Insper

Leitura 2 - vetores unidimensionais, matrizes e strings

Igor Rafael Hashimoto

20-02-2018

Vetores unidimensionais

A linguagem *C* permite que declaremos vetores de tamanho fixo usando uma sintaxe muito simples:

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

Basta adicionar `[]` após o tipo e teremos um vetor de tamanho constante (i.e. não dependente da entrada do usuário). O acesso a elementos também é bastante simples:

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

imprime a primeira posição do vetor. Diferentemente de outras linguagens, *C* não verifica os índices automaticamente nem inicializa o elementos do vetor com 0. Podemos verificar isto compilando e rodando o arquivo *exemplo3/erro_comum1.c*.

```
#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;
}
```

gcc -Wall -pedantic -std=c99 -Og erro_comum1.c -o erro_comum1
./erro_comum1

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.

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

```
double length(double arr[3]);
```

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.

```
double lenght(double arr[], int n);
```

Lembre-se que arrays em *C* não conhecem seu tamanho, então é de sua responsabilidade acessar somente elementos válidos.

Exercícios:

1. Implemente a função `media` em *exemplo3/vetor.c*.

Strings

Como já deve ser conhecimento comum, caracteres são representados por números segundo uma codificação. A mais simples dela é a ASCII, que representa até 127 caracteres usando o tipo `char`.

Uma string em *C* é um vetor de caracteres terminado pelo caractere `\0`. Esta convenção foi adotada pois em *C* vetores não possuem embutido seu tamanho. Assim, para descobrir o tamanho de uma string percorremos todos os caracteres até encontrar o `\0`. Veja abaixo uma função que imprime uma string.

```
void print_str(char str[]) {
    int i;
    for (i = 0; str[i] != '\0'; i++) {
        printf("%c", str[i]);
    }
    printf("\n");
}
```

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 1: Tabela ASCII com valores em decimal e hexa

Exercícios:

1. Pesquise como usar *printf* com strings.
2. Pesquise como usar a função *fgets* para ler strings da linha de comando.
3. Crie uma função que imprima uma string ao contrário.
4. Bônus: utilize recursão para a tarefa acima.

Matrizes

Podemos criar matrizes de tamanho fixo usando a seguinte notação

```
long matriz[100][200];
```

O acesso é feito usando a seguinte notação:

```
matriz[i][j] = 10;
```

onde `i` é a linha e `j` a coluna a ser acessada. Diferentemente de vetores unidimensionais e strings, para usar uma matriz é necessário conhecer sua última dimensão. Isto significa que ao passar matrizes como argumento para funções elas precisam ter uma dimensão específica. Veremos com mais detalhes por que isso ocorre em Sistemas de Hardware e Software.

Veja abaixo um exemplo que calcula a soma de uma matriz tamanho 3 por 3.

```
long soma(long matriz[3][3]) {
    long s = 0;
    int i, j;
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            s += matriz[i][j];
        }
    }
    return s;

}
```

Este tipo de construção é pouco usado em *C* pois somente é útil em casos que o tamanho da matriz NÃO depende da entrada do usuário. Exemplos comuns são vetores e matrizes de rotação/translação para geometria 2D/3D e áreas de memória usadas em displays para sistemas embarcados.

Exercício: faça um programa que:

1. leia uma matriz 5x5 de double da entrada padrão, some 1 em todos os elementos e imprima o resultado. Seu programa deve funcionar com a matriz abaixo

```
1 2 3 4 5
5 4 3 2 1
4 5 6 7 8
0 9 8 7 6
4 6 8 0 1
```