

Algoritmos e Programação

Aula 11 - Strings

Priscila Delabetha

1 Caractere

- Tipo de dado: **char**
 - Representa uma letra
 - Uma variável do tipo char armazena um número, que corresponde a uma letra da tabela ASCII

Tabela ASCII

- **ASCII** (Código Padrão Americano para o Intercâmbio de Informação) é uma codificação de caracteres de oito bits baseada no alfabeto inglês. Os códigos ASCII representam texto em computadores, equipamentos de comunicação, entre outros dispositivos que trabalham com texto.
- A codificação define **128 caracteres**, preenchendo completamente os sete bits disponíveis. Desses, **33 não são imprimíveis**, como caracteres de controle atualmente não utilizáveis para edição de texto, porém amplamente utilizados em dispositivos de comunicação, que afetam o processamento do texto. Exceto pelo caractere de espaço, o restante é composto por caracteres imprimíveis.

Tabela ASCII

Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII
0	00	NUL	32	20	(blank)	64	40	@	96	60	~
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	=	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	\$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	,	71	47	G	103	67	g
8	08	BS	40	28	(72	48	H	104	68	h
9	09	HT	41	29)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	,	91	5B	[123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	(delete)

Leitura

```
char caract;
```

```
scanf(" %c", &caract);
```

```
caract = getchar( );
```

Precisamos de algo mais

E se eu solicitar ao meu usuário que digite seu nome para armazenar?

O CNPJ de uma empresa?

O usuário para o ranking de um jogo?

Não é viável ler um caractere de cada vez, nem imprimi-los!

Strings

- **Strings são cadeias ou sequências de caracteres.**
- Em C, as strings são vetores de caracteres que têm como diferencial um **caractere delimitador de fim '\0'** (caractere da posição 0 da tabela ASCII).
- Toda string é um vetor de caracteres, mas nem todo vetor de caracteres é uma string
- Exemplo:

vet	b	r	a	s	i	l	\0		
-----	---	---	---	---	---	---	----	--	--

um vetor de caracteres **vet** de 9 posições, apenas com as posições de 0 a 6 ocupadas e as outras posições com lixo.

Mas por que o \0 ?

Esse caractere é o 0 do código ASCII (não é o caractere 0, é o elemento de número 0 do código ASCII), e é o **delimitador de final de string**.

Ou seja, ele representa o fim de uma string.

Se não tiver o delimitador ao fim de um vetor de caracteres, não é string, é apenas um vetor de caracteres (sim, são coisas diferentes).

String

- Declaração

```
char identificador[número de caracteres];
```

- Exemplo

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

- O tamanho de uma string deve sempre prever a inclusão do caractere delimitador '\0'. ← Isso é um caractere!

Inicialização de strings

A inicialização de strings pode ser feita de várias maneiras:

- Exemplo 1:
 - `char primeiro_nome[15] = "Ana";`
 - O sistema insere os caracteres indicados entre as aspas duplas no vetor `primeiro_nome`, a partir da posição 0, e insere na posição 3 do arranjo o caractere '\0'.
- Exemplo 2:
 - `char primeiro_nome[15] = { 'A', 'n', 'a' };`
 - O sistema insere os caracteres entre chaves, a partir da posição 0. Se o tamanho do vetor for superior ao número de caracteres nele armazenados, as posições não ocupadas serão preenchidas com zero, ou seja, nesse caso o arranjo terá um caractere terminador '\0', pois será preenchido com zeros a partir da terceira posição.
- Exemplo 3:
 - `char primeiro_nome[] = "Ana";`
 - O sistema determina o número de caracteres entre as aspas duplas, soma um para o caractere terminador, e cria uma *string* com o tamanho igual a tamanho da string + 1.

Inicialização de strings

Por exemplo:

Pra inicializar uma string que armazene “Aula C”, qual deve ser o tamanho dessa string?

```
char nome[7];
```

No caso de strings variáveis, é responsabilidade do programador reservar esse espaço adicional. Lembre-se de que o compilador C não faz consistência da indexação de vetores e, portanto, o dimensionamento inadequado de strings não é detectado pelo compilador.

Leitura

- **Leitura**

- Formato: **%s, mas sem o & antes do nome da variável.** O *scanf* encerrará a leitura do *string* assim que um branco for encontrado na *string* de entrada, ou um caractere de fim de linha ou de tabulação.

Ex.:

```
char nome_cliente[20];
scanf ("%s", nome_cliente);
```

Porém, se o usuário digitar
“Ana Paula” apenas “Ana”
será armazenado!

Leitura de strings com *gets*

- **Leitura**

- O *gets* permite colocar na variável todos os caracteres introduzidos, sem estar limitado a uma palavra.

Gets vem de *get string*

```
char n[20];  
gets(n);
```

A chamada **gets(n)** lê uma string e a armazena no vetor n.
O <enter>, digitado para finalizar a entrada, é automaticamente substituído por '\0'.

