

21. Arranjos

DIM0321

2015.1

Outline

① Ponteiros e arranjos

② Strings / Cadeias de caracteres

DIM0321

21. Arranjos

2015.1 1 / 26

DIM0321

21. Arranjos

2015.1 2 / 26

① Ponteiros e arranjos

② Strings / Cadeias de caracteres

Equivalências

Notação

- $a[i] \hat{=} *(a + i)$

Diferença

- Um ponteiro é uma variável

```
1 int *a, *b;  
2 a++; // OK  
3 a = b; // OK
```

- Um nome de arranjo **não** é uma variável

```
1 int a[];  
2 a++; // KO  
3 a = 0; // KO
```

DIM0321

21. Arranjos

2015.1 3 / 26

DIM0321

21. Arranjos

2015.1 4 / 26

Por que usar arranjos ?

Observação

Não é viável utilizar variáveis simples quando um programa precisa armazenar e processar:

- uma quantidade de dados variável
- muitos dados **do mesmo tipo**

- Arranjos armazenam um número fixo de elementos do mesmo tipo
- O tamanho do arranjo pode ser decidido no tempo de execução em função de outros dados.
- A capacidade é definida quando o arranjo é criado
- O tipo dos elementos é definido no código fonte
- Cada elemento pode ser lido ou escrito individualmente

Exemplo

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     char name[4];
6     name[0] = 'v';
7     name[1] = 64;
8     name[2] = 'u';
9     name[3] = 'a';
10    printf("%s\n", name);
11 }
```

Elementos

Organização na memória

- Elementos são armazenados de forma sequencial na memória
- As posições são identificadas por um número inteiro
- O identificador da posição é chamado de índice

Índices válidos

- Os índices válidos de um arranjo a vão de 0 até $n - 1$.
- Eles são um deslocamento/shift do endereço de base do arranjo.

Declaração de um arranjo

```
1 <tipo> nome[N];
2 <tipo> nome[N] = {elemento1, ..., elementoN};
```

Observações

- ' <tipo> ' = tipo dos elementos do arranjo
- ' nome ' = identificador do arranjo
- ' N ' = capacidade do arranjo

Declaração e efeito

A declaração

```
1| int a[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

define um arranjo de tamanho 10, i.e. um bloco de 10 objetos consecutivos do tipo `int` com nomes `a[0]`, `a[1]`, ..., `a[9]`

a:	0	1	2	3	4	5	6	7	8	9
	a[0]	a[1]								a[9]

- `a[i]` é o *i*-ésimo elemento do arranjo (começando com 0)

Exemplo : Acesso & atribuição

```
1| #include <stdio.h>
2|
3| int f(int m) {
4|     if (m == 4) return 13;
5|     return 0;
6| };
7|
8| int main(void) {
9|     float vetor[100], notas[30];
10|    int numeros[15];
11|    int posicao = 7 + 4;
12|    float valor = 3.2f;
13|    vetor[posicao] = valor;
14|    numeros[f(4)] = 7;
15|    numeros[10] = numeros[13] + 1;
16|    notas[16] = 10.f;
17|    notas[25] = (6.f + 7.f + 8.f) / 3;
18|    scanf("%i %f", &numeros[5], &notas[15]);
19|    printf("%i %f", numeros[10], notas[25]);
20|    return 0;
21| }
```

Exemplo

```
1| #include <stdio.h>
2|
3| int main(void)
4| {
5|     float notas[3];
6|     int n, i;
7|     printf("Número de alunos: ");
8|     scanf("%i", &n);
9|     for (i = 0; i < n; i++) {
10|         printf("Nota do aluno %i: ", i + 1);
11|         scanf("%f", &notas[i]);
12|     }
13|     return 0;
14| }
```

Dia dos meses

```
1| #include <stdio.h>
2| #define MESES 12
3|
4| int main(void)
5| {
6|     int dias[MESES] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7|     int i;
8|     for (i = 0; i < MESES; i++) {
9|         printf("Mes %d tem %d dias\n", i + 1, dias[i]);
10|    }
11|    return 0;
12| }
```

Dia dos meses (sizeof)

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int dias[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
6     int i;
7     for(i = 0; i < sizeof(dias) / sizeof(int); i++) {
8         printf("Mes %d tem %d dias\n", i + 1, dias[i]);
9     }
10    return 0;
11 }
```

Inicialização de vetores

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int indefinido[4];
6     int i;
7     for (i = 0; i < 4; i++) {
8         printf("%2d%14d\n", i, indefinido[i]);
9     }
10    return 0;
11 }
```

Exemplo

- Como ler n valores e imprimir estes valores em ordem inversa?

