Unidade III

5 VETORES

5.1 Introdução a vetores (arrays unidimensionais)

Em C, vetores (ou arrays unidimensionais) são estruturas que armazenam vários valores do mesmo tipo em posições sequenciais (ou consecutivas) da memória. Cada elemento do vetor é acessado por meio de um índice numérico, que em C sempre começa em 0 e vai até tamanho – 1. Assim, em um vetor de tamanho n, o primeiro elemento está na posição 0 e o último na posição n – 1.

Em outras palavras, podemos definir um vetor como uma estrutura de dados que armazena uma coleção de elementos do mesmo tipo, organizados em sequência na memória, e acessados por meio de índices inteiros.

Imagine uma fileira de armários numerados em um depósito. Cada armário tem um número (o índice) e, dentro de cada armário, há um objeto (o valor armazenado). Para encontrar um objeto específico, você precisa saber o número do armário (índice). Da mesma forma, para acessar um valor específico em um vetor, você usa o seu índice.

Vetores são ideais para representar listas de dados, como notas de alunos, pontuação em jogos, listas de preços de produtos ou elementos de um inventário.

Exemplo: imagine um conjunto de caixas numeradas (índice) de 0 até n – 1, sendo n = 6 a quantidade total de caixas. Cada caixa guarda um número (valor).

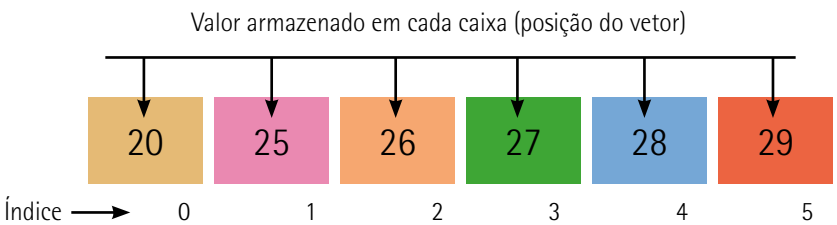


Figura 32 – Exemplo de vetor (caixas numeradas)

A figura 32 ilustra o conceito de um vetor (ou array) na programação utilizando a analogia de caixas numeradas. Assim, demonstra que um vetor é uma estrutura de dados que armazena uma coleção de elementos do mesmo tipo, em que cada elemento pode ser acessado através de um índice numérico. O índice começa geralmente em 0 e vai até o tamanho do vetor menos 1. Em C, o índice do vetor sempre é iniciado com 0.

A legenda acima das caixas indica que cada uma representa uma posição dentro do vetor, e o número dentro da caixa é o valor armazenado nessa posição. Na figura 32, temos um conjunto de seis caixas, sendo n = 6 a quantidade total de caixas. Cada caixa tem um índice, sendo a primeira caixa de índice 0 e a última, n – 1. Dentro de cada caixa podemos guardar um número (20, 25, 26, 27, 28, 29), e estes são os valores que estão armazenados em cada uma das posições do vetor. O tipo desses valores pode variar (inteiros, decimais, caracteres etc.).

5.1.1 Declaração e preenchimento dos valores do vetor

Ao declarar um vetor na linguagem de programação C, devemos obrigatoriamente dizer qual é o tipo associado a ele. Vetores podem ser de qualquer tipo de dado válido, ou seja, qualquer tipo de dado que possa ser usado para declarar variáveis também pode ser usado para declarar vetores.

Definição básica: em C, um vetor deve ter: tipo nome_do_vetor[tamanho].

Exemplo: sintaxe de um vetor na linguagem C: int notas[5];, onde int é o tipo (inteiro), notas é o nome do vetor e 5 entre colchetes é o tamanho do vetor, ou seja, quantas posições o vetor tem.

Podemos acessar suas posições da seguinte forma (nome e posição):

```
notas[0], notas[1], notas[2], notas[3], notas[4]
```

O índice sempre começa em 0, portanto, em um vetor de tamanho 5, o último elemento estará na posição 4 (ou seja, tamanho – 1).

Podemos inicializar as posições do vetor na declaração do vetor ou depois da declaração.

No primeiro exemplo, vamos preencher o vetor (inicializando na declaração):

```
int idades[3] = {10, 20, 30};
```

Esse vetor foi declarado do tipo inteiro (int), com o nome de idades e inicializado com os valores 10, 20 e 30.

Se você fornecer menos valores do que o tamanho, os restantes serão zerados automaticamente.

Agora vamos preencher o vetor (inicializando após a declaração):

```
int notas[2];
notas[0] = 10;  // primeiro elemento
notas[1] = 30;  // último elemento
```

Cuidado com os limites do vetor: em C não há verificação automática de limites, ou seja, não há proteção contra acesso fora dos limites do vetor. Isso pode causar comportamento indefinido ou erros graves. O compilador não avisa o erro.

```
//ERRO: fora do limite! Pode corromper memória
idades[10] = 99;
```

Também podemos pedir ao usuário para digitar o tamanho do vetor e, depois de ter esse valor, declaramos o vetor e passamos a variável que tem o tamanho ao vetor.

```
int tam;

printf("Digite o tamanho do vetor: ");
scanf("%d", &tam);//tam armazena o tamanho do vetor

int vetor[tam];
```

Caso o usuário digitasse 3, seria armazenado esse valor na variável tam, consequentemente, o vetor teria o tamanho 3.

No próximo código, preenche-se o vetor com tamanho predefinido:

```
int i, nota[4];

printf("Digite a primeira nota: ");
scanf("%d", &nota[0]);

printf("Digite a segunda nota: ");
scanf("%d", &nota[1]);

printf("Digite a terceira nota: ");
scanf("%d", &nota[2]);

printf("Digite a quarta nota: ");
scanf("%d", &nota[3]);
```

Explicação resumida:

- Declara um vetor com quatro posições do tipo int.
- Exibe uma mensagem solicitando cada nota a ser digitada.
- Usa scanf() para armazenar a nota digitada em cada posição do vetor.

Simplificando:

```
printf("Digite 4 notas: ");
scanf("%d %d %d %d",&nota[0],&nota[1],&nota[2],&nota[3]);
```

A entrada de dados foi compactada em uma única linha de código. O usuário digita as quatro notas na mesma linha, separadas por espaço, e elas são atribuídas diretamente às posições do vetor.

