ESTRUTURA DE DADOS: VETORES E MATRIZES

ALUNOS: JEFFERSON CRISTINO DA COSTA - 11821BSI227

NATHAN SILVA NEVES - 12111BSI247

LUIZ OTAVIO DIAS - 12021BSI207

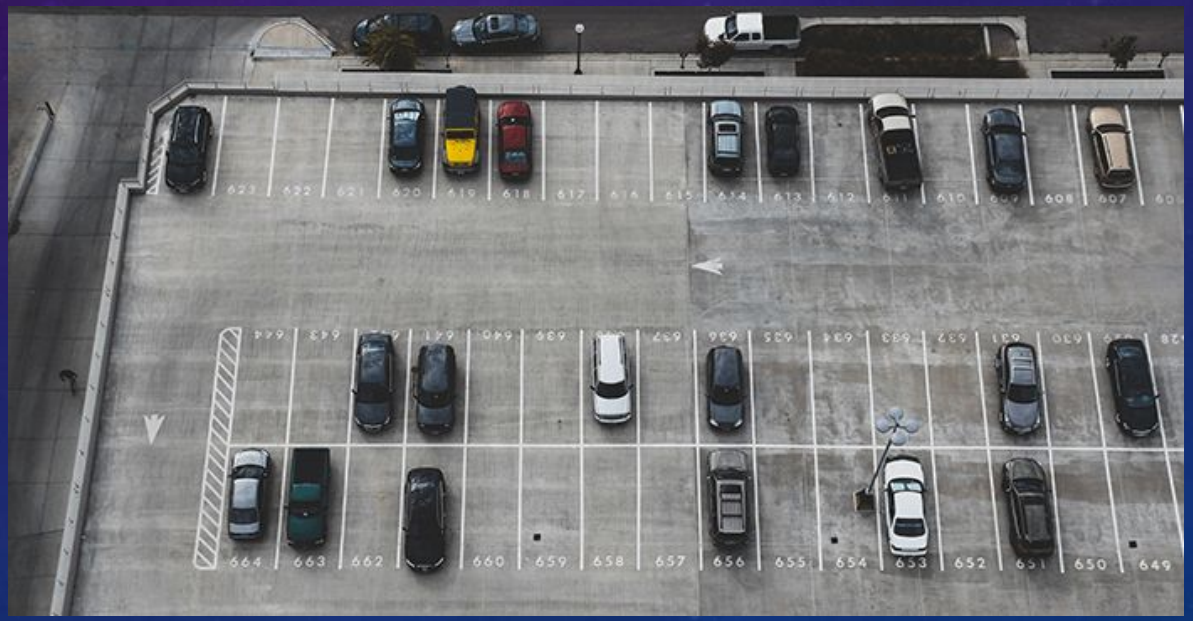
Vetores

Um vetor (ou array unidimensional) é uma estrutura de dados que armazena elementos do mesmo tipo em posições consecutivas da memória.

- Possuem tamanho fixo determinado na alocação.
- Elementos acessados por índice
- Armazenam dados do mesmo tipo
- Eficiência no acesso direto

Analogia

- Cada "vaga" no estacionamento tem um número de identificação (índice do array).
- Dentro de cada vaga, temos um carro (o valor armazenado no array).
- Para acessar o carro na vaga 2, usamos carros[2], que retorna 30.



Declaração de Vetor em C

```
int vetor[5];
// Declara um vetor de 5 inteiros sem inicialização

int vetor[5] = {1, 2, 3, 4, 5};
// Vetor com 5 elementos já inicializados

int vetor[] = {1, 2, 3, 4, 5};
// O compilador infere o tamanho (5 elementos)

int vetor[5] = {1, 2};
// Equivalente a {1, 2, 0, 0, 0}

int *vetor = (int *)malloc(5 * sizeof(int));
// Aloca dinamicamente um vetor de 5 inteiros
```

Declaração de Vetor em C

```
const int vetor[3] = {1, 2, 3};
// O conteúdo do vetor não pode ser alterado

char texto[] = "Olá";
// Vetor de caracteres terminado em '\0' automaticamente

char texto[] = {'O', 'l', 'á', '\0'};

char *nomes[] = {"Ana", "Carlos", "Maria"};
```

Preenchimento de Vetor em C

```c
#include <stdio.h>

int main() {
    // Iniciando vetor com 5 posições
    int numeros[5] = {10, 20, 30, 40, 50};

    // Exibindo a posição e os valores do vetor
    for (int i = 0; i < 5; i++) {
        printf("Elemento %d: %d\n", i, numeros[i]);
    }

    return 0;
}
```

Matrizes

Uma matriz (ou array bidimensional) é uma estrutura que armazena dados em forma de tabela, organizada em linhas e colunas.

- Estrutura bidimensional (linhas e colunas)
- Elementos acessados por índices (linha, coluna)
- Utilizados para armazenar relações entre dados.

Matrizes

- Cada andar representa uma linha da matriz.
- Cada vaga em um andar representa uma coluna da matriz.

Declaração de Matrizes em C

```
int matriz[3][4]; // Matriz com 3 linhas e 4 colunas (sem inicialização)

int matriz[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
}; // Matriz com 2 linhas e 3 colunas (com inicialização)

int matriz[3][3] = {
    {1, 2},
    {3},
    {4, 5, 6}
};
// Matriz com 3 linhas e 3 colunas com inicialização parcial
```

1	2	0
3	0	0
4	5	6

Declaração de Matrizes em C

```
int **matriz;
int linhas = 3, colunas = 4;

matriz = (int **)malloc(linhas * sizeof(int *)); // Aloca um array de ponteiros

for (int i = 0; i < linhas; i++) {
    matriz[i] = (int *)malloc(colunas * sizeof(int)); // Aloca cada linha
}
```

```
char nomes[3][10] = {
    "Ana",
    "Carlos",
    "Maria"
};
```

Preenchimento de Matrizes em C

```c
#include <stdio.h>

int main() {
    int n = 5; // Tamanho da matriz (5x5)
    int matriz[n][n];

    // Preenchendo a matriz com 0 e 1 para formar um "X"
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j || i + j == n - 1) {
                matriz[i][j] = 1; // Diagonais principais recebem 1
            } else {
                matriz[i][j] = 0; // Restante recebe 0
            }
        }
    }
```

```
// Exibindo a matriz
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Conclusão

- Vetores são úteis para armazenar sequências de valores.
- Matrizes são úteis para organizar dados em formato tabular.
- Ambas as estruturas permitem acesso rápido e eficiente aos elementos.
- A escolha entre vetor e matriz depende da necessidade do problema.

Obrigado!!!