Insper

Mutirão C - vetores, matrizes e strings

Igor Montagner

20-02-2018

Na última aula trabalhamos com conceitos básicos de *C* e com tipos de dados simples (inteiros e números fracionários). Hoje veremos as primeiras estruturas complexas em *C*: vetores, matrizes e strings.

Introdução

Criamos vetores **de tamanho fixo** em *C* usando a seguinte sintaxe:

```
long vetor[100];
```

Basta adicionar `[]` após o tipo e teremos um vetor de tamanho constante (i.e. não dependente da entrada do usuário). **Não é possível redimensionar o vetor**. Depois de declarado ele terá para sempre o mesmo tamanho. O acesso a elementos também é bastante simples:

```
printf("%ld\n", vetor[0]);
```

A linha acima imprime o primeira valor do vetor. O uso com `scanf` segue a mesma lógica dos tipos simples:

```
scanf("%ld", &vetor[0]);
```

Tarefa 1: O código abaixo compila? Se sim, qual é sua saída?

```
#include <stdio.h>

int main(int argc, char *argv[]) {
        long vetor[5];
        int i;

        for(i = 0; i <= 5; i++) {
                if (vetor[i] % 2 == 0) {
                        printf("Par! ");
                }
                printf("i: %d, vetor[i]: %ld\n", i, vetor[i]);
        }

        return 0;
}
```

Tarefa 2: O código acima está no arquivo *exemplos/erro_comum1.c*. Compile e execute ele. Os resultados foram os esperados? Se não, você consegue explicar o por que eles foram diferentes?

Como podemos ver, o código compila, mas seu comportamento é indefinido por duas razões:

1. O vetor não é inicializado com 0 quando é criado.
2. A posição `vetor[5]` não é válida e pode resultar na leitura de dados inválidos da memória.

> **!** Diferentemente de outras linguagens, *C* não verifica os índices automaticamente nem inicializa o elementos do vetor com 0. Além disto, não é possível obter o tamanho de um vetor a partir de seu nome.

Um ponto importante é que o sistema de tipos não permite a conversão/passagem de vetores do mesmo tipo mas de tamanhos diferentes. Assim, a função abaixo não aceitaria como argumento a variável `double arr[4]`, pois só aceita vetores de tamanho 3.

```c
double soma(double arr[3]) {
    double s = 0;
    for (int i = 0; i < 3; i++) {
        s += arr[i];
    }
    return s;
}
```

Para fazê-la aceitar vetores de tamanho qualquer poderíamos mudar sua assinatura para não checar o tamanho do vetor e passá-lo como argumento da função.

```c
double soma(double arr[], int n);
```

Tarefa 3: Modifique a função `soma` acima para aceitar vetores de qualquer tamanho. Lembre-se que arrays em *C* não conhecem seu tamanho, então é de sua responsabilidade acessar somente elementos válidos e checar se o vetor não está vazio (ou seja, `n < 1`)

Tarefa 4: Vamos agora juntar vetores com a aula passada e com a leitura de ontem: escreva do zero um programa que

1. Leia um inteiro `n` do terminal (número de elementos do vetor)
2. Leia `n` números fracionários e guardá-los em um array.
3. Chame uma função para calcular a média do vetor.
4. Imprima a média calculada.

Seu programa deverá calcular a média usando uma função `avg` escrita por você mesmo e pode supor que `n < 100`. Para facilitar seus testes, escreva dois arquivos de entrada e use `<` para rodar o programa.

Tarefa 5: Modifique seu programa acima para que ele imprima também a variância do vetor.

Strings

Como visto na expositiva, strings são vetores de caracteres sendo que o último elemento da string contém um caractere `'\0'`. Logo, uma string declarada como `char str[100]` pode guardar strings de **até 99** caracteres (mais 1 para o `'\0'`). Se a string tiver comprimento menor o restante das posições simplesmente não é utilizado.

Figure 1: String é um vetor de `char` terminado em `'\0'`.

Tarefa 6: Em Python não existe diferença entre "a" e `'a'`. Isto é verdade em *C*?

Tarefa 7: Um aluno fez a seguinte função para cópia de strings.

```
void copia_string(char str1[], char str2[]) {
    int i = 0;
    while (str1[i] != '\0') {
        str2[i] = str1[i];
        i++;
    }
}
```

Existem pelo menos dois problemas graves neste código. Você consegue identificá-los?

Tarefa 8: Escreva abaixo uma função que recebe uma string como parâmetro e retorne quantas vezes o caractere `1` aparece.

Tarefa 9: Escreva abaixo uma função que recebe uma string como parâmetro e retorna 1 se ela é um palíndromo ou 0 caso contrário.