Esse tipo de preenchimento gera muitas linhas de código repetitivas, mesmo para vetores pequenos. Imagine um vetor com 100 posições: preencher manualmente, sem usar laços de repetição, se torna inviável e pouco eficiente. Por isso, o uso de laços for ou while é a forma mais adequada e escalável para manipular vetores, especialmente quando o tamanho é grande ou definido pelo usuário.

Preenchendo o vetor com um laço de repetição, normalmente utilizamos um laço for.

Por exemplo, com tamanho predefinido e uso de laço for:

```
int i, vetor[10];

for(i = 0; i < 10; i++){
    printf("Digite o %dº valor: ", i+1);
    scanf("%d ", &vetor[i]);
}
```

Esse programa lê 10 números inteiros digitados pelo usuário e armazena cada um deles em um vetor chamado vetor.

Explicação:

- **Declaração das variáveis**: int i, vetor[10];
  - **i**: será usada como contador no for.
  - **vetor[10]**: declara um vetor com 10 posições do tipo int.
- Laço for(i = 0; i < 10; i++) {...}
- Laço que se repete 10 vezes, com i variando de 0 até 9.
- Em cada repetição, temos a impressão de uma mensagem ao usuário e à leitura de um elemento do vetor.
- **printf("Digite o %dº valor: ",i+1);**: exibe a mensagem no formato "Digite o 1º valor: ", "Digite o 2º valor: ", ..., até o 10º valor.

– i +1 é usado para que o usuário veja a numeração começando do 1 (mais intuitivo), pois o índice (i) do vetor começa em 0.

- **scanf("%d",&vetor[i]);**: lê o número digitado pelo usuário e armazena na posição i do vetor.

O programa vai pedir 10 valores ao usuário, um por um, e armazená-los sequencialmente no vetor nas posições vetor[0] até vetor[9].

No código a seguir, temos o laço for e o tamanho definido pelo usuário:

```
int i, tam;

printf("Digite o tamanho do vetor: ");
scanf("%d", &tam);//tam armazena o tamanho do vetor

int vetor[tam];

for(i = 0; i < tam; i++){
    printf("Entre com o valor da posição número %d: ", i+1);
    scanf("%d", &vetor[i]);
}
```

Esse programa permite que o usuário informe quantos valores deseja armazenar em um vetor, ou seja, o tamanho só é conhecido em tempo de execução, conforme o valor digitado pelo usuário. Utilizando um laço de repetição, for permite que o usuário preencha dinamicamente cada posição do vetor.

Vantagem desse código:

- Permite que o tamanho do vetor seja definido em tempo de execução.
- Funciona para qualquer quantidade de elementos que o usuário desejar (desde que a memória permita).
- É dinâmico e escalável, diferentemente de declarações fixas com 3 ou 4 posições.

Saiba mais

Para saber mais a respeito da linguagem C, leia:

BACKES, A. *Linguagem C*: completa e descomplicada. São Paulo: Grupo GEN, 2018.

5.1.2 Impressão de vetor

Imprimir um vetor na linguagem C significa mostrar na tela os valores armazenados em suas posições de memória. Como um vetor contém vários elementos, é necessário acessar cada posição individualmente informando cada uma delas ou utilizando um laço de repetição, normalmente um for.

Durante esse processo, usamos o índice do vetor para indicar qual elemento será exibido e a função printf() para realizar a impressão no console (saída na tela).

Observação

A impressão ou o preenchimento de um vetor com o uso de um laço de repetição dependem do uso de um índice, que começa em 0 e vai até tamanho – 1.

O código a seguir está sem uso de laço de repetição. Ele somente é adotado quando serão impressos poucos elementos do vetor; se o vetor for muito grande, esse processo fica inviável.

```
printf("%d ", vetor[0]);
printf("%d ", vetor[1]);
...
printf("%d ", vetor[n-1]);
```

O uso de laço for é o método mais comum para percorrer o vetor, permitindo acessar e imprimir cada item com precisão, independentemente do tamanho do vetor.

O código a seguir traz a impressão com laço de repetição for: as variáveis i, tamanho e o vetor devem ser declaradas antes do for. Observe:

```
int i, tamanho = 5;
int vetor[tamanho];

for(i = 0; i < tamanho; i++){
    printf("%d ", vetor[i]);
}
```

O código a seguir ilustra a impressão em ordem inversa com laço de repetição for:

```
int i, tamanho = 5;
int vetor[tamanho];

for(i = tamanho; i >= 0; i--){
    printf("%d ", vetor[i]);
}
```

Também é possível permitir que o usuário digite o tamanho do vetor, conforme já visto anteriormente.

Lembrete

O índice de um vetor em C sempre começa em 0 e vai até tamanho – 1.

Use laços for para percorrer e preencher vetores de forma eficiente. Evite acessar posições fora dos limites do vetor, pois C não faz verificação automática de limites, isso pode causar erros de execução graves.

5.2 Operações com vetores

Vetores são estruturas que permitem armazenar vários valores, bem como realizar diversas operações com seus dados. Em C, as operações com vetores envolvem acessar, manipular e analisar os elementos armazenados, geralmente com o auxílio de laços de repetição.

No código a seguir, o programa preenche manualmente um vetor e imprime seus valores em ordem crescente com um com laço for.

```c
#include <stdio.h>

int main(){
    int i, tamanho = 3;
    int vetor[tamanho];

    vetor[0] = 1;
    vetor[1] = 2;
    vetor[2] = 3;
    //ou sem a variável tamanho
    //int vetor[5]={1, 2, 3};

    for(i = 0; i < tamanho; i++){
        printf("%d ", vetor[i]);
    }

    return 0;
}
```

Saída na tela:

```
1 2 3
```

Figura 33

Explicação:

- Esse programa declara um vetor com tamanho variável, preenche suas posições manualmente com valores inteiros e, em seguida, imprime esses valores usando um laço for.
- O laço for percorre o vetor de i = 0 até i < tamanho (0, 1, 2).
- Imprime os valores do vetor, um após o outro, com espaço entre eles.