Exemplo

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     unsigned n;
6     int i;
7     printf("Entre com um inteiro > 0: ");
8     scanf("%u", &n);
9
10    int v[n];
11    for (i = 0; i < n; i++) {
12        printf("v[%u] = ?", i);
13        scanf("%d", &v[i]);
14    }
15
16    for (i = n - 1; i >= 0; i--) {
17        printf("v[%u] = %d\n", i, v[i]);
18    }
19
20    for (i = 0; i < n; i++) {
21        printf("v[%u] = %d\n", n - i - 1, v[n - i - 1]);
22    }
23
24 }
```

Arranjos e ponteiros

- Um arranjo é um ponteiro.
- $v == \&v[0]$
- $v[i] == *(v + i)$
- $\&v[i] == v + i$
- $\text{char } s[]$ ou $\text{char } *s$ são equivalentes quando usado como parâmetros formais.
- A função pode usar o ponteiro como um arranjo também.
- Use sempre $\text{char } *s$ nos parâmetros de função
 - ▶ O uso como ponteiro fica explícito

Exemplo

```
1 | #include <stdio.h>
2 |
3 | void pp_data_before(int *data)
4 | {
5 |     printf("PP DATA: %d", data[-1]);
6 | }
7 |
8 | int main(void)
9 | {
10 |     int datas[31], *pti;
11 |     pti = datas; /* ou */ pti = &datas[0];
12 |
13 |     if (datas + 2 == &datas[2]) { printf("ok\n"); }
14 |     if (*(datas + 2) == datas[2]) { printf("ok\n"); }
15 |     datas[29] = 3;
16 |     pp_data_before(datas + 30); // ou &datas[30]
17 |     return 0;
18 | }
```

Atenção ! verificações não existem

A linguagem C não tem verificação de limites de arranjos

- Existe a possibilidade que um programa defina um vetor com apenas dez elementos, mas atribua valores até o vigésimo
- Neste caso, os dez valores adicionais estarão sobrescrevendo valores armazenados em outras posições de memória

Exemplo

```
1 | #include <stdio.h>
2 | int main(void)
3 | {
4 |     int n = 4;
5 |     int v[2] = { 0, 1 };
6 |     v[2] = 3;
7 |     printf("%p %p %p\n", &n, &v[0], &v[1]);
8 |     printf("%d\n", n);
9 |     return 0;
10 | }
```

① Ponteiros e arranjos

② Strings / Cadeias de caracteres

Introdução

Ideia central

- Um texto = arranjo de 'char'.

Exemplo

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i;
6     char str[5];
7     printf("Digite uma cadeia de caracteres: ");
8     scanf("%s", str);
9     printf("Voce digitou: %s\n", str);
10    printf("%c", str[0]);
11    printf("%c\n", str[1]);
12    for(i = 0; str[i] != '\0'; i++) {
13        printf("%c", str[i]);
14    }
15    return 0;
16 }
```

DIM0321

21. Arranjos

2015.1 21 / 26

Exemplo : strlen

Assunto

Escrever uma função para calcular o tamanho de um *string*.

DIM0321

21. Arranjos

2015.1 23 / 26

Observações

Sobre a biblioteca padrão

- A biblioteca padrão de C possui diversas sub-rotinas para manipular textos (usando arranjos de caracteres)
- Essas sub-rotinas devem ser usadas com **cuidado** pois elas fazem hipóteses que não são verificadas

Elementos

- O caractere especial '0' é usado para indicar o final do texto
- O usuário deve garantir que o espaço de memória necessário foi devidamente alocado.
- Trataremos detalhadamente deste assunto em uma outra aula

DIM0321

21. Arranjos

2015.1 22 / 26

Soluções

Versão 1

```
1 unsigned mystrlen(char *s)
2 {
3     unsigned n = 0;
4     for (n = 0; s[n] != '\0'; n++);
5     return n;
6 }
```

Versão 2

```
1 unsigned mystrlen2(char *s)
2 {
3     unsigned n = 0;
4     while (*s++) n++;
5     return n;
6 }
```

DIM0321

21. Arranjos

2015.1 24 / 26

Referências

Precisões

[KR88] 5

[Bac13] 6, 7, 10

-  André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.
-  Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.

Perguntas ?



<http://dimap.ufrn.br/~richard/dim0321>