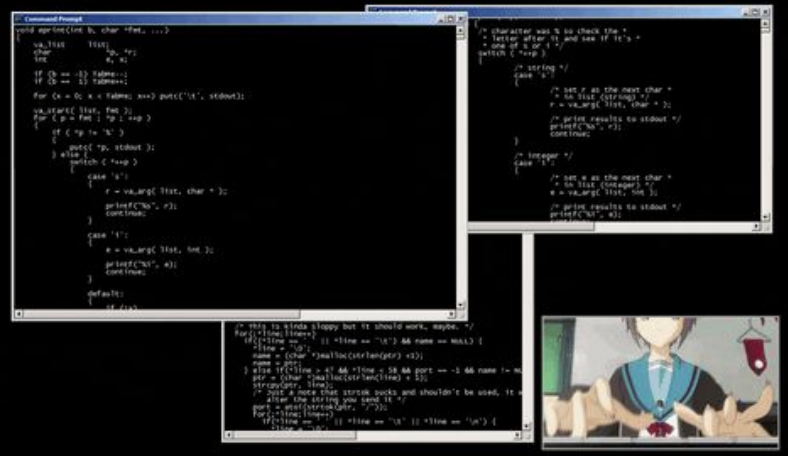
Algoritmos e Estrutura de Dados I

Alocação dinâmica: Vetores e Matrizes

Profa. Dra. Rosana Rego

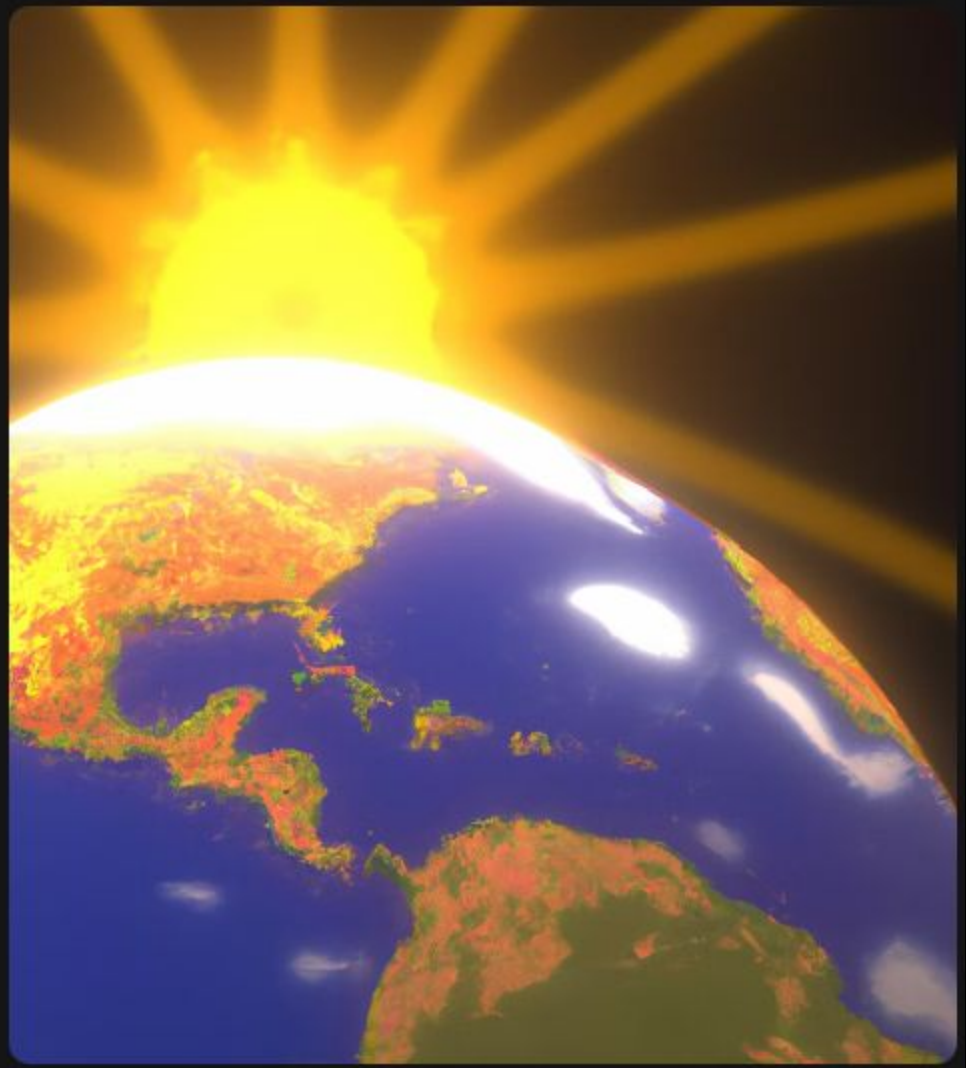
Sumário



Introdução

Vetores e matrizes são recursos fundamentais para a programação em C. Eles permitem o armazenamento de dados em estruturas de dados de forma eficiente. Esta apresentação discutirá a importância e a aplicabilidade destes recursos no mundo real.

A linguagem C fornece suporte para vetores e matrizes, permitindo aos programadores criar algoritmos eficientes para resolver problemas complexos.



A Importância dos Vetores e Matrizes

