



Programação 2

Aula 2

Thiago Cavalcante – thiago.cavalcante@penedo.ufal.br

30 de outubro de 2019

Universidade Federal de Alagoas – UFAL

Campus Arapiraca

Unidade de Ensino de Penedo

Estrutura de dados

- Linear
- Contígua
- Homogênea
- Estática

mútiplas variáveis × **um** array

Vetores e matrizes (Arrays)

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

int main() {
    float n1, n2, n3;
    printf("Digite a nota de 3 estudantes: ");
    scanf("%f",&n1);
    scanf("%f",&n2);
    scanf("%f",&n3);
    float media = (n1 + n2 + n3) / 3.0;
    if(n1 > media) printf("nota: %f\n", n1);
    if(n2 > media) printf("nota: %f\n", n2);
    if(n3 > media) printf("nota: %f\n", n3);
    return 0;
}
```

Vetores e matrizes (Arrays)

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

int main() {
    float n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12;
    printf("Digite a nota de 100 estudantes: ");
    scanf("%f",&n1);
    scanf("%f",&n2);
    scanf("%f",&n3);
    scanf("%f",&n4);
    scanf("%f",&n5);
    scanf("%f",&n6);
    scanf("%f",&n7);
    scanf("%f",&n8);
    scanf("%f",&n9);
    scanf("%f",&n10);
    scanf("%f",&n11);
    scanf("%f",&n12);
}
```

Declarando um array

```
tipo nome[tamanho];
```

```
float notas[100];
```

tamanho é um valor inteiro e não pode ser uma variável

Acessando um elemento

`nome[índice]`

$0 \leq \text{índice} \leq (\text{tamanho} - 1)$ ⚠

acesso funciona como uma variável

índice informa quantas posições pular na memória

Vetores e matrizes (Arrays)

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

int main() {
    int notas[100];
    int i;
    for (i = 0; i < 100; i++){
        printf("Digite a nota do aluno %d", i);
        scanf("%d", &notas[i]);
    }
    return 0;
}
```


Arrays multidimensionais

```
tipo nome[tam_1][tam_2]...[tam_n];
```

```
int arr[100][200][50];
```

2 dimensões \implies matriz

Acessando um elemento

`nome[ind_1][ind_2]...[ind_n]`

$$0 \leq \text{ind}_1 \leq (\text{tam}_1 - 1)$$

$$0 \leq \text{ind}_2 \leq (\text{tam}_2 - 1)$$

...

$$0 \leq \text{ind}_3 \leq (\text{tam}_n - 1)$$

arrays são **sempre** armazenados linearmente na memória

Inicializando um array

- Laço **for**
- Atribuição direta
- Operador chaves **{ }**
 - Com/sem tamanho
 - Total/parcial

não se pode atribuir um array a outro 

Vetores e matrizes (Arrays)

```
int i; /* LAÇO FOR */
int arr[3];
for(i = 0; i < 3; i++) {
    arr[i] = 2 * i;
}
```

```
int arr[3]; /* ATRIBUIÇÃO DIRETA */
arr[0] = 0;
arr[1] = 2;
arr[2] = 4;
```

```
int arr[3] = {0, 2, 4}; /* OPERADOR CHAVES */
int arr[] = {0, 2, 4}; // sem tamanho
int arr[3] = {0, 2};    // parcial
```

Vetores e matrizes (Arrays)

```
int i, k; /* LAÇO FOR */
int arr[2][2];
for(i = 0; i < 2; i++) {
    for(j = 0; j < 2; j++) {
        arr[i][j] = i + j;
    }
}
```

```
int arr[2][2]; /* ATRIBUIÇÃO DIRETA */
arr[0][0] = 0;
arr[0][1] = 1;
arr[1][0] = 1;
arr[1][1] = 2;
```

```
int arr[2][2] = {0, 1, 1, 2}; /* OPERADOR CHAVES */
int arr[2][2] = {{0, 1}, {1, 2}};
```

Exercícios

**Declarada como um array
comum do tipo char**

```
char nome[tamanho];
```


Principal diferença

caractere '`\0`' **obrigatório** para indicar o final da string

string de tamanho x armazena $x - 1$ caracteres

Inicializando uma string

```
char str[20] = {'U', 'F', 'A', 'L', '\0'};  
char str[20] = "UFAL";
```

não se pode atribuir uma string a outra ⚠

Arrays de caracteres (Strings)

```
/* COPIANDO UMA STRING */

#include <stdio.h>
#include <stdlib.h>

int main() {
    int i;
    char str1[20] = "Hello World";
    char str2[20];
    for (i = 0; str1[i] != '\0'; i++)
        str2[i] = str1[i];
    str2[i] = '\0';
    return 0;
}
```

Lendo uma string do teclado

```
char string[20];  
  
scanf("%s", string);  
gets(string);  
fgets(string, 20, stdin);
```

Arrays de caracteres (Strings)

- 👎 scanf: desconsidera espaços
- 👎 gets: não evita estouro do *buffer*
- 👍 fgets: mais recomendada, não desconsidera espaços, impede o estouro do *buffer* e leva em conta o '`\n`'

Escrevendo uma string na tela

