Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Evandro Chagas Ribeiro da Rosa

Arrays

Agrupar múltiplos valores do mesmo tipo

- Variáveis dos tipos `int`, `float` e `char` armazenam um valor de cada tipo por vez.
- Usando um array, é possível fazer com que uma variável armazene uma lista de valores.
- Os valores são armazenados de maneira contínua na memória.
- O tamanho do array precisa ser conhecido em tempo de compilação.

Inicializando Arrays

```
const int TAMANHO = 6;

int lista1[4];

float lista2[TAMANHO];

int lista3[] = {8, 3, 3, 5, 1};

int lista4[8] = {4, 4, 3};
```

Representação na Memória

- `lista1` → `|?|?|?|?|`
- `lista2` → `|?|?|?|?|?|?|`
- `lista3` → `|8|3|3|5|1|`
- `lista4` → `|4|4|3|0|0|0|0|0|`

`?` representa lixo de memória

Acessando Elementos do Array

Cada elemento do array pode ser indexado por um número inteiro positivo no intervalo de 0 ao tamanho do array - 1.

Exemplo

- Array de tamanho 5
- `índices = |0|1|2|3|4|`
- `valores = |8|3|3|5|1|`
- Usar índices maiores que 4 é *comportamento indefinido*

Escrita

```
int lista[3];
lista[0] = 4;
lista[1] = 6;
lista[2] = 1;
```

Leitura

```
int total = lista[0] + lista[1] + lista[2];
```

Laço Baseado em Intervalo

A instrução `for` é amplamente usada para iterar sobre um array.

Iterar a lista usando índices

```
const int tamanho = 5;
int lista[tamanho] = {1, 2, 3, 4, 5};

for (int i = 0; i < tamanho; i++) {
    std::cout << lista[i] << " ";
}
```

Iterar a lista por elemento

```
int lista[] = {1, 2, 3, 4, 5};

for (int item : lista) {
    std::cout << item << " ";
}
```

Atribuição com Arrays

```
const int tamanho = 5;
int lista1[tamanho] = {1, 2, 3, 4, 5};
int lista2[tamanho];
```

Como copiar os valores de `lista1` para `lista2`?

❌ Incorreto

```
lista2 = lista1;    // erro
```

✅ Correto

```
for (int i = 0; i < tamanho; i++) {
    lista2[i] = lista1[i];
}
```

`std::cin` com Arrays

```
const int tamanho = 5;
int lista[tamanho];
```

Como usar `std::cin` para escrever valores no array `lista`?

❌ Incorreto

```
std::cin >> lista;    // erro
```

✅ Correto

```
for (int i = 0; i < tamanho; i++) {
    std::cin >> lista[i];
}
```

```
for (int& item : lista) {
    std::cin >> item;
}
```

`std::cout` com Arrays

```
const int tamanho = 5;
int lista[tamanho];
```

Como usar `std::cout` para ler o array `lista`?

❌ Incorreto

```
std::cout << lista << '\n';
```

Será impresso o endereço de memória onde começa o array.

✅ Correto

```
for (int i = 0; i < tamanho; i++)
    std::cout << lista[i] << ' ';
std::cout << '\n';
```

```
for (int item : lista)
    std::cout << item << ' ';
std::cout << '\n';
```

Passando Arrays como Argumento

```
void print_array(int[], int);

int main() {
    int lista[] = {10, 100, 200, 300};
    print_array(lista, 4);
}

void print_array(int arg[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        std::cout << arg[i] << ' ';
    }
    std::cout << '\n';
}
```

- `arg` é uma variável que armazena o início do array;
- Não é necessário passar o tamanho do array para `arg`;
- No parâmetro da função, o tamanho do array é ignorado.
- Sempre é necessário fornecer o tamanho do array por outro parâmetro.

Modificando o Array Dentro da Função

```c
void print_array(int[], int);
void dobrar_array(int[], int);

int main() {
    int lista[] = {10, 100, 200, 300};
    dobrar_array(lista, 4);
    print_array(lista, 4);
}

void dobrar_array(int lista[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        lista[i] *= 2;
    }
}
```

Como é o endereço do início do array que é copiado para a função, a variável de parâmetro se comporta como uma variável de referência.

Garantir que a Função Não Altere o Array

```
void print_array(const int[], int);

int main() {
    int lista[] = {10, 100, 200, 300};
    print_array(lista, 4);
}

void print_array(const int arg[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        std::cout << arg[i] << ' ';
    }
    std::cout << '\n';
}
```

Uma boa prática para garantir que o array passado como argumento não será alterado é definir o parâmetro como `const`.

🔧 Funções Úteis

Imprimindo o Conteúdo do Array

```
void print_array(const int arg[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        std::cout << arg[i] << ' ';
    }
    std::cout << '\n';
}
```

🔧 Funções Úteis

Somar Elementos do Array

```
int somar(const int lista[], int tamanho) {
    int total = 0;
    for (int i = 0; i < tamanho; i++) {
        total += lista[i];
    }

    return total;
}
```

🔧 Funções Úteis

Calcular a Média dos Valores do Array

```
int somar(const int lista[], int tamanho);

double media(const int lista[], int tamanho) {
    double total = somar(lista, tamanho);
    return total / tamanho;
}
```

🔧 Funções Úteis

Maior Valor do Array

```c
#include <limits.h>

int maior_valor(const int lista[], int tamanho) {
    int max = INT_MIN;
    for (int i = 0; i < tamanho; i++) {
        if (lista[i] > max) {
            max = lista[i];
        }
    }

    return max;
}
```