Vetores e matrizes são estruturas de dados fundamentais para a programação em C. Eles permitem o armazenamento de dados em estruturas de dados de forma eficiente. Estas estruturas de dados permitem aos programadores criar algoritmos para resolver problemas complexos.

Vetores e matrizes também permitem aos programadores armazenar dados de forma organizada e acessar esses dados de forma rápida e eficiente.

Aplicabilidade no Mundo Real

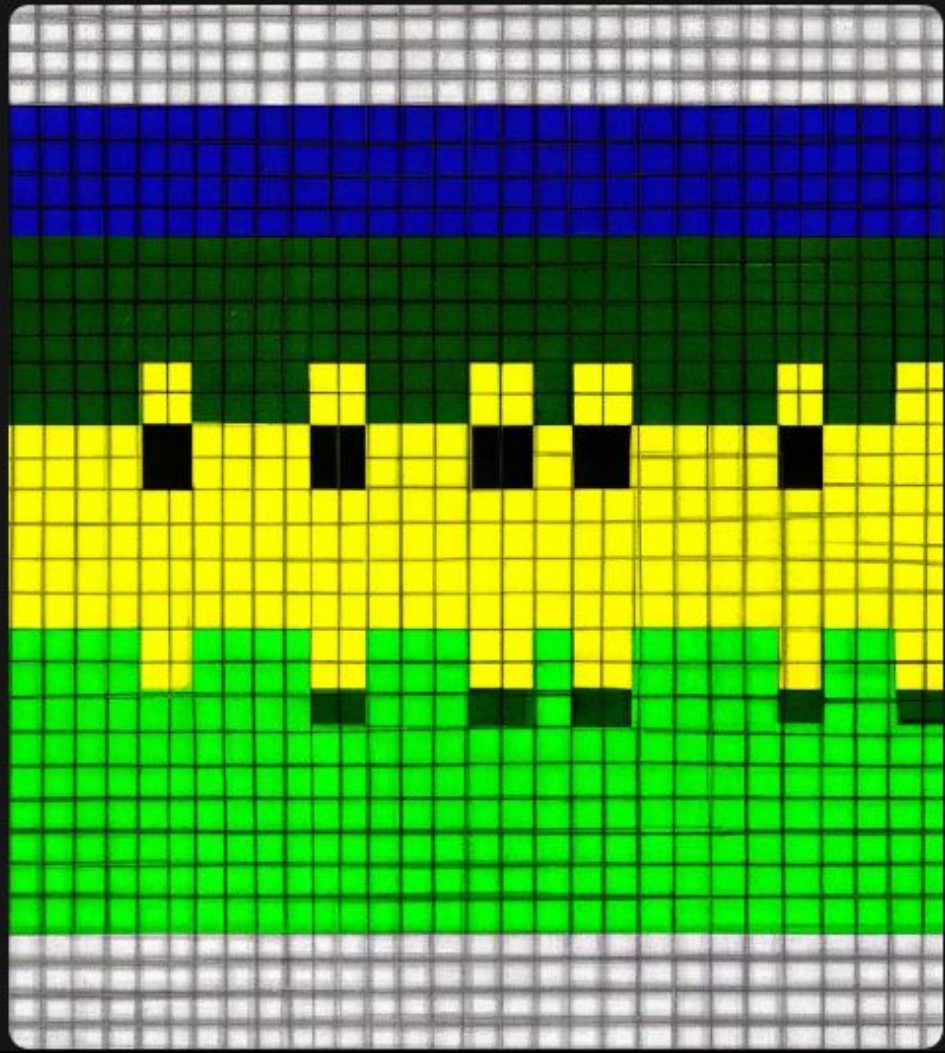
Vetores e matrizes têm muitas aplicações no mundo real. Por exemplo, eles são usados em jogos para armazenar informações sobre os jogadores e as ações que eles tomam. Eles também são usados em sistemas de inteligência artificial, para processar e armazenar informações sobre o ambiente.

