Programação Estruturada

Aula 10 - Arrays

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Até agora, todas as variáveis vistas foram escalares
  - São capazes de armazenar apenas um dado
- Para armazenar uma coleção de valores, a linguagem C possui os arrays
- Os arrays podem ser unidimensionais ou multidimensionais
  - Vamos focar nos unidimensionais

Arrays

- Um array é uma estrutura de dados que contém uma coleção de valores todos do mesmo tipo
- Os valores são chamados de elementos
- Podemos acessá-los individualmente através da sua posição no array

Arrays

- Os elementos de um array unidimensional são organizados sequencialmente, um após o outro

Arrays

- Para declarar um array, precisamos especificar o tipo do dado e o número de elementos
    - `int a[10]`

Acessando elementos

- Para acessar um elemento de um array, nós usamos o nome do array seguido de um número inteiro entre colchetes
    - Chamamos esse número entre colchetes de índice
- Os índices de um array de tamanho `N` começam em 0 e terminam em `N-1`
    - `a[2]` - acessa o terceiro elemento do array

Acessando elementos

- Podemos usar a mesma sintaxe para atribuir valores a um array
  - `a[2] = 10` - atribui o valor 10 à terceira posição do array
- Se um array é de um tipo T, então cada elemento do array é tratado como uma variável do tipo T

Arrays e Laços

- Arrays e laços são usados com frequência em conjunto
- Laços são usados para executar comandos para cada elemento de um array

```c
int a[10];

for (int i=0; i<10; i++) {
    a[i] = 0;
}

for (int i=0; i<10; i++) {
    printf("%d ", a[i]);
}
```

Acessando elementos

- C não verifica se o índice do array é válido
  - Podemos colocar um valor maior que o último índice
- O comportamento do programa será indefinido

Acessando elementos

- O índice para acessar um elemento pode ser uma expressão
  - `a[i+j] = 10`

```
int a[10];
int i;

i=0;
while (i<10) {
    a[i++] = 0;
}
```

Exercício

- Escreva um programa que recebe 10 números e os exibe em ordem reversa

Inicialização

- Podemos definir o valor inicial de um array na sua declaração
- `int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`

Inicialização

- Se a quantidade de valores iniciais for menor que o tamanho do array, os elementos restantes são preenchidos com 0
- `int a[10] = {1, 2, 3, 4, 5}`
  - O array resultante é: `1 2 3 4 5 0 0 0 0 0`
- Usando essa ideia, para inicializar um array com zeros, temos:
  - `int a[10] = {0}`
- A quantidade de valores iniciais não pode ser maior que o tamanho do array

Inicialização

- Passando os valores iniciais, podemos omitir o tamanho do array
- `int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`
    - O array tem tamanho 10

Inicialização

- Suponha que queremos inicializar um array com 100 elementos
  - O 3o elemento é 29
  - O 100 elemento é 48
  - Todos os outros elementos são 0
- Como fazer?

Inicialização

- Podemos usar os inicializadores designados:
- `int a[100] = {[2] = 29, [99] = 48}`
- Os números entre colchetes são as posições
- Os valores após o `=` são os valores para as determinadas posições

Inicialização

- A ordem não importa:
  - `int a[100] = {[99] = 48, [2] = 29}`
- Omitindo o tamanho do array, o compilador infere o tamanho de acordo com as posições passadas:
  - `int a[] = {[2] = 29, [99] = 48}`

sizeof

- O operador `sizeof` determina o tamanho de um array (em bytes)

```
int a[10];
printf("%d", sizeof(a));
```

- Se um inteiro tiver 4 bytes, o programa cima exibe 40

sizeof

- Podemos usar `sizeof(a) / sizeof(a[0])` para calcular o tamanho de um array

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

for (int i=0; i<sizeof(a)/sizeof(a[0]); i++) {
    printf("%d ", a[i]);
}
```

Arrays multidimensionais

- Um array pode ter qualquer número de dimensões
- Para declarar um array de duas dimensões, temos:
- `int m[5][9]`
    - 5 linhas e 9 colunas

Acessando elementos

- Para acessar um elemento, precisamos especificar a linha e a coluna
  - O elemento da 2 linha e 4 coluna: `m[3][5]`

Arrays multidimensionais

- Apesar de visualizarmos um array de duas dimensões como uma tabela, C armazena ele como uma sequência de linhas
- A primeira linha é armazenada, em seguida a segunda, depois a terceira, etc.

Arrays bidimensionais e laços

- Para acessar todos os elementos de um array bidimensional é muito comum usarmos laços aninhados
- O programa abaixo cria e imprime uma matriz identidade

Arrays bidimensionais e laços

```
int a[10][10];

    for (int linha=0; linha<10; linha++) {
        for (int coluna=0; coluna<10; coluna++) {
            if (linha == coluna) {
                a[linha][coluna] = 1;
            } else {
                a[linha][coluna] = 0;
            }
        }
    }

    for (int linha=0; linha<10; linha++) {
        for (int coluna=0; coluna<10; coluna++) {
            printf("%d ", a[linha][coluna]);
        }
        printf("\n");
    }
```

Inicialização de arrays multidimensionais

- Podemos inicializar um array bidimensional aninhando inicializações de arrays unidimensionais

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}}
```

Inicialização de arrays multidimensionais

- Se faltarem linhas na inicialização, o compilador preenche com 0

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0}}
```

Inicialização de arrays multidimensionais

- Se faltarem colunas na inicialização, o compilador preenche com 0

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1},
               {0, 1, 0, 1, 1, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}}
```

Inicialização de arrays multidimensionais

- Você pode omitir as chaves internas
- Após terminar uma linha, o compilador passa automaticamente para outra

```
int m[5][9] = {1, 1, 1, 1, 1, 0, 1, 1, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0,
               0, 1, 0, 1, 1, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 1, 1, 1}
```

Inicialização de arrays multidimensionais

- Inicializadores designados também funcionam para arrays multidimensionais
- `int ident[2][2] = {[0][0] = 1, [1][1] = 1}`
    - Matriz identidade 2x2

Arrays constantes

- Utilizando a palavra reservada `const`, o programador define que um array não pode ser modificado

```
const int a[10] = {1,2,3,4,5,6,7,8,9,10};
a[0] = 99;
```

- O código acima retorna um erro de compilação