Letra: %c - Código ASCII: %d\n", letra, letra);

    letra = 86;
    printf("Letra: %c - Código ASCII: %d\n", letra, letra);

    letra++;
    printf("Letra: %c - Código ASCII: %d\n", letra, letra);

    return 0;
}
```

Resultado da execução:

Letra: s - Código ASCII: 115
Letra: V - Código ASCII: 86
Letra: W - Código ASCII: 87

Lendo e exibindo caracteres

```
#include <stdio.h>

int ehLetra(char);

int main(void) {
    char letra;
    printf("Informe um caractere: ");
    scanf("%c", &letra);
    printf("Caractere Lido: %c\n", letra);
    printf("Código ASCII: %d\n", letra);
    if (ehLetra(letra)) {
        printf("%c é uma letra\n", letra);
    }
    else {
        printf("%c não é uma letra\n", letra);
    }
    return 0;
}
```

```
int ehLetra(char c) {
    if (c>='A' && c<='Z') {
        return 1;
    }
    else if (c>='a' && c<='z') {
        return 1;
    }
    else {
        return 0;
    }
}
```

Atividades

1. Crie uma função que receba um caractere como parâmetro e retorne 1 se o caractere recebido for um dígito ou 0 caso contrário. Use a seguinte assinatura para a função:

```
int ehDigito(char) ;
```

2. Modifique o programa do slide anterior de forma que ele classifique o caractere lido como letra, dígito ou caractere especial
3. Escreva uma função que receba um caractere como parâmetro e, caso o caractere recebido seja uma letra minúscula, retorne a sua equivalente em maiúsculas. Use a seguinte assinatura para a função:

```
char maiuscula(char) ;
```

4. Escreva uma função equivalente para retornar letras em minúsculas

Cadeias de caracteres (*strings*)

Cadeias de caracteres (*strings*) em linguagem C são representados por vetores de char, terminadas, obrigatoriamente, pelo caractere nulo '\0'

Cada cadeia de caracteres (*string*) deve reservar um caractere a mais para armazenar o caractere nulo, como terminador da *string*

Ao declarar uma string, você pode inicializá-la atribuindo valores (caracteres) às posições do vetor

C	a	i	c	o	\0
---	---	---	---	---	----

```
#include <stdio.h>

int main(void) {
    char cidade[6];
    char uf[] = {'R', 'N', '\0'};
    char pais[] = "Brasil";

    cidade[0] = 'C';
    cidade[1] = 'a';
    cidade[2] = 'i';
    cidade[3] = 'c';
    cidade[4] = 'o';
    cidade[5] = '\0';
    printf("Cidade: %s\n", cidade);
    printf("UF: %s\n", uf);
    printf("País: %s\n", pais);
    return 0;
}
```

Por que o terminador?

Por que é necessário finalizar as strings com o caractere nulo?

C	a	i	c	o	\0	#	\$	\$	3	@	a	q	5	4
---	---	---	---	---	----	---	----	----	---	---	---	---	---	---

O terminador de strings '\0' sinaliza o final da string dentro do vetor

A	c	a	r	i	\0	8	8	:)	(m	M	A	,
---	---	---	---	---	----	---	---	---	---	---	---	---	---	---

Se há uma variável do tipo vetor de caracteres, é possível preenchê-la sem usar todas as posições

C	u	r	r	a	i	s		N	o	v	o	s	\0	%
---	---	---	---	---	---	---	--	---	---	---	---	---	----	---

Por exemplo:

A	s	s	u	\0	<	\0	-	+		\n	}	}	}	k
---	---	---	---	----	---	----	---	---	--	----	---	---	---	---

```
int main(void) {  
    char cidade[15];  
    ...
```

S	~	a	o		F	e	r	n	a	n	d	o	\0	k
---	---	---	---	--	---	---	---	---	---	---	---	---	----	---

M	o	s	s	o	r	'	o	\0	\0	\0	\0	\0	\0	\0
---	---	---	---	---	---	---	---	----	----	----	----	----	----	----


Inicialização de strings

```
#include <stdio.h>
```

```
int main(void) {  
    char s1[] = "";  
    char s2[] = "Rio Grande do Norte";  
    char s3[51];  
    char s4[51] = "Programação";
```

```
    printf("S1: %s\n", s1);  
    printf("S2: %s\n", s2);  
    printf("S3: %s\n", s3);  
    printf("S4: %s\n", s4);  
    return 0;  
}
```

Resultado da execução:

S1:
S2: Rio Grande do Norte
S3: 
S4: Programação

s1	\0												
s2	R	i	o		G	r	a	n	d	e	...	t	e \0
s3	5	#	\$	w	@	\0	[^	1	...	=	\$ *
s4	P	r	o	g	r	a	m	a	ç	ã	o \0	...	

Lendo e exibindo strings

A função `scanf()` é uma das opções utilizadas para ler cadeias de caracteres (strings) em linguagem C

Entretanto, o uso da função `scanf()` possui algumas limitações

Por isso, existem algumas funções alternativas para esta tarefa, definidas na biblioteca `<string.h>`