Vetores e matrizes também são usados em aplicativos de negócios, para armazenar e processar dados financeiros. Eles são usados para criar modelos de previsão de mercado, para ajudar os investidores a tomar decisões informadas.

Aplicação de Matrizes em Processamento de Imagem Digital

As matrizes são usadas na criação e manipulação de imagens digitais, permitindo que sejam aplicados filtros e transformações que possibilitam a edição de fotos.

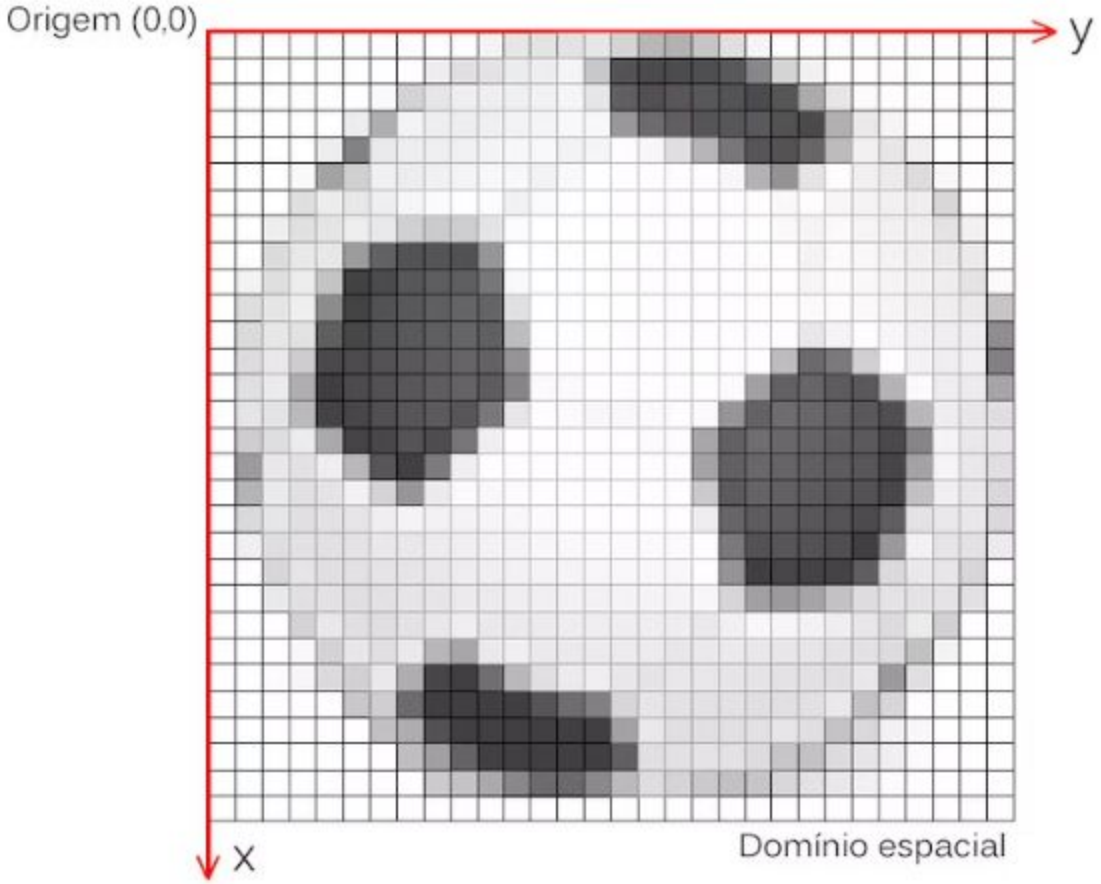
Matrizes também são usadas em sistemas de detecção de objetos e reconhecimento de imagens, permitindo que os computadores possam identificar e interpretar imagens.