No próximo código, temos a impressão do vetor em **ordem inversa**. Note que o vetor não foi alterado, somente a impressão está em ordem inversa.

```c
#include <stdio.h>

int main(){
    int i, tamanho = 3;
    int vetor[tamanho];

    vetor[0] = 1; vetor[1] = 2; vetor[2] = 3;
    //ou sem a variável tamanho
    //int vetor[5]={1, 2, 3};

    for(i = tamanho-1; i >= 0; i--){
        printf("%d ", vetor[i]);
    }

    return 0;
}
```

Saída na tela:

```
3 2 1
```

Figura 34

Esse programa é um exemplo simples que mostra como imprimir um vetor de trás para frente (ordem inversa). Inicialmente, definimos um vetor com três posições, preenchendo manualmente cada posição com um valor. Para tal, utiliza um laço for para imprimir os elementos em ordem reversa.

- **Laço para impressão em ordem inversa**:
  - Começa em i = tamanho - 1;, (última posição do vetor), ou seja, i = 2.
  - Enquanto i for maior ou igual a 0, imprime vetor[i].
  - A cada passo, i é decrementado em 1.

No código a seguir, temos a leitura e a exibição de vetor com for. Esse programa permite que o usuário digite cinco números inteiros, que serão armazenados em um vetor. Após a leitura dos valores, o programa exibe todos os números digitados na tela, em sequência e separados por espaço.

```c
#include <stdio.h>

int main(){
    int i, vetor[5];
    //Primeiro laço for - Leitura dos valores
    for(i = 0; i < 5; i++){
        printf("Digite o %dº valor: ", i+1);
        scanf("%d", &vetor[i]);
    }
    printf("\n"); //para separar a entrada da saída

    //Segundo laço for - Impressão dos valores
    for(i = 0; i < 5; i++){
        printf("%d ", vetor[i]);
    }

    return 0;
}
```

O programa solicita ao usuário que digite cinco números inteiros e armazena esses valores em um vetor. Em seguida, imprime todos os números digitados, um ao lado do outro.

- **Primeiro laço for**: leitura dos valores. Esse laço executa cinco vezes (de i = 0 até i = 4). Em cada repetição:
  - Solicita ao usuário um número.
  - Armazena esse número na posição vetor[i].
  - O uso de i + 1 serve apenas para mostrar 1º, 2º etc.
- **Segundo laço for**: impressão dos valores
  - Esse laço também percorre o vetor.
  - Em cada repetição, imprime o valor armazenado na posição i.

No código a seguir, temos a soma dos elementos de um vetor. Esse programa permite que o usuário digite cinco números inteiros, que serão armazenados em um vetor. Após a leitura dos valores, o programa realiza a soma de todos os elementos do vetor e exibe o resultado final na tela.

```c
#include <stdio.h>

int main() {
    int vetor[5], i, soma = 0;

    // Leitura dos valores
    for(i=0; i<5; i++){
        printf("Digite o %dº valor: ", i+1);
        scanf("%d", &vetor[i]);
    }

    // Cálculo da soma
    for(i=0; i<5; i++){
        soma = soma+vetor[i];//soma += vetor[i];
    }

    // Exibição do resultado (soma)
    printf("\nSoma: %d\n", soma);

    return 0;
}
```

A operação de soma é realizada utilizando um laço for, que percorre todas as posições do vetor e acumula os valores em uma variável auxiliar chamada soma.

- **Declaração das variáveis do tipo inteiro (int)**:
  - **vetor[5]**: vetor de cinco posições para armazenar os números.
  - **i**: variável auxiliar usada nos laços for.
  - **soma**: inicializada com 0, será usada para acumular os valores do vetor.
- **Primeiro laço for**: faz a leitura dos valores.
- **Segundo laço for**: faz a soma dos valores.

O código a seguir traz a média dos elementos de um vetor. Esse programa permite que o usuário digite cinco números inteiros, que serão armazenados em um vetor. Após a leitura dos valores, o programa realiza a soma de todos os elementos e calcula a média aritmética, exibindo o resultado final com duas casas decimais na tela.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");

    int vetor[5], i, soma = 0;
    float media;

    //Leitura dos valores
    for(i = 0; i < 5; i++){
        printf("Digite o %dº valor: ", i+1);
        scanf("%d", &vetor[i]);
    }

    //Soma dos elementos
    for(i = 0; i < 5; i++){
        soma += vetor[i];
    }

    media = soma / 5.0; //Cálculo da média

    //Exibição da média
    printf("\nMédia: %.2f\n", media);

    return 0;
}
```

Observe a seguir a explicação simplificada:

- O vetor guarda cinco valores digitados pelo usuário.
- Um laço for soma todos os valores.
- A média é calculada dividindo a soma por 5.0 (usamos 5.0 com ponto para garantir divisão real).
- O programa exibe a média com duas casas decimais usando %.2f.

O código a seguir busca elemento com vetor pré-preenchido. Esse programa tem um vetor preenchido automaticamente com cinco valores inteiros fixos. O usuário informa um número e o programa verifica se esse valor está presente no vetor, exibindo uma mensagem correspondente.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");

    int vetor[5] = {4, 8, 15, 16, 23};
    int i, valor, encontrado = 0;

    //Exibe os valores do vetor
    printf("Vetor: ");
    for(i = 0; i < 5; i++) {
        printf("%d ", vetor[i]);
    }

    //Entrada do valor a ser buscado
    printf("\n\nDigite um valor para buscar: ");
    scanf("%d", &valor);

    // Busca no vetor
    for(i = 0; i < 5; i++){
        if(vetor[i] == valor){
            encontrado = 1;
            break;
        }
    }

    //Imprime Resultado
    if(encontrado)
        printf("Valor encontrado no vetor!\n");

    else
        printf("Valor NÃO encontrado.\n");

    return 0;
}
```

Explicação:

- **Declaração das variáveis do tipo inteiro (int)**:
  - **vetor[5]**: vetor de cinco posições preenchido com valores fixos.
  - **i**: variável de controle do laço for.
  - **valor**: número que o usuário deseja procurar.
  - **encontrado**: variável usada como indicador (0 igual não encontrado, 1 igual encontrado).
- **Primeiro laço for**: exibe todos os valores armazenados no vetor.
- **Segundo laço for**: o laço percorre as posições do vetor.
  - Em cada passo, o valor do vetor[i] é comparado com a variável valor. Se forem iguais:
    - A variável encontrado recebe o valor 1.
    - O break interrompe o laço, pois o valor já foi localizado.
- **Impressão do resultado da busca do valor**:
  - if(encontrado) printf("Valor encontrado no vetor!\n");
  - else printf("Valor NÃO encontrado.\n");
  - Após a busca, o programa verifica o valor da variável encontrado.
  - Se for 1, exibe que o valor foi encontrado.
  - Se continuar 0, mostra que o valor não existe no vetor.

