

# Aula 05 - Introdução à Linguag C Programação: Estr. Dados Strings

OPL e IC1

Prof: Anilton Joaquim da Silva

[Anilton.ufu@outlook.com](mailto:Anilton.ufu@outlook.com)

# Caracteres

- Além dos tipos de dados numéricos com os quais temos trabalhado até agora, outro tipo de dado é muito importante no desenvolvimento de programas de computador, o tipo caractere. Estes tipos são a base para representação de informação textual como, por exemplo, a frase "eu amo programar em C".
- Variáveis com caracteres, em C, são declarados com sendo do tipo **char**, e sua leitura e escrita ocorre como qualquer outro tipo de dado. Por Exemplo:

```
#include <stdio.h>
int main()
{
    char letra;
    printf("digite uma letra qualquer seguida de enter: ");
    scanf("%c", &letra);
    printf("voce digitou %c\n" , letra);
    return 0;
}
```

- Se você pretende atribuir um caracter diretamente a uma variável, é importante se atentar à seguinte notação: caracteres são sempre escritos entre aspas simples. Por exemplo, 'a', '3' ou '\*'.

# Strings

- No C uma string é um vetor de caracteres terminado com um caractere nulo. O caracter nulo é um caractere com valor inteiro igual a zero (código ASCII igual a 0). O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo '\0'.
- Forma Geral:  
**char nome\_da\_string [tamanho\_string];**
- Isto declara um vetor de caracteres (uma string) com número de posições igual a *tamanho*. Note que, como temos que reservar um caractere para ser o terminador nulo, temos que declarar o comprimento da string como sendo, no mínimo, um caractere maior que a maior string que pretendemos armazenar. Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra João nela. Teremos:

J	o	ã	o	\0	...	...
---	---	---	---	----	-----	-----

- As duas células não usadas têm valores indeterminados. Isto acontece porque o C *não* inicializa variáveis, cabendo ao programador esta tarefa. Portanto as únicas células que são inicializadas são as que contêm os caracteres 'J', 'o', 'ã', 'o' e '\0' .

# Exemplo – Vetores de Caracteres

- Vetores podem conter dados de quaisquer tipos. Isto é, você pode declarar vetores de números reais ou inteiros, booleanos, e até tipos definidos por você, uma vez que aprenda como definir novos tipos. Um outro tipo interessante é o caractere, ou simplesmente char. Por exemplo, vamos definir um programa que defina uma string STR sendo inicializada com “Programacao”, leia um caracter do teclado, e lê um vetor de 10 caracteres. Todos os valores são impressos na tela.
- Cada caracter será lido com o **scanf(“%c” , ...)**, digite o nome e pressione o <enter>. (%c - lê todo caracter inclusive enter e espaço)

```
#include <stdio.h>
#include <stdio.h>
#define tam 10
int main()
{
    char nome[tam], str[tam] = "Programacao", L;

    printf ("digite uma letra: ");
    scanf ("%c", &L);
    fflush(stdin); //limpar o enter do L (buffer)
    printf ("Digite o seu nome: ");
    for (int i=0; i<tam; i++)
    {
        scanf ("%c", &nome[i]);
    }
    printf ("\nO nome digitado foi: ");
    for (int i=0; i<tam; i++)
    {
        printf ("%c", nome [i]);
    }
    printf ("\nA letra digitada foi: %c", L);
    printf ("\nO string STR inicializada foi: %s", str);
    printf ("\n\n");
    return 0;
}
```

# Exemplo

- Alteremos o programa para que leia até 100 caracteres, mas que pare de lê-los tão logo um “.” seja digitado. Para representar um caractere em C, use aspas simples, isto é, ' .' .
- Neste exemplo usamos não a função **scanf** e sim a função **getche()** da biblioteca **conio.h**. Esta função lê um caracter do teclado sem a necessidade de pressionar a tecla <enter> em cada um dos caracteres.

```
//lê um conjunto de no máximo 100 caracteres
//terminando a leitura com ponto '.'

#include <stdio.h>
#include <conio.h>
#define tam 100

int main()
{
    char str[tam];
    int i = 0;
    printf ("Digite uma frase ('.' para terminar): ");
    do
    {
        str[i] = getche();
        i++;
    }while (i<tam && str[i-1]!='.');

    printf ("\n\nA frase digitada foi: ");
    for (i=0; (i<tam && str[i]!='.); i++)
    {
        printf ("%c", str[i]);
    }
    return 0;
}
```

# Tabela ASCII

- Caracteres são, na verdade, números disfarçados e seguem uma codificação específica. Uma pessoa pode decidir que o 'a' será o 1, o 'b' será o 2 e assim por diante. Mas como outra pessoa que receber a informação saberá disso? Para evitar este problema a representação de caracteres como números foi padronizada. Os principais padrões existentes são:
  - **ASCII** - American Standard Code for Information Interchange
  - **EBCDIC** - Extended Binary Coded Decimal Interchange Code
  - **UNICODE**.

# Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	j
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

# Lendo e imprimindo Strings

- Declarando uma string:  
`char nome[30];` → vetor de 30 caracteres
- Lendo uma string:  
`scanf ("%s", nome);`  
→ lê somente uma única palavra  
`gets (nome);`  
→ lê uma frase com até 30 caracteres
- Imprimindo uma string:  
`printf ("%s", nome); ou`  
`puts (nome);`  
→ imprime nome: palavra ou frase

OBS: no exemplo

- `#include <stdio.h>`
- `int main()`
- `{`
- `char nome[30];`
- `int nota1,nota2,nota3;`
- `float nota_final;`

- `printf("digite o nome do aluno: ");`
- `gets(nome);`
- `printf("digite a primei. nota: ");`
- `scanf("%d", &nota1);`
- `printf("digite a segunda nota: ");`
- `scanf("%d", &nota2);`
- `printf("digite a tercei. nota: ");`
- `scanf("%d", &nota3);`
- `nota_final = (nota1 + nota2 + nota3)/3.0;`
- `printf("nome aluno: %s\n", nome);`
- `printf("nota final: %f \n", nota_final);`
- `if (nota_final < 6)`
- `{ printf("aluno: reprovado \n"); }`
- `else`
- `{ printf("aluno: aprovado \n"); }`
- `return 0;`
- `}`

# Funções que manipulam Strings

- Quando trabalhamos com strings é muito comum a realização de algumas tarefas como descobrir o tamanho da palavra digitada pelo usuário, comparar duas palavras para saber a ordem, ou ainda, concatenar duas palavras em uma única. Para isso, a biblioteca string.h fornece algumas funções prontas.

Função	Descrição
strlen	retorna o tamanho (em caracteres) da palavra passada como argumento.
strcpy	copia o conteúdo da segunda <i>string</i> para a primeira.
strcat	concatena o texto da segunda <i>string</i> na primeira.
strcmp	compara duas <i>strings</i> (vide exemplo a seguir).
stricmp	compara duas <i>strings</i> sem diferenciar maiúsculas e minúsculas.
atoi	converte uma <i>string</i> para o inteiro correspondente.
atof	converte uma <i>string</i> para o número real correspondente.

# Funções que manipulam Strings

- Função **Strlen()**:

Sua forma geral é:

***strlen (string);***

- A função **strlen()** retorna o comprimento da string fornecida. O terminador nulo não é contado. Isto quer dizer que, de fato, o comprimento do vetor da string deve ser um a mais que o inteiro retornado por **strlen()**.

# Funções que manipulam Strings

- Função **strcpy()**:

Sua forma geral é:

*strcpy (string\_destino, string\_origem);*

- A função **strcpy()** copia integralmente a string-origem para a string- destino.  
Obrigatoriamente as duas strings devem possuir o mesmo tamanho senão erros poderão ocorrer.

# Funções que manipulam Strings

- Função **strcat()**:

Sua forma geral:

*strcat (string\_destino,string\_origem);*

- A função concatena (junta) a string de origem com a string destino. A string de origem permanecerá inalterada. É importante observar que a string de destino deverá ser definida com o tamanho capaz de receber as duas string, caso contrario erros poderão ocorrer.

# Funções que manipulam Strings

- Função **strcmp()**:

Sua forma geral é:

***strcmp (string1,string2);***

- A função **strcmp()** compara a string1 com a string2.

Se as duas forem idênticas a função retorna zero.

Se elas forem diferentes a função retorna não-zero.

Valores de retorno:

- string1 = string2 → retorna valor zero (0)
- string1 > string2 → retorna valor inteiro positivo
- string1 < string2 → retorna valor inteiro negativo

# Exemplo: trata funções de strings

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

int main()
{
    char str1[50], str2[50];
    int i;
    float f;
    printf ("Entre primeiro nome: ");
    gets(str1);
    printf ("Entre ultimo nome: ");
    gets(str2);
    strcat(str1, " "); //junto espaco com str1
    strcat(str1, str2);
    printf ("Seu nome completo e: %s\n", str1);
    printf ("Ele possui %d caracteres.\n",
           strlen(str1));
```