a) Imagem original

b) Imagem redimensionada (30x30 pixel)

c) Representação da imagem em matriz de pixel (zoom)



$$\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{bmatrix}$$

Fig. 1

Fig. 2

Vantagens dos Vetores e Matrizes

Vetores e matrizes oferecem várias vantagens para os programadores. Eles permitem aos programadores armazenar e acessar dados de forma eficiente. Eles também permitem aos programadores criar algoritmos para resolver problemas complexos de forma rápida e eficiente.

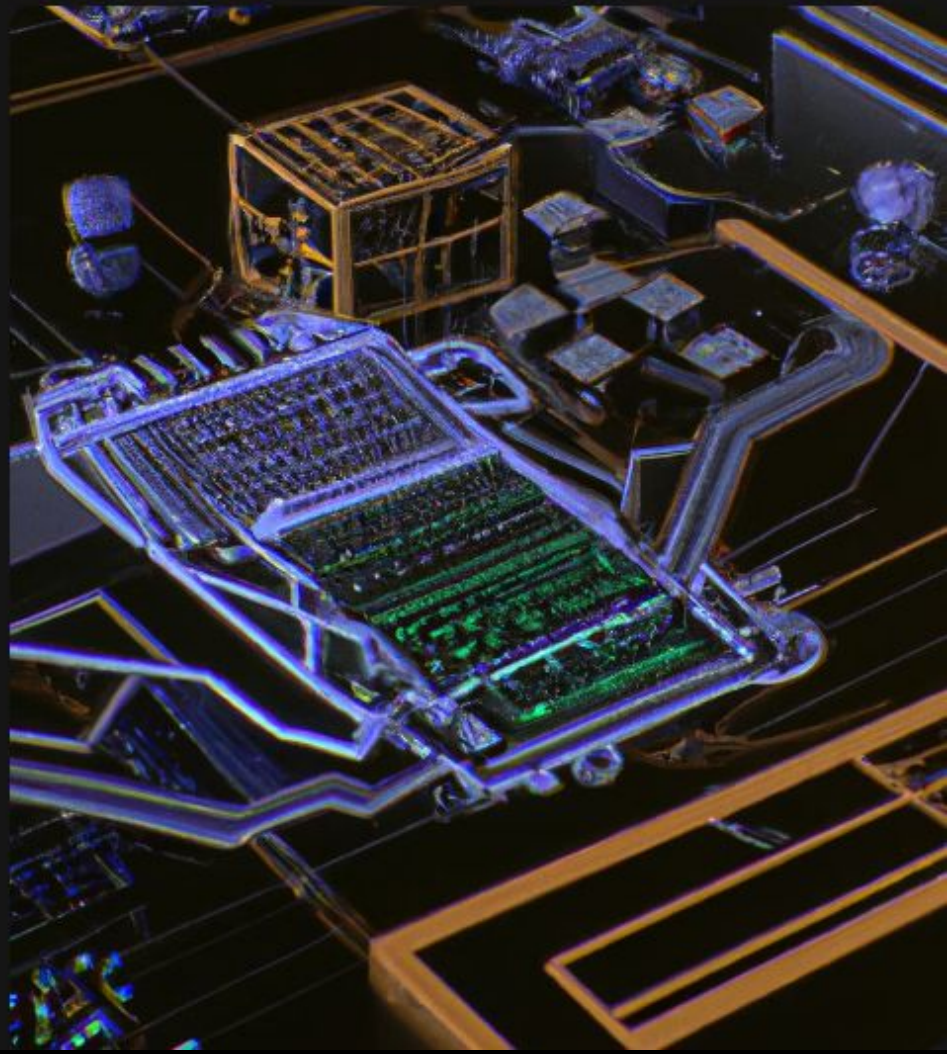
Vetores e matrizes também permitem aos programadores criar códigos mais limpos e reutilizáveis. Eles também permitem aos programadores criar códigos mais confiáveis, já que os programadores podem testar seus algoritmos com mais facilidade.