```
char string[20] = "UFAL";
```

```
printf("%s", string);  
fputs(string, stdout);
```

Funções para manipulação de strings

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

Arrays de caracteres (Strings)

- `strlen(str)`
 - retorna tamanho da string
 - não considera o `'\0'`
- `strcpy(dest, orig)`
 - copia uma string em outra ⚠
- `strcat(dest, orig)`
 - insere uma string no final de outra ⚠
- `strcmp(str1, str2)`
 - checa se duas strings são iguais
 - *case-sensitive*
 - 0: iguais / 1: diferentes

Exercícios

Tipos básicos: `char`, `int`, `float`,
`double`, `void`

Tipos compostos: `array`

Comandos usados para definir novos tipos de dados

- `struct` (estruturas/registros)
- `union` (uniões)
- `enum` (enumerações)
- `typedef`

Declarando uma estrutura

```
struct nome {          | struct cadastro {  
    tipo1 campo1;      |     char nome[50];  
    tipo2 campo2;      |     int  idade;  
    ...                |     char rua[100];  
    tipoN campoN;      |     int  numero;  
};                      | };
```

variável que agrupa variáveis

estruturas diferentes podem ter campos iguais

Declarando uma variável do tipo da estrutura

```
struct nome {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
} variavel1, variavel2;  
  
struct nome variavel3, variavel4;
```

Acessando os campos de uma estrutura (operador ponto ' . ')

```
struct cadastro c;  
  
strcpy(c.nome, "Thiago");  
c.idade = 473;  
strcpy(c.rua, "Rua Principal");  
c.numero = 1082;
```

cada campo pode ser visto como uma variável comum

Inicialização de estruturas

```
/* TOTAL */  
struct cadastro c = {  
    "Thiago",  
    473,  
    "Rua Principal",  
    1082  
};  
/* PARCIAL */  
struct cadastro c = {"Thiago", 473};
```

Array de estruturas

```
struct cadastro c1, c2, c3, c4;
```

```
struct cadastro c[4];  
c[2].idade = 200;
```


Atribuição entre estruturas

```
struct ponto {          | struct outro_ponto {  
    int x;              |     int x;  
    int y;              |     int y;  
};                       | };
```

```
struct ponto p1, p2 = {1, 2};  
struct outro_ponto p3 = {3, 4};
```

```
p1 = p2; /* OK */  
p1 = p3; /* ERRADO */
```

Estruturas aninhadas

```
struct endereco {      | struct cadastro {  
    char rua[50];      |     char nome[50];  
    int  numero;       |     int  idade;  
};                     |     struct endereco end;  
                       | };
```

```
struct cadastro cad;  
cad.end.numero = 10;
```

Declarando uma união

```
union nome {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
}
```

struct × **union**

estrutura reserva espaço para armazenar todos os campos

união reserva espaço para o maior campo e a memória é compartilhada

apenas um membro pode ser armazenado por vez
a modificação de um elemento afeta todos

Tipos definidos pelo programador

```
union exemplo {  
    short int x;      /* TAMANHO: 2 BYTES */  
    unsigned char c; /* TAMANHO: 1 BYTE */  
}
```

```
union exemplo u;      /* TAMANHO: 2 BYTES */
```

Principal uso da união

```
union exemplo {  
    short int x;           /* TAMANHO: 2 BYTES */  
    unsigned char c[2]; /* TAMANHO: 2 BYTES */  
}
```

subdividir um tipo básico em um array de partes menores

Declarando uma enumeração

```
enum nome { identificadores };  
enum semana {  
    Domingo, Segunda, Terca, Quarta,  
    Quinta, Sexta, Sabado  
};
```

identificadores são palavras separadas por vírgula
palavras constituem as constantes criadas pela
enumeração

Declarando uma variável do tipo da enumeração

```
enum semana {  
    Domingo, Segunda, Terca, Quarta,  
    Quinta, Sexta, Sabado  
} s1 s2;
```

```
enum semana s3;
```


Enumeração pode ser vista
como uma **lista de constantes**

Cada constante é representada
por um **número inteiro**,
começando do 0

Valores das constantes podem ser definidos pelo programador

```
enum semana {  
    Domingo=1,  
    Segunda, /* = 2 */  
    Terca,   /* = 3 */  
    Quarta=7,  
    Quinta,  /* = 8 */  
    Sexta,   /* = 9 */  
    Sabado   /* = 10 */  
}; /* VALORES DA TABELA ASCII SÃO VÁLIDOS */
```

Comando typedef

```
typedef tipo_existente novo_nome;  
typedef int inteiro;
```

não cria um novo tipo, apenas um sinônimo para um tipo existente

variáveis `tipo_existente` e `novo_nome` podem ser usadas conjuntamente (são do mesmo tipo)

typedef+struct

typedef struct cad {		typedef struct {
char nome[50];		char nome[50];
int idade;		int idade;
char rua[100];		char rua[100];
int numero;		int numero;
} Cadastro;		} Cadastro;

```
typedef struct cad Cadastro;
```

Exercícios