🔧 Funções Úteis

Menor Valor do Array

```c
#include <limits.h>

int menor_valor(const int lista[], int tamanho) {
    int min = INT_MAX;
    for (int i = 0; i < tamanho; i++) {
        if (lista[i] < min) {
            min = lista[i];
        }
    }

    return min;
}
```

🔧 Funções Úteis

Copiar um Array

```c
void copiar_array(const int fonte[], int destino[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        destino[i] = fonte[i];
    }
}
```

🔧 Funções Úteis

Somar dois Arrays

```c
void somar_array(const int lhs[], const int rhs[], int resultado[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        resultado[i] = lhs[i] + rhs[i];
    }
}
```

Qual o valor de `lista1`, `lista2` e `lista3`?

```c
int lista1[4] = {1, 2, 3, 4};
int lista2[4] = {4, 3, 2, 1};
int lista3[4];
somar_array(lista1, lista2, lista3, 4);
```

```c
int lista1[4] = {1, 2, 3, 4};
int lista2[4] = {4, 3, 2, 1};
int lista3[4];
somar_array(lista1, lista2, lista1, 4);
```

Array de Duas Dimensões

A Matriz

- O que vimos até agora são arrays de uma dimensão*.

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

- É possível criar um array de duas dimensões:

```
[0][0] [0][1] [0][2] [0][3] [0][4] [0][5]
[1][0] [1][1] [1][2] [1][3] [1][4] [1][5]
[2][0] [2][1] [2][2] [2][3] [2][4] [2][5]
[3][0] [3][1] [3][2] [3][3] [3][4] [3][5]
```

* Em alguns casos chamados de *vetor*.

Declarando um Array 2D

```
const int N_LINHAS = 2;
const int N_COLUNAS = 3;
int matriz[N_LINHAS][N_COLUNAS];
```

	Coluna 0	Coluna 1	Coluna 2
Linha 0	`matriz[0][0]`	`matriz[0][1]`	`matriz[0][2]`
Linha 1	`matriz[1][0]`	`matriz[1][1]`	`matriz[1][2]`

Inicializando um Array 2D

```
int matriz[][2] = {{8, 5}, {7, 9}, {6, 3}};
```

	Coluna 0	Coluna 1
Linha 0	8	5
Linha 1	7	9
Linha 2	6	3

Representação na Memória

|8|5|7|9|6|3|

As colunas são armazenadas de maneira sequencial na memória.

Usando Array 2D em Função

```
void print_matriz(const int[][2], int);

int main() {
    int matriz[][2] = {{8, 5}, {7, 9}, {6, 3}};
    print_matriz(matriz, 3);
}

void print_matriz(const int arg[][2], int linhas) {
    for (int linha = 0; linha < linhas; linha++) {
        for (int item : arg[linha]) {
            std::cout << item << '\t';
        }
        std::cout << '\n';
    }
}
```

- É necessário definir o tamanho da coluna no tipo do parâmetro `arg`.
- É necessário passar o número de linhas de `arg` em um parâmetro extra.
- Similar ao array 1D, `arg` é uma cópia do endereço inicial do array 2D.

🔧 Funções com Array 2D: Somar Colunas

```c
const int COLUNAS = 2;
void somar_colunas(const int[][COLUNAS], int[], int);
void print_array(const int[], int);

int main() {
    int matriz[][COLUNAS] = {{8, 5}, {7, 9}, {6, 3}};
    int soma_colunas[COLUNAS];
    somar_colunas(matriz, soma_colunas, 3);
    print_array(soma_colunas, COLUNAS);
}

void somar_colunas(const int matriz[][COLUNAS], int resultado[], int linhas) {
    for (int i = 0; i < COLUNAS; i++) resultado[i] = 0;

    for (int linha = 0; linha < linhas; linha++) {
        for (int coluna = 0; coluna < COLUNAS; coluna++) {
            resultado[coluna] += matriz[linha][coluna];
        }
    }
}
```

🔧 Funções com Array 2D: Somar Linhas

```c
const int COLUNAS = 2;
void somar_linhas(const int[][COLUNAS], int[], int);
int somar(const int[], int);
void print_array(const int[], int);

int main() {
    int matriz[][COLUNAS] = {{8, 5}, {7, 9}, {6, 3}};
    int soma_linhas[3];

    somar_linhas(matriz, soma_linhas, 3);
    print_array(soma_linhas, 3);
}

void somar_linhas(const int matriz[][COLUNAS], int resultado[], int linhas) {
    for (int linha = 0; linha < linhas; linha++) {
        resultado[linha] = somar(matriz[linha], COLUNAS);
    }
}
```

Complete o Código

```
int main() {
    int matriz[][...] = ...;
    int media_linhas[...];
    int media_colunas[...];

    calcular_media_linhas(matriz, media_linhas, ...);
    calcular_media_colunas(matriz, media_colunas, ...);

    std::cout << "Valor médio das linhas\n";
    print_array(media_linhas, ...);
    std::cout << "Valor médio das colunas\n";
    print_array(media_colunas, ...);
    std::cout << "Valor médio da matriz\n";
    std::cout << calcular_media_matriz(matriz, ...) << '\n';

    return 0;
}
```

Use a matriz abaixo no código:

12	24	32	21	42
14	67	87	65	90
19	1	24	12	8

Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Evandro Chagas Ribeiro da Rosa