Alocação Dinâmica

A alocação dinâmica de memória é um processo de alocar espaço de memória durante a execução de um programa.

Isso permite que os programas aloquem e libertem espaço de memória conforme necessário, o que pode ser útil em situações onde o tamanho das estruturas de dados não é conhecido no momento da compilação.

Em C, a alocação dinâmica de memória é geralmente realizada usando as funções `malloc()`, `calloc()` e `realloc()`.



ATENÇÃO

É importante notar que a alocação dinâmica de memória também exige liberação manual de memória, usando a função free(). Se essa liberação não for realizada, poderá ocorrer vazamento de memória, o que pode causar problemas de desempenho e estabilidade.

Declarando Vetores Dinamicamente

```
int *vetor = (int *) malloc(tamanho * sizeof(int));
```

ou

```
int *vetor = (int *) calloc(tamanho, sizeof(int));
```

onde "tamanho" é o número de elementos que você deseja alocar para o vetor e "int" é o tipo de dado do vetor.

A função `malloc()` aloca um bloco contíguo de memória do tamanho especificado, enquanto a função `calloc()` aloca um bloco contíguo de memória e inicializa todos os bytes para 0.

Exemplo

Ao usar as funções malloc e calloc, é importante verificar se a alocação foi bem-sucedida, pois elas retornam o ponteiro NULL em caso de falha.

```
int tamanho = 5;

int *vetor = (int *) malloc(tamanho * sizeof(int));

if (vetor == NULL) {

    printf("Erro ao alocar memória");

    exit(1);

}
```

Exemplo

É importante notar que ao usar vetores dinamicamente é necessário liberar manualmente a memória quando não for mais preciso usando a função free().

```
free(vetor);
```

Exemplo: Alocação Dinâmica de Vetor de inteiros

```c
 include <stdio.h>
 include <stdlib.h>
int main() {
    int tamanho=5, i;
    int *vetor;
    vetor = (int *) malloc(tamanho * sizeof(int));
    if (vetor == NULL) {
        printf("Erro ao alocar memória");
        exit(1);
    }
```

Continuação exemplo

```
    // Lendo valores do usuário e armazenando no veto
    printf("Entre com os valores do vetor: \n");
    for (i = 0; i < tamanho; i++) {
        scanf("%d", &vetor[i]);
    }

    // Liberando a memória alocada
    free(vetor);
    return 0;
}
```

Função Realloc

A função realloc() é uma função em C que é usada para redimensionar uma área de memória já alocada dinamicamente. Ela tem a seguinte sintaxe:

```
void *realloc(void *ptr, size_t size);
```

A função realloc() é útil quando você precisa alocar mais memória para um vetor ou matriz dinâmica, mas não quer perder os dados já armazenados nele. Ela também pode ser usada para liberar espaço desnecessário, se o novo tamanho for menor do que o antigo.