Resumo:

- O vetor já vem preenchido com {4, 8, 15, 16, 23}.
- O usuário digita um valor para procurar.
- O programa percorre o vetor comparando os elementos.
- Informa se o valor foi encontrado ou não.

No exemplo a seguir, temos um código que traz o maior valor em um vetor. Esse programa permite que o usuário digite cinco números inteiros, armazena esses valores em um vetor e, em seguida, verifica qual é o maior valor presente entre os elementos digitados. Ao final, o programa exibe o maior número encontrado.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");

    int vetor[5], i, maior;

    //Leitura dos valores
    for(i = 0; i < 5; i++){
        printf("Digite o %dº valor: ", i+1);
        scanf("%d", &vetor[i]);
    }

    //Inicializa maior com o primeiro valor do vetor
    maior = vetor[0];

    //Verifica os demais valores
    for(i = 1; i < 5; i++){
        if(vetor[i] > maior){
            maior = vetor[i];
        }
    }

    //Exibição do maior valor
    printf("\nMaior valor: %d\n", maior);

    return 0;
}
```

Observe a seguir a explicação simplificada:

- O vetor armazena cinco valores digitados pelo usuário.
- A variável maior começa com o valor da primeira posição do vetor.
- O laço for percorre os valores restantes. Se encontrar um valor maior, atualiza a variável maior.
- Ao final, imprime o maior valor encontrado.

Agora temos um código com o menor valor em um vetor. Esse programa permite que o usuário digite cinco números inteiros, armazena esses valores em um vetor e, em seguida, verifica qual é o menor valor presente entre os elementos digitados. Ao final, o programa exibe o menor número encontrado.

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

int main(){
    setlocale(LC_ALL, "Portuguese");

    int vetor[5], i, menor;

    //Leitura dos valores
    for(i = 0; i < 5; i++){
        printf("Digite o %d° valor: ", i+1);
        scanf("%d", &vetor[i]);
    }

    //Inicializa menor com o primeiro valor do vetor
    menor = vetor[0];

    //Verifica os demais valores
    for(i = 1; i < 5; i++){
        if(vetor[i] < menor){
            menor = vetor[i];
        }
    }

    //Exibição do menor valor
    printf("\nMenor valor: %d\n", menor);

    return 0;
}
```

A explicação é similar ao programa do código apresentado anteriormente:

- O vetor recebe cinco números do usuário.
- A variável menor começa com o valor da primeira posição.
- O laço percorre os outros valores. Se encontrar um valor menor, atualiza a variável menor.
- Ao final, exibe o menor valor encontrado.

No exemplo a seguir, o código conta quantos números pares existem em um vetor. Esse programa permite que o usuário digite 10 números inteiros, armazena esses valores em um vetor e, ao final, informa quantos desses números são pares.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");

    int i, pares=0, vetor[10];

    // Leitura dos valores
    for(i=0; i<10; i++){
        printf("Digite o %dº valor: ", i+1);
        scanf("%d", &vetor[i]);
    }

    // Verifica quantos valores são pares
    for(i=0; i<10; i++){
        if(vetor[i] % 2 == 0){
            pares++;
        }
    }

    //Impressão do Resultado
    printf("\nQuantidade de números pares: %d\n", pares);

    return 0;
}
```

Observe a seguir a explicação simplificada:

- **Declaração de variáveis do tipo inteiro (int)**:
  - **vetor[10]**: vetor com 10 posições.
  - **i**: variável índice no for.
  - **pares**: contador que será incrementado sempre que um número par for encontrado no vetor.
- **Primeiro laço**: faz a leitura dos valores.
- O segundo laço percorre o vetor e conta quantos elementos são pares. Se vetor[i] % 2 == 0, então o número é par.
- Ao final, mostra a quantidade de pares encontrados.

O código a seguir traz um programa para criar um vetor com tamanho n definido pelo usuário e preenche o vetor com o valor de i * 2 em cada posição. Depois, imprime o vetor em ordem normal; em seguida, em ordem decrescente.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");
    int n, i;

    printf("Digite o tamanho do vetor: ");
    scanf("%d", &n);

    int vetor[n];

    //Preenchendo o vetor com valor de i*2
    for(i=0; i<n; i++) {
        vetor[i] = i*2;
    }

    //Impressão em ordem normal
    printf("\nVetor em ordem normal:\n");
    for(i=0; i<n; i++){
        printf("%d ", vetor[i]);
    }

    //Impressão em ordem decrescente
    printf("\n\nVetor em ordem inversa:\n");
    for(i=n-1; i>=0; i--){
        printf("%d ", vetor[i]);
    }

    printf("\n");

    return 0;
}
```

Observe a seguir a explicação simplificada:

- O usuário informa o valor de n. O vetor é criado com esse tamanho.
- Cada posição recebe o dobro do seu índice (i * 2).
- O primeiro laço for imprime em ordem normal crescente.
- O segundo laço for imprime em ordem decrescente.

O exemplo a seguir traz um código com a lista de produtos e preços. Esse programa permite que o usuário digite os nomes e preços de cinco produtos. Após a entrada dos dados, ele mostra a lista completa com os produtos e seus respectivos preços. Por fim, calcula o valor total da compra.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");
    //Vetor de strings (matriz): 5 produtos com até 29 letras + '\0'
    char produtos[5][30];
    float precos[5]; //Vetor de preços
    float total = 0;
    int i;

    // Leitura dos produtos e preços
    for(i=0; i<5; i++){
        printf("Digite o nome do %dº produto: ", i+1);
        // lê até espaço (usa espaço antes para limpar o buffer)
        scanf(" %[^\n]", produtos[i]);

        printf("Digite o preço do produto %s: R$ ", produtos[i]);
        scanf("%f", &precos[i]);

        total = total+precos[i]; // acumula no total
    }

    // Exibição dos produtos e preços
    printf("\nLista de produtos:\n");
    for(i=0; i<5; i++){
        printf("%d. %s - R$ %.2f\n", i + 1, produtos[i], precos[i]);
    }