```
printf ("Entre outro nome: ");
gets(str2);
//comparacao de strings
if(strcmp(str1, str2) == 0)
{
    printf ("os dois nomes sao iguais.\n");
}
else if(strcmp(str1, str2) < 0)
{
    printf ("%s vem antes de %s\n", str1, str2);
}
else
{
    printf ("%s vem antes de %s\n", str2, str1);
}

return 0;
```

# Exercícios

1. Faça um programa que leia uma string e troque todas as ocorrências de uma letra L<sub>1</sub> pela letra L<sub>2</sub> em uma string. A string e as letras L<sub>1</sub> e L<sub>2</sub> devem ser lidas pelo teclado.
2. Faça um programa que leia 3 strings e as imprima em ordem alfabetica.
3. Faça um programa que leia uma string qualquer de tamanho N e imprima esta string com todos os caracteres em maiusculo.

# Matrizes de Caracteres

- Da mesma forma que vetores de caracteres podem ser manipuladas de forma especial no C, também podem as matrizes de caractere. Na verdade, se um vetor de caracteres é o que chama-se de uma string, então uma matriz bidimensional de caracteres é, na verdade, um vetor de strings. Por exemplo,

```
char nomes[10][20];
```

- Esta definição pode ser vista como um vetor de 10 strings, cada uma com 20 caracteres, e cada string pode ser manipulada como tal.

# Matrizes de Caracteres

- Matrizes de caractecres são matrizes bidimensionais. Imagine uma string. Ela é um vetor. Se fizermos um vetor de strings estaremos fazendo uma lista de vetores. Esta estrutura é uma matriz bidimensional do tipo char.
- Forma Geral:  
*char nome\_variável [num\_strings][tamanho\_strings];*
- Como acessar uma string individual? Bastas usar apenas o primeiro índice da matriz.  
*nome\_da\_variável [índice];*

# Exemplo: lê e imprime M nomes

```
• #include<string.h>
• #include <stdio.h>
• #define M 10
• #define N 11
• int main()
• {
•     char mat[M][N];
•     int i;
•     printf("Digite %d palavras de no maximo %d caracteres \n", M, N-1);
•     for(i = 0; i < M; i++)
•     {
•         printf("%d: ", i);
•         gets(mat[i]);
•     }
•     printf("\nAs palavras lidas foram as seguintes \n");
•     for(i = 0; i < M; i++)
•     {
•         printf("%s\t(com tamanho = %d) \n", mat[i], strlen(mat[i]));
•     }
•     return 0;
• }
```

# Inicializando Matrizes de Caracteres

- Podemos inicializar matrizes de caracteres, assim como podemos **inicializar variáveis**. A forma geral de uma matriz com inicialização é:  
*char nome\_var [tam1][tam2] = {lista\_de\_valores};*
- A lista de valores é composta por valores (do mesmo tipo da variável) separados por vírgula. Os valores devem ser dados na ordem em que serão colocados na matriz. Exemplos:

```
char str [10] = { 'J', 'o', 'a', 'o', '\0' };
```

```
char str [10] = "Joao";
```

```
char str_vet [3][10] = { "Joao", "Maria", "Jose" };
```

```
char dias_Semana[7][10] = {"Segunda", "Terça", "Quarta",  
"Quinta", "Sexta", "Sabado", "Domingo"};
```

# Exemplo: Lista alunos que tiraram media.

```
• //lê uma lista de alunos e suas notas e mostra
• // os nomes dos alunos acima da média
• #include <stdio.h>
• #include<string.h>
• #define qtd 100
• #define tam 30
• int main()
{
    char nomes[qtd][tam];
    float notas[qtd], soma, media;
    int N, i;
    printf("Lista alunos acima da media. \n");
    printf("Digite numero de alunos: ");
    scanf("%d", &N);
    //le os dados de N aluno (nome e nota)
    for (i=0; i<N; i++)
    {
        fflush(stdin); //limpar buffer do teclado -
        <enter> do valor de n/notas
        printf("Nome aluno %d: ", i+1);
        gets(nomes[i]);
        printf("Nota aluno %d: ", i+1);
        scanf("%f", &notas[i]);
    }
    //calcula_media
    float soma = 0;
    for (i=0; i<N; i++)
    {
        soma = soma + notas[i];
    }
    media = soma/N;
    printf("A media das notas eh: %f \n",
media);

    //mostra_acima_media
    printf("Alunos que tiraram media: \n");
    for (i=0; i<N; i++)
    {
        if (notas[i] > media)
            printf("%s \n", nomes[i]);
    }
    return 0;
}
```

# Exercícios

1. Fazer um programa que leia uma lista de N nomes de pessoas de até 10 caracteres e imprima esta mesma lista de nomes todos em letras maiúsculas.
2. Fazer um programa que leia uma lista de N nomes de pessoas de até 10 caracteres, calcule e imprima a quantidade de homens, mulheres e indefinidos, da seguinte forma:
  - Nome terminado com letra ‘o’ – masculino;
  - Nome terminado com letra ‘a’ – feminino;
  - Nomes terminado com outras letras – indefinido.