Exemplo

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int tamanho = 5, novo_tamanho;
    int *vetor = (int *) malloc(tamanho * sizeof(int));
    printf("\nEntre com o novo tamanho do vetor: ");
    scanf("%d", &novo_tamanho);
    vetor = (int *) realloc(vetor, novo_tamanho * sizeof(int));
    if (vetor == NULL) {
        printf("Erro ao alocar memória");
        exit(1);
    }
    free(vetor);
    return 0;}
```


Declarando matrizes

Declarando Matrizes Dinamicamente

Alocar uma matriz dinamicamente em C é semelhante a alocar um vetor dinamicamente. A diferença é que uma matriz é um conjunto de vetores, então é necessário alocar cada linha da matriz separadamente.

Para alocar uma matriz dinamicamente, você pode usar a função malloc() duas vezes. A primeira vez para alocar o vetor de ponteiros que representa as linhas da matriz, e a segunda vez para alocar cada linha da matriz.

Afinal, o que é uma matriz?

Uma matriz é uma estrutura de dados bidimensional que consiste em uma coleção de números, geralmente representados por elementos da mesma tipagem, dispostos em linhas e colunas. Ela é utilizada para representar dados relacionados a tabelas, imagens, entre outros.

$$A = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 5 & 1 \\ 2 & 1 & 3 \end{bmatrix}$$

A matriz A possui 3 linhas e 3 colunas, e os elementos estão dispostos em 3 linhas e 3 colunas.

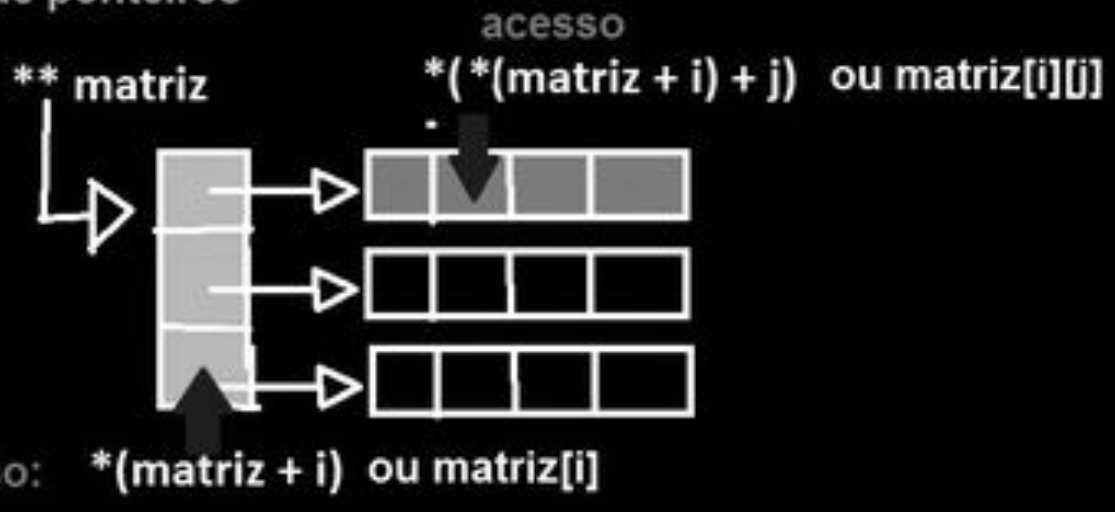
Exemplo: Alocando Matrizes Dinamicamente

```c
#include <stdio.h>

#include <stdlib.h>

int main() {

    int linhas=3, colunas=3, i, j;

    // Alocando vetor de ponteiros

    int **matriz = (int **) malloc(linhas * sizeof(int *));

    // Alocando cada linha da matriz

    for (i = 0; i < linhas; i++) {

        matriz[i] = (int *) malloc(colunas * sizeof(int));

    }
```

Exemplo Continuação

```
// Liberando a memória alocada
    for (i = 0; i < linhas; i++) {
        free(matriz[i]);
    }
    free(matriz);
    return 0;
}
```

Matriz representada como
vetor de ponteiros



QUANTIDADE DE LINHAS (PONTEIROS).
CADA PONTEIRO É UMA LINHA

QUANTIDADE DE COLUNA

Exemplo: Multiplicação de Matrizes

```c
#include <stdio.h>

#include <stdlib.h>

int main() {

    int linhasA, colunasA, linhasB, colunasB, i, j, k;

    printf("Entre com o número de linhas e colunas da matriz A: ");

    scanf("%d %d", &linhasA, &colunasA);

    printf("Entre com o número de linhas e colunas da matriz B: ");

    scanf("%d %d", &linhasB, &colunasB);
```

Continuação

```
// Verificando se as matrizes podem ser mu
    if (colunasA != linhasB) {
        printf("As matrizes não podem ser multiplicadas.\n");
        return 0;
    }    // Alocando matriz A
    int **matrizA = (int **) malloc(linhasA * sizeof(int *));
    for (i = 0; i < linhasA; i++) {
        matrizA[i] = (int *) malloc(colunasA * sizeof(int));
    }
```

Continuação