    // Total da compra
    printf("\nTotal da compra: R$ %.2f\n", total);

    return 0;
}
```

Observe a seguir a explicação simplificada:

- **produtos[5] [30];**: guarda os nomes dos cinco produtos.
- **precos[5];**: guarda os preços desses produtos.
- **total;**: soma todos os preços para dar o valor final da compra.
- **Primeiro laço for**: é usado para ler os nomes dos produtos; ler os preços; somar os preços no total.

- **scanf("%[^\n]",...);**: permite que o usuário digite nomes com espaço.
- **Segundo laço for**: mostra nome e o preço de cada produto.
- **printf final**: mostra quanto deu o total da compra.

Acentua-se que é preciso seguir as boas práticas. Na maioria dos programas estruturados, a recomendação é:

- **Separar entrada e saída**
  - Primeiro, preencha o vetor com os dados.
  - Depois, use outro laço apenas para processar ou exibir os valores.
- **Inicializar variáveis acumuladoras com um valor exemplo (soma=0, total=0, contador=0)**
- **Usar constantes ou variáveis para o tamanho do vetor**
  - Evite usar números fixos (como 10, 100) diretamente no código.
- **Não acessar posições fora do limite**
  - O índice do vetor começa em 0.
  - Em um vetor de tamanho 5, os índices válidos vão de 0 a 4.
  - Acessar vetor[5] causará erro de execução (estouro de memória).
- **Comentários claros**
  - Comente o que o vetor representa (exemplo: notas, preços, nomes).
  - Isso ajuda entender melhor o código no futuro.

6 MATRIZES

6.1 Matrizes (arrays bidimensionais)

Em linguagem de programação C, matrizes são vetores com duas dimensões: linhas e colunas. Elas são usadas para representar tabelas, grades, mapas ou qualquer estrutura que envolva dois eixos (linha, coluna) como: (mapas e tabuleiros).

Matriz é um tabuleiro com linhas e colunas. Ideal para mapas, fases e batalhas.

Observe a seguir a sintaxe da declaração de matriz:

```
tipo nome Matriz[tamanho1Linha] [tamanho2Coluna];
```

```
int matriz1 [4][3];

double matriz2[4][3];
```

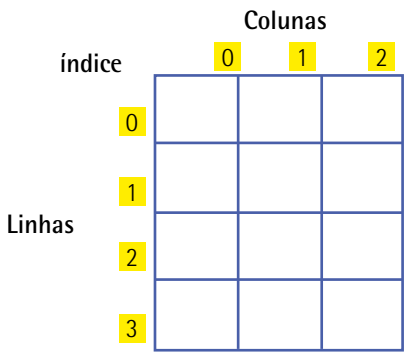
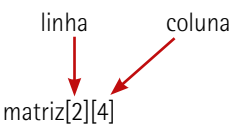


Figura 35 – Exemplo da estrutura da matriz: o exemplo mostra uma representação visual de uma matriz bidimensional em C. Em uma matriz do tipo matriz[lin][col], o primeiro índice representa a linha e o segundo, a coluna

matriz[2][2]

	coluna0	coluna1
linha0	matriz[0][0]	matriz[0][1]
linha1	matriz[1][0]	matriz[1][1]



	coluna0	coluna1	coluna2	coluna3
linha0	matriz[0][0]	matriz[0][1]	matriz[0][2]	matriz[0][3]
linha1	matriz[1][0]	matriz[1][1]	matriz[1][2]	matriz[1][3]

Figura 36

Primeiro acessamos a linha, depois a coluna:

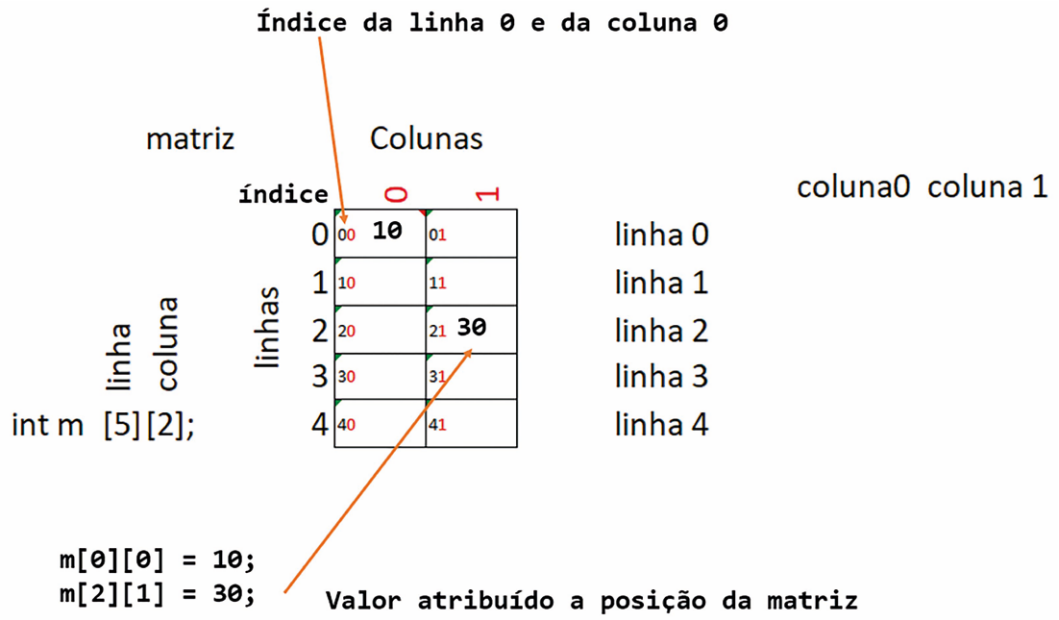


Figura 37

Matriz é um vetor de vetores. Internamente, a matriz é um vetor unidimensional, em que cada elemento é um vetor unidimensional.

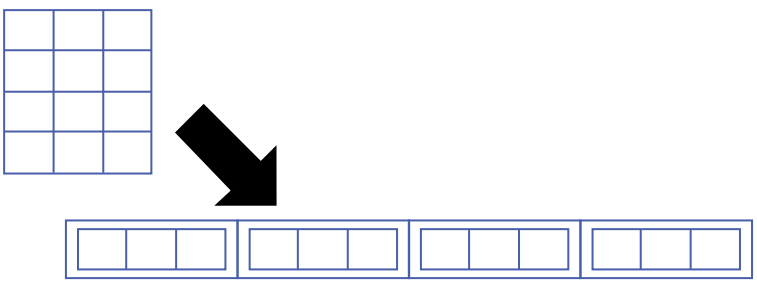


Figura 38

Exemplo de percurso em matrizes:

```
int matriz[4][3];
int i, j;
for(i = 0; i < 4; i++)
    for(j = 0; j < 3; j++)
        matriz[i][j];
```

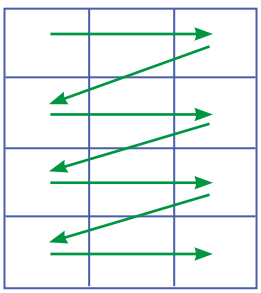


Figura 39

A lógica de for aninhado na matriz funciona assim:

- O primeiro for externo controla qual linha está sendo manipulada.
- O segundo for interno percorre todas as colunas daquela linha.

6.2 Manipulação de matrizes: acesso e operações comuns

O exemplo a seguir traz um programa para imprimir todos os elementos de uma matriz com valores predefinidos na tela, formatando em forma de tabela (linhas e colunas).

```
#include <stdio.h>

int main() {
    int m[3][4] = {{3,4,5,5},{1,3,6,5},{8,1,2,5}};

    int i, j;
    for(i=0; i<3; i++){
        for(j=0; j<4; j++){
            //imprimindo a matriz
            printf("%d ", m[i][j]);
        }
         printf("\n");
    }
    return 0;
}
```

Explicação:

- Cria-se uma matriz com três linhas e quatro com valores predefinidos, como int m[3][4] = {{3,4,5,5},{1,3,6,5},{8,1,2,5}}

Observe a ilustração da matriz:

Tabela 8

	Coluna 0	Coluna 1	Coluna2	Coluna 3
Linha 0	3	4	5	5
Linha 1	1	3	6	5
Linha 2	8	1	2	5

- O laço externo (i) percorre as linhas da matriz.
- O laço interno (j) percorre as colunas de cada linha.
- printf("%d ",m[i][j]); imprime cada valor da linha.
- printf("\n"); separa visualmente as linhas da matriz.

Saída na tela:

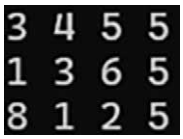


Figura 40

O programa a seguir cria uma matriz com cinco linhas e duas colunas. Em seguida, preenche cada posição da matriz com a soma do número da linha com o número da coluna (ou seja, i + j). Por fim, o programa imprime os valores da matriz.

```c
#include <stdio.h>

int main() {
    int i, j, lin = 5, col = 2;
    int matriz[lin][col];

    //Quando há apenas um comando dentro do for,
    //não é necessário usar chaves { }
    for(i=0; i<lin; i++) // percorre cada linha
        for(j=0; j<col; j++) //percorre cada coluna
            matriz[i][j] = i+j; //preenche a matriz com a soma dos índices

    for(i = 0; i < lin; i++){ //percorre cada linha
        for(j = 0; j < col; j++){//percorre cada coluna
            printf("%d ", matriz[i][j]);//imprime o valor da posição atual
        }
        printf("\n");//quebra de linha após imprimir cada linha da matriz
    }
    return 0;
}
```

- **Declaração de variáveis**:

– **int lin = 5.col = 2**: definem o tamanho da matriz (cinco linhas e duas colunas).

– **int matriz[lin][col];**: matriz do tipo int com cinco linhas e duas colunas.

– **int i, j;: i e j**: variáveis para os laços for.

- O primeiro par de for preenche a matriz com valores de (i + j).

- O segundo par de for exibe a matriz no formato de tabela.

Saída na tela:

Figura 41

O exemplo ilustra um programa somando todos os elementos de uma matriz com valores predefinidos e exibindo o resultado na tela.

```c
#include <stdio.h>

int main() {
    int i, j, soma=0;

    int m[3][4]={{1, 2, 3, 4},
                 {1, 1, 1, 1},
                 {4, 3, 2, 1}
    };

    for(i=0; i<3; i++){
        for(j=0; j<4; j++){
            soma += m[i][j];
        }
    }
    printf("Soma total = %d\n", soma);

    return 0;
}
```

- **Declaração das variáveis**:
  - int i, j, soma = 0;
  - **i e j**: são variáveis usadas para percorrer a matriz.
  - **soma**: acumulador que guardará a soma de todos os elementos da matriz.
- **int m[3][4]={}**: cria uma matriz m com três linhas e quatro colunas com valores predefinidos.
- O primeiro for de i percorre as linhas (de 0 a 2).
- O segundo for de j percorre as colunas (de 0 a 3).
- Em cada passo, o valor da posição m[i][j] é somado ao total acumulado na variável soma. Por fim, imprime-se o resultado de soma.
- O programa percorre cada valor e acumula na variável soma.

Saída na tela:

```
Soma total = 24
```

Figura 42

Agora temos um programa que percorre uma matriz 3x4 de inteiros e calcula a média aritmética dos valores de cada linha. Para cada linha, o programa soma os quatro elementos e exibe a média com duas casas decimais.

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

int main() {
    setlocale(LC_ALL, "Portuguese");

    int i, j, somaLinha;
    int m[3][4]={{1, 2, 3, 4},
                 {1, 1, 1, 1},
                 {4, 3, 2, 1}
    };

    for(i=0; i<3; i++) {
        somaLinha = 0;
        for(j=0; j<4; j++) {
            somaLinha += m[i][j];
        }
        float media = somaLinha / 4.0;
        printf("Média da linha %d = %.2f\n", i+1, media);
    }

    return 0;
}
```

Dentro do for(i), o programa percorre as linhas em cada iteração:

- Primeiro, a variável somaLinha é zerada.
- Em seguida, o for(j) (laço interno) percorre as colunas da linha atual:
  - **somaLinha += m[i][j]**; soma os quatro valores da linha.
- **Após sair do for(j) e retornar ao for(i)**:
  - A média é calcula e exibida com duas casas decimais, e a numeração da linha é ajustada para começar em 1 (i + 1).