```
#include <stdio.h>

int main(void) {
    char cidade[51];
    printf("Informe a cidade onde você nasceu: ");
    scanf("%s", cidade);
    printf("Você é natural de: %s\n", cidade);
    return 0;
}
```

Resultado da execução:

```
Informe a cidade onde você nasceu: Campina Grande
Você é natural de: Campina
```

Lendo e exibindo strings

A utilização da função `scanf("%s")`, em conjunto com o especificador de formato `%s` captura uma sequência de caracteres não brancos

Isso impede, por exemplo, a captura de nomes compostos (com espaço em branco entre as palavras)

A versão apresentada ao lado é mais robusta, capturando uma sequência

```
#include <stdio.h>

int main(void) {
    char cidade[51];
    printf("Informe a cidade onde você nasceu: ");
    scanf(" %50[^\n]", cidade);
    printf("Você é natural de: %s\n", cidade);
    return 0;
}
```

Resultado da execução:

```
Informe a cidade onde você nasceu: Campina Grande
Você é natural de: Campina Grande
```

Lendo e exibindo strings

Uma função bastante utilizada para leitura de cadeias de caracteres é a função `fgets()`, que substitui a antiga função `gets()`

A versão apresentada ao lado funciona de forma semelhante à anterior, só que é mais simples de usar

```
#include <stdio.h>

int main(void) {
    char nome[255];
    printf("Informe seu nome: ");
    fgets (nome, 255, stdin);
    printf ("Seu nome é: %s\n", nome);
    return 0;
}
```

Resultado da execução:

```
Informe seu nome: Fulano de Tal dos Anzóis
Seu nome é: Fulano de Tal dos Anzóis
```

Atividades

1. Escreva um programa em linguagem C que leia e exiba dados do usuário (nome, CPF, data de nascimento, etc). O programa deverá validar os dados lidos utilizando as funções desenvolvidas anteriormente:

```
int ehletra(char);
```

```
int ehDigito(char);
```

2. Exiba um relatório com todas as informações fornecidas pelo usuário impressas em letras maiúsculas, use a função existente:

```
char maiuscula(char);
```

Algumas funções da biblioteca <string.h>

```
// Copia o conteúdo da string2 na string1
char *strcpy (char *dest, const char *src);

// Copia os n primeiros caracteres da string2 na string1
char *strncpy(char *string1, const char *string2, size_t n);

// Concatena a string2 no final da string1
char *strcat(char *string1, const char *string2);

// Concatena n caracteres da string2 no final da string1
char *strncat(char *string1, const char *string2, size_t n);

// Compara duas strings e determina a ordem (alfabética) das duas
int strcmp(const char *string1, const char *string2);

// Compara os n primeiros caracteres de duas strings
int strncmp(const char *string1, char *string2, size_t n);

// Retorna o tamanho de uma string
int strlen(const char *string);
```


A função strcpy()

A função strcpy() copia o conteúdo de uma string para outra string

A string de origem (segundo parâmetro) permanecerá inalterada, mas a string de destino (primeiro parâmetro) terá seu valor alterado para o novo conteúdo

Sua forma geral:

```
char *strcpy(char *string1, const char
*string2);
```

Exemplo:

```
#include <stdio.h>
#include <string.h>

int main () {
    char str1[100] = "";
    char str2[100] = "";
    printf ("Entre com uma string: ");
    fgets (str1, 100, stdin);
    printf("Antes:\n");
    printf ("String 1 = %s\n", str1);
    printf ("String 2 = %s\n", str2);
    strcpy (str2, str1);
    printf("Depois:\n");
    printf ("String 1 = %s\n", str1);
    printf ("String 2 = %s\n", str2);
    return(0);
}
```

A função strcat()

A função strcat() concatena (agrupa) duas strings

A string de origem permanecerá inalterada, mas seu conteúdo será anexado ao fim da string de destino

Sua forma geral:

```
char *strcat(char *string1, const char
*string2);
```

Exemplo:

```
#include <stdio.h>
#include <string.h>

int main () {
    char str1[100] = "";
    char str2[100] = "";
    printf ("Entre com uma string: ");
    fgets (str1, 100, stdin);
    strcpy (str2, "Voce digitou a string ");
    strcat (str2, str1);
    printf ("\n\n String 2 = %s\n", str2);
    return(0);
}
```

Implementação e uso da função strlen()