```
// Alocando matriz B
    int **matrizB = (int **) malloc(linhasB * sizeof(int *));
    for (i = 0; i < linhasB; i++) {
        matrizB[i] = (int *) malloc(colunasB * sizeof(int));
    }    // Lendo valores do usuário e armazenando nas matrizes
    printf("Entre com os valores da matriz A: \n");
    for (i = 0; i < linhasA; i++) {
        for (j = 0; j < colunasA; j++) {
            scanf("%d", &matrizA[i][j]);
        }
    }
```

Continuação

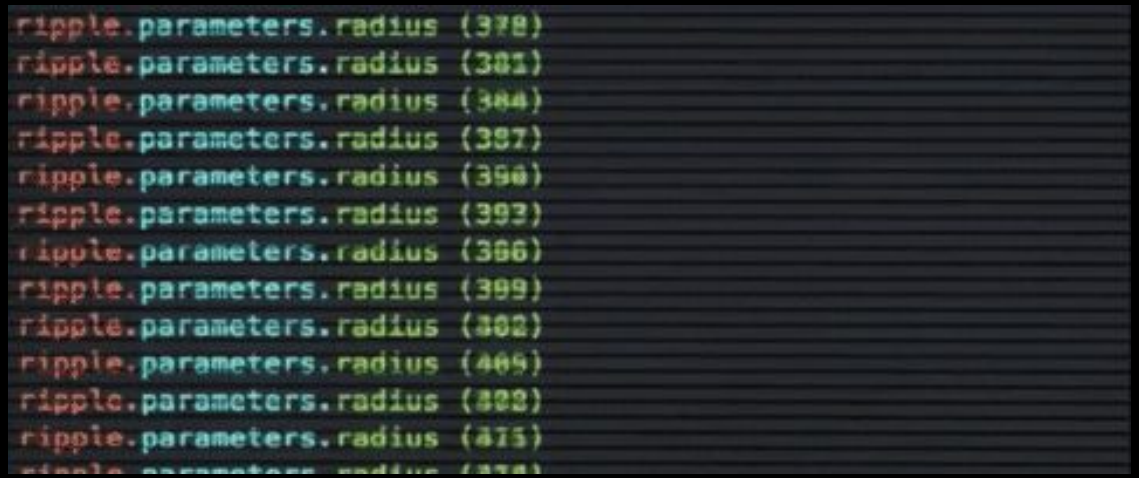
```
printf("Entre com os valores da matriz B: \n");

    for (i = 0; i < linhasB; i++) {

        for (j = 0; j < colunasB; j++) {

            scanf("%d", &matrizB[i][j]);

        }

    }

    // Alocando matriz C

    int **matrizC = (int **) malloc(linhasA * sizeof(int *));

    for (i = 0; i < linhasA; i++) {

        matrizC[i] = (int *) malloc(colunasB * sizeof(int));

    }
```

Continuação

```
    for (i = 0; i < linhasA; i++) {

        for (j = 0; j < colunasB; j++) {

            for (k = 0; k < colunasA; k++) {

                matrizC[i][j] += matrizA[i][k] * matrizB[k][j];

            }

        }

    }

    return 0;

}
```

Vamos testar seus conhecimentos…

PIN do jogo: **05332832**



Exercício

1. Exercício: Aloque dinamicamente uma matriz de inteiros 5x5 e preencha cada elemento da matriz com o seu respectivo índice (matriz[i][j] = i*5 + j). Em seguida, imprima a matriz.

   Nota: Não se esqueça de liberar a memória alocada após o uso da matriz.

Conclusão

Vetores e matrizes são recursos fundamentais para a programação em C. Eles permitem o armazenamento de dados em estruturas de dados de forma eficiente. Estes recursos têm muitas aplicações no mundo real e oferecem várias vantagens para os programadores.

A linguagem C fornece suporte para vetores e matrizes, permitindo aos programadores criar algoritmos eficientes para resolver problemas complexos. Esta apresentação discutiu a importância e a aplicabilidade destes recursos no mundo real.

Referência para estudo

Livro base: ***Introdução a Estrutura de dados de Waldemar Celes.***

Capítulo 5. Assunto: ***Vetores e Alocação dinâmica.***

Capítulo 8. Assunto: ***Matrizes***