Saída na tela:

```
Média da linha 1 = 2,50
Média da linha 2 = 1,00
Média da linha 3 = 2,50
```

Figura 43

O código a seguir traz notas de alunos com média. O programa em C é usado para armazenar as notas de quatro alunos, sendo que cada um tem três notas. O programa deve calcular e exibir a média de cada aluno.

```c
#include <stdio.h>
#include <locale.h>

int main() {
    setlocale(LC_ALL, "Portuguese");
    //matriz com 4 linhas e 3 colunas
    //cada linha representa um aluno, cada coluna uma nota
    float notas[4][3], soma, media;
    int i, j;

    // Leitura das notas
    for(i = 0; i < 4; i++) {
        printf("Aluno %d:\n", i+1);
        for(j=0; j<3; j++) {
            printf("Digite a %dª nota: ", j + 1);
            scanf("%f", &notas[i][j]);
        }
    }

    // Cálculo e exibição da média de cada aluno
    printf("\nMédias dos alunos:\n");
    for(i=0; i<4; i++) {
        soma = 0;
        for(j=0; j<3; j++) {
            soma += notas[i][j];
        }
        media = soma / 3.0;
        printf("Aluno %d: Média = %.2f\n", i+1, media);
    }

    return 0;
}
```

Explicação resumida:

- A matriz notas [4][3] guarda quatro linhas (alunos) e três colunas (notas).
- O primeiro for lê todas as notas dos alunos.

- O segundo for calcula a soma e a média de cada linha (aluno).
- A média é exibida com duas casas decimais.

Para a saída na tela, a simulação de execução do programa é similar ao exemplo acentuado anteriormente.

```
Aluno 1:
Digite a 1ª nota: 7
Digite a 2ª nota: 8
Digite a 3ª nota: 9
Aluno 2:
Digite a 1ª nota: 5
Digite a 2ª nota: 6
Digite a 3ª nota: 7
Aluno 3:
Digite a 1ª nota: 8
Digite a 2ª nota: 6
Digite a 3ª nota: 6
Aluno 4:
Digite a 1ª nota: 4
Digite a 2ª nota: 6
Digite a 3ª nota: 5

Médias dos alunos:
Aluno 1: Média = 8,00
Aluno 2: Média = 6,00
Aluno 3: Média = 6,67
Aluno 4: Média = 5,00
```

Figura 44

Podemos ampliar esse programa para representar um caso mais realista no qual, além de calcular a média de cada aluno, informamos o total de alunos aprovados e o total de reprovados.

Para isso, será necessário:

- Calcular a média de cada aluno.
- Verificar se a média for maior ou igual a 7.0, o aluno é considerado aprovado; caso contrário, será reprovado.
- Acumular a quantidade de alunos aprovados e reprovados e exibir esses totais ao final do programa.

O exemplo a seguir representa essa melhoria no programa. Ele é similar ao acentuado anteriormente: lê as notas de quatro alunos e calcula a média individual de cada um. Em seguida, mostra a média de cada aluno com duas casas decimais. No final do programa, informa quantos alunos foram aprovados (média ≥ 7) e quantos foram reprovados (média < 7).

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL,"Portuguese");

    float notas[4][3], soma, media;
    int i, j, aprovados=0, reprovados=0;

    //Leitura das notas
    for(i=0; i<4; i++){
        printf("Aluno %d:\n", i+1);
        for(j=0;j<3;j++){
            printf("Digite a %dª nota: ",j+1);
            scanf("%f",&notas[i][j]);
        }
    }

    //Cálculo das médias e contagem
    printf("\nMédias dos alunos:\n");
    for(i=0;i<4;i++){
        soma=0;
        for(j=0;j<3;j++){
            soma = soma+notas[i][j];
        }
        media=soma/3.0;
        printf("Aluno %d: Média = %.2f\n",i+1,media);

        if(media>=7.0){
            aprovados++;
        }
        else{
            reprovados++;
        }
    }

    //Impressão do resultado final
    printf("\nTotal de alunos aprovados: %d\n",aprovados);
    printf("Total de alunos reprovados: %d\n",reprovados);

    return 0;
}
```

Esse programa lê três notas para cada um dos quatro alunos, calcula a média individual de cada aluno e mostra o resultado.

Depois, o programa verifica se a média é maior ou igual a 7:

- Se for verdadeiro, o aluno é considerado aprovado.
- Caso contrário, é reprovado.

Ao final, o programa exibe:

- A média de cada aluno.
- O total de alunos aprovados.
- O total de alunos reprovados.

Saída na tela:

```
Aluno 1:
Digite a 1ª nota: 6
Digite a 2ª nota: 7
Digite a 3ª nota: 8
Aluno 2:
Digite a 1ª nota: 5
Digite a 2ª nota: 4
Digite a 3ª nota: 6
Aluno 3:
Digite a 1ª nota: 8
Digite a 2ª nota: 9
Digite a 3ª nota: 10
Aluno 4:
Digite a 1ª nota: 4
Digite a 2ª nota: 2
Digite a 3ª nota: 8

Médias dos alunos:
Aluno 1: Média = 7,00
Aluno 2: Média = 5,00
Aluno 3: Média = 9,00
Aluno 4: Média = 4,67

Total de alunos aprovados: 2
Total de alunos reprovados: 2
```

Figura 45

O programa acentuado a seguir lê três notas para cada um dos quatro alunos, calcula a média individual de cada aluno e mostra o resultado.

```c
#include <stdio.h>
#include <locale.h>

#define NUM_ALUNOS 4
#define NUM_NOTAS 3
#define MEDIA_APROVACAO 7.0

int main() {
    setlocale(LC_ALL, "Portuguese");

    float notas[NUM_ALUNOS][NUM_NOTAS], soma, media;
    int i, j, aprovados = 0, reprovados = 0;

    //Leitura das notas
    for(i = 0; i < NUM_ALUNOS; i++){
        printf("Aluno %d:\n", i + 1);
        for(j = 0; j < NUM_NOTAS; j++){
            printf("Digite a %dª nota: ", j + 1);
            scanf("%f", &notas[i][j]);
        }
    }

    //Cálculo das médias
    printf("\nMédias dos alunos:\n");
    for(i = 0; i < NUM_ALUNOS; i++){
        soma = 0;
        for(j = 0; j < NUM_NOTAS; j++){
            soma += notas[i][j];
        }
        media = soma / NUM_NOTAS;
        printf("Aluno %d: Média = %.2f\n", i + 1, media);

        if(media >= MEDIA_APROVACAO){
            aprovados++;
        } else {
            reprovados++;
        }
    }