Escrita

- **Escrita**
 - O formato usado também é o **%s**.

Ex.:

```
char nome[20];  
//leitura  
printf("O nome do usuario é%s\n", nome);
```

Escrita de string com *puts*

- **Escrita**

- *puts()* de **put string**, e sua sintaxe é idêntica a da *gets()*:
- `puts(nome_da_string);`

Ex.:

```
char nome[ ]="Priscila Delabetha";
puts(nome);
```

Exemplo:

```
#include<stdio.h>
int main(void) {
    char nome[31], sobrenome[31], nascimento[11];
    printf("Nome: ");
    gets(nome);

    printf("Sobrenome: ");
    gets(sobrenome);

    printf("Data de nascimento: ");
    gets(nascimento);

    printf("\nNome completo: %s %s\n", nome, sobrenome);
    printf("Data de nascimento: "); puts(nascimento);
}
```

Outra função de leitura para strings

- Uma solução para o problema de tamanho de string é a função ***fgets()***,
- Ela recebe três dados: a string que vai armazenar o que vai ser digitado (no nosso caso é a variável "str"), o tamanho da string e de onde vai ler (ela pode ler de um arquivo de texto, por exemplo).
- Como vamos usar leitura do teclado, usamos ***stdin***.

Exemplo:

```
char n[30];  
fgets(n, 30, stdin);
```

Atenção ao “s” ou ‘s’

- As strings são representadas entre **aspas duplas** e os caracteres entre **apóstrofos**.
- Por definição, toda string tem o caractere terminador ‘\0’ ao final.
- Assim, “A” e ‘A’ NÃO são a mesma coisa!!
 - “A” : **vetor de 2 caracteres**-‘A’ e ‘\0’.
 - ‘A’ : **um único caractere**.
- Uma string é sempre um vetor de caracteres (com ‘\0’ ao final), mas um vetor de caracteres nem sempre é uma string !

Cópia de vetores

Strings são vetores, **não se pode** atribuir um vetor a outro de forma direta, precisamos copiar célula a célula

```
string1 = string2
```

WRONG!

Funções para manipular Strings!

Para facilitar o uso e evitar trabalho e estruturas de código na execução de comparações, atribuições...

Utilizamos a biblioteca `#include <string.h>`

Algumas de suas funções são:

- `strcpy`
- `strcat`
- `strlen`
- `strcmp`
- `strchr`

Função strcpy

Copia string_origem para string_destino.

Formato:

```
strcpy(string_destino, string_origem);
```

Exemplo:

```
#include <string.h>
int main( ) {
    char string_origem[10], string_destino[10];
    printf("Forneca um nome: ");
    gets(string_origem);
    strcpy(string_destino, string_origem);
    printf("\n%s\n", string_destino );
    return 0;
}
```

Qual é a saída desse programa?

Função strcat

A string_origem, é **anexada** ao final da string_destino.

Formato:

```
strcat(string_destino, string_origem);
```

Exemplo:

```
#include <string.h>

int main( ) {
    char string_origem[20], string_destino[40];
    printf("Forneca um texto: ");
    gets(string_origem);
    strcpy(string_destino, "O texto digitado foi: ");
    strcat(string_destino, string_origem);
    printf("\n%s\n", string_destino );
    return 0;
}
```

Qual é a saída desse programa?

Função strlen

Retorna o **tamanho** de uma string, sem contar o '\0'.

Formato:

```
strlen(string);
```

Exemplo:

```
#include <string.h>

int main( ){
    char string_primeiro[40], string_segundo[40];
    printf("Forneca um texto: ");
    gets(string_primeiro);
    printf("Forneca um texto: ");
    gets(string_segundo);
    printf("\n%s tem comprimento %d\n", string_primeiro, strlen(string_primeiro));
    printf("\n%s tem comprimento %d\n", string_segundo, strlen(string_segundo));
    return 0;
}
```

Qual é a saída desse programa?

Função strcmp

Compara duas strings, s1 e s2, caractere a caractere, com base na posição dos caracteres na tabela ASCII.
Se s1 e s2 forem iguais, retorna zero.

Se s1 for **maior** que s2, retorna um valor **maior que zero**.

Se s1 for **menor** que s2, retorna um valor **menor que zero**.

Formato:

```
strcmp(s1, s2);
```

Exemplo:

```
#include <string.h>
int main( ) {
    int i;
    char string_primeiro[40], string_segundo[40];
    for (i = 1; i <=3; i++) {
        printf("Forneca um texto: ");
        gets(string_primeiro);
        printf("Forneca um texto: ");
        gets(string_segundo);
        printf("Resultado da comparacao de %s com %s: %d", string_primeiro,
string_segundo,
        strcmp(string_primeiro, string_segundo) );
    }
}
```

Qual é a saída desse
programa?