> **!** Valide suas soluções com um professor (ou um colega já validado) antes de prosseguir

Vamos agora partir para leitura e escrita de strings no terminal. Para imprimir uma string no terminal basta usar o código `%s` na string passada para o printf:

```
char str[10] = "world!";
printf("hello %s\n", str);
```

Já a leitura de strings é feita usando a função *fgets*, que lê uma linha inteira de caracteres a partir de um arquivo ou do terminal. Veja o exemplo abaixo. O último argumento `stdin` representa o terminal.

```
char str[10];
fgets(str, 10, stdin); // precisamos passar o tamanho máximo + 1 da nossa string.
```

Tarefa 8: Faça, do zero, um programa que leia uma string (tamanho máximo 200) e crie uma nova string trocando toda letra por maiúsculas. Seu programa deverá imprimir a string original e sua versão em maiúsculas. Seu programa deve funcionar para strings contendo números, símbolos, espaços e letras maiúsculas e minúsculas. Consulte a tabela abaixo, se necessário.

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

asciichars.com

Figure 2: Tabela ASCII com valores em decimal e hexa

Matrizes

Vamos agora retomar os exercícios de imagens e trabalhar com matrizes. Como visto na expositiva, matrizes em *C* nada mais são que vetores colocados um após o outro.

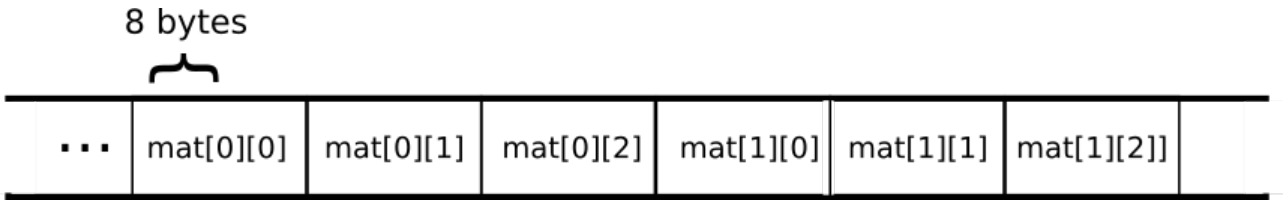

Figure 3: Representação de matriz de inteiros na memória

Seu acesso é feito com a seguinte notação

```
int matriz[10][20];

matriz[0][3] /* acessa linha 0, coluna 3 */

scanf("%d", &matriz[1][5]); // armazena inteiro digitado na posição 1,5 */
```

O formato de imagens mais simples existente é o `pgm`, que reprenta uma imagem em níveis de cinza como uma matriz com valores entre 0 (para preto) e 255 (para branco). Seu formato é o seguinte.

```
P5
W H
255
.........
```

Ou seja, primeiro lemos uma linha com a string "P5", depois dois inteiros `w` e `h` representando a largura e a altura da imagem, depois o valor 255. Então lemos `w * h` valores representando os pixels da imagem. Disponibilizamos várias imagens de exemplo na pasta *exemplos*. Vocês podem abrí-las com qualquer editor de texto para ver seu conteúdo.

> **!** Nas tarefas abaixo estamos supondo que você usa `<` para passar o conteúdo das imagens exemplo para seu programa no terminal e `>` para salvar o resultado do terminal em uma nova imagem *pgm*.

Para as tarefas abaixo você pode supor que as imagens tem tamanho máximo 512×512. Para deixar seu código mais limpo, defina duas constantes `MAXW` e `MAXH` para guardar estes valores.

Tarefa 9: Crie, do zero, um programa que lê o cabeçalho de uma imagem *pgm* (primeiras três linhas) passada no terminal e imprima as dimensões da imagem. Não se esqueça de ler também o número `255` na terceira linha.

Tarefa 10: Crie uma função `void le_imagem(int mat[MAXW][MAXH], int w, int h)` que lê os valores da matriz da imagem e os escreve em `mat`.

Tarefa 11: Crie uma função `void escreve_imagem(int max[MAXW][MAXH], int w, int h)` que escreve o cabeçalho e todos os pixels de uma imagem no terminal seguindo o formato *pgm* descrito acima.

Tarefa 12: Verifique que seu programa está correto fazendo uma função `main` que simplesmente lê uma imagem e logo em seguida a escreve no terminal. Verifique visualmente que a imagem de saída é igual a original.

Tarefa 13: Finalmente, crie uma função `void limiar(int max[MAXW][MAXH], int w, int h, int lim)` que aplica um limiar de 127 na imagem e chame-a na sua função `main`. Verifique visualmente que a imagem de saída é a esperada.