```
#include <stdio.h>
#include <string.h>

int main(void) {
    int tam;
    char nome[255];

    printf("Informe seu nome: ");
    scanf("%s", nome);
    // fgets (nome, 255, stdin);
    // faça um teste usando a função fgets
    tam = strlen(nome);
    // tam = comprimento(nome);
    // teste a sua própria função strlen()
    printf ("Seu nome tem %d caracteres\n", tam);
    return 0;
}
```

```
int comprimento(const char* str) {
    int t = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        t++;
    }
    return t;
}
```

Erros comuns na manipulação de strings

Tentativa de atribuição direta:

```
char str1[100], str2[100], str3[100];

printf ("Entre com uma palavra: ");
fgets (str1, 100, stdin);

str2 = str1;           // Forma errada!!!
strcpy (str2, str1);    // Forma correta!!!
```

Tentativa de comparação direta:

```
char str1[100], str2[100], str3[100];

printf ("Entre com três palavras: ");
fgets (str1, 100, stdin);
fgets (str2, 100, stdin);
fgets (str3, 100, stdin);

if (str2 == str1)           // Errado!!!
if (strcmp(str2,str1) == 0); // Correto!!!
if (strncmp(str3,str2,n) == 0); // Correto!!!
```

Strings dinâmicas

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    char linha[256];
    char *nome, *email, *fone;
    int tam;

    printf("Alocação Dinâmica de Strings\n");
    printf("Digite seu nome: ");
    scanf(" %255[^\n]", linha);
    tam = strlen(linha);
    nome = (char*) malloc(tam+1);
    strcpy(nome, linha);

    printf("Digite seu e-mail: ");
    scanf(" %255[^\n]", linha);
    tam = strlen(linha);
    email = (char*) malloc(tam+1);
    strcpy(email, linha);
```

```
    printf("Digite seu celular: ");
    scanf(" %255[^\n]", linha);
    tam = strlen(linha);
    fone = (char*) malloc(tam+1);
    strcpy(fone, linha);
```

```
    printf("\nDados Lidos\n");
    printf("Nome: %s\n", nome);
    printf("E-mail: %s\n", email);
    printf("Celular: %s\n", fone);
```

```
    // Quando a string não for mais necessária
    free(nome);
    free(email);
    free(fone);
    return 0;
}
```

Vetor de strings estático

Vetor estático (matriz de caracteres):

```
char nomes[50][20];
```

Valores correspondentes aos limites de quantidade e tamanho devem ser definidos antecipadamente

A alocação total de memória é realizada quando é feita a declaração da variável

Desvantagens:

Pode haver superdimensionamento na quantidade de palavras e no tamanho de cada palavra

```
#include <stdio.h>

int main(void) {
    int tam;
    char nomes[50][20];
    printf("Vetor estático de palavras\n");
    printf("Quantas palavras deseja cadastrar? ");
    scanf("%d", &tam);
    for (int i = 0; i < tam; i++) {
        printf("Informe a palavra %d: ", i+1);
        scanf("%s", nomes[i]);
    }
    printf("\n\n= = Banco de Palavras = = \n");
    for (int i = 0; i < tam; i++) {
        printf("%d. %s\n", i+1, nomes[i]);
    }
    return 0;
}
```

Vetor de strings dinâmico

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static char* lelinha(void);
char* duplica(char*);

int main(void) {
    int tam;
    char **nomes;
    printf("Vetor dinâmico de palavras\n");
    printf("Quantas palavras deseja cadastrar? ");
    scanf("%d", &tam);
    nomes = (char**) malloc(tam * sizeof(char*));
    for (int i = 0; i < tam; i++) {
        printf("Informe a palavra %d: ", i+1);
        nomes[i] = lelinha();
    }
}
```

```
printf("\n\n= = Banco de Palavras = = \n");
for (int i = 0; i < tam; i++) {
    printf("%d. %s\n", i+1, nomes[i]);
}
return 0;
}
```

```
static char* lelinha(void) {
    char linha[256];
    // printf("Informe nome: "); removido
    scanf(" %255[^\n]", linha);
    return duplica(linha);
}
```

```
char* duplica(char* s) {
    int n;
    n = strlen(s) + 1;
    char* d = (char*) malloc(n * sizeof(char));
    strcpy(d, s);
    return d;
}
```

Bibliografia Recomendada

Celes, W. *et al.* Introdução a Estruturas de Dados com Técnicas de Programação em C, 2ª ed. Rio de Janeiro: Elsevier, 2016.

Sebesta, R. W., Conceitos de Linguagens de Programação, 9ª ed. Porto Alegre: Bookman, 2016.

Varejão, F., Linguagens de Programação: Conceitos e Técnicas, Rio de Janeiro: Campus, 2004.