    //Resultado final
    printf("\nTotal de alunos aprovados: %d\n", aprovados);
    printf("Total de alunos reprovados: %d\n", reprovados);

    return 0;
}
```

O uso da diretiva #define permite criar constantes simbólicas, o que:

- Melhora a clareza (o código se torna mais autoexplicativo).
- Facilita a manutenção (basta mudar o valor em um único lugar).
- Evita repetição desnecessária de números mágicos no código.

Benefícios diretos:

- Se a escola mudar para cinco notas por aluno, você só altera o valor de NUM_NOTAS.
- Se o critério de aprovação mudar para 6.0, só muda MEDIA_APROVACAO.
- O código fica mais legível e profissional.

Observação

#define não cria uma variável nem usa ponto e vírgula no final, é apenas uma substituição de texto simbólica feita pelo pré-processador.

Saída na tela:

```
Aluno 1:
Digite a 1ª nota: 8
Digite a 2ª nota: 7
Digite a 3ª nota: 4
Aluno 2:
Digite a 1ª nota: 8
Digite a 2ª nota: 9
Digite a 3ª nota: 7
Aluno 3:
Digite a 1ª nota: 4
Digite a 2ª nota: 5
Digite a 3ª nota: 6
Aluno 4:
Digite a 1ª nota: 7
Digite a 2ª nota: 5
Digite a 3ª nota: 6

Médias dos alunos:
Aluno 1: Média = 6,33
Aluno 2: Média = 8,00
Aluno 3: Média = 5,00
Aluno 4: Média = 6,00

Total de alunos aprovados: 1
Total de alunos reprovados: 3
```

Figura 46

O exemplo a seguir é de busca: o programa em C verifica se um valor existe na matriz. Ele também indica em qual posição (linha e coluna) o valor foi encontrado ou informa que ele não existe na matriz.

```c
#include <stdio.h>
#include <locale.h>

#define LINHAS 3
#define COLUNAS 3

int main(){
    setlocale(LC_ALL, "Portuguese");

    int i, j;
    int matriz[LINHAS][COLUNAS] = {
        {10, 20, 30},
        {40, 50, 60},
        {70, 80, 90}
    };

    int valor, encontrado = 0;

    printf("Digite um valor para buscar na matriz: ");
    scanf("%d", &valor);

    // Busca pelo valor na matriz
    for(i=0; i<LINHAS; i++){
        for(j=0; j<COLUNAS; j++){
            if(matriz[i][j]==valor){
                printf("Valor %d encontrado\n", valor);
                printf("Posição [%d][%d]\n", i, j);
                encontrado = 1;
            }
        }
    }

    if(encontrado==0){
        printf("Valor %d não encontrado na matriz.\n", valor);
    }

    return 0;
}
```

São utilizados os comandos #define para declarar as constantes linhas e colunas, que representam, respectivamente, o número de linhas e colunas da matriz. O programa tem a seguinte sequência:

- Solicita ao usuário um valor para buscar.
- Percorre toda a matriz usando dois laços for.
- Se encontrar o valor, mostra a posição [linha][coluna].
- Se não encontrar, avisa o usuário.
- Trabalha com matriz bidimensional.
- Usa #define para deixar o código mais legível e flexível.
- Aplica laços aninhados para percorrer a matriz.
- Utiliza condição simples (if) para comparar e identificar valores.

Observação

Em alguns exemplos, optamos por utilizar a diretiva #define para definir constantes, enquanto em outros isso não foi aplicado. Essa escolha foi proposital, com o objetivo de permitir ao aluno experimentar diferentes abordagens de codificação.

No código a seguir, por exemplo, o tamanho da matriz é definido pelo próprio usuário em tempo de execução, possibilitando maior flexibilidade e interação com o programa, além de reforçar o conceito de alocação baseada em entrada dinâmica.

O código conta os valores pares e ímpares em uma matriz. Lê os elementos de uma matriz com dimensões definidas pelo usuário e, ao final, mostra quantos valores pares e quantos ímpares foram digitados.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");

    int linhas, colunas, i, j, pares = 0, impares = 0;

    // Entrada do tamanho da matriz
    printf("Digite o número de linhas da matriz: ");
    scanf("%d", &linhas);

    printf("Digite o número de colunas da matriz: ");
    scanf("%d", &colunas);

    // Declaração da matriz com tamanho definido pelo usuário (VLA)
    int matriz[linhas][colunas];

    // Leitura dos valores da matriz
    printf("\nDigite os valores da matriz %dx%d:\n", linhas, colunas);
    for(i=0; i<linhas; i++){
        for(j=0; j<colunas; j++){
            printf("Elemento [%d][%d]: ", i+1, j+1);
            scanf("%d", &matriz[i][j]);

            // Contagem de pares e ímpares
            if(matriz[i][j] % 2 == 0){
                pares++;
            }else{
                impares++;
            }
        }
    }

    // Exibição dos resultados
    printf("\nTotal de números pares: %d\n", pares);
    printf("Total de números ímpares: %d\n", impares);

    return 0;
}
```

O programa usa dois laços aninhados para percorrer toda a matriz e lê nove números após cada leitura:

- **linhas e colunas** : armazenam o tamanho da matriz digitado pelo usuário.
- **i e j**: usados como índices nos laços for.
- **pares e impares**: contadores iniciados em 0.
- O primeiro for percorre as linhas e o segundo percorre as colunas.
- Para cada posição da matriz, o valor digitado é lido e testado:
  - Usa (matriz[i][j] % 2 == 0) para verificar se é par ou ímpar.
  - Se for divisível por 2, é par – incrementa pares.
  - Caso contrário, é ímpar – incrementa ímpares.
- Soma a quantidade de cada tipo e exibe ao final.

Observação

Alocação variável (VLA): a linha int matriz[linhas][colunas]; só funciona se o compilador suportar o padrão C99 ou superior.

No exemplo a seguir, o programa lê os elementos de uma matriz 3x3 (valores digitados pelo usuário) e, ao final, mostra qual foi o maior valor e qual foi o menor valor encontrado na matriz.

```c
#include <stdio.h>
#include <locale.h>

int main(){
    setlocale(LC_ALL, "Portuguese");

    int matriz[3][3];
    int i, j, maior, menor;

    printf("Digite os valores da matriz 3x3:\n");

    // Leitura da matriz
    for(i=0; i<3; i++){
        for(j=0; j<3; j++){
            printf("Elemento [%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);
        }
    }
    maior = menor = matriz[0][0];
    for(i=0; i<3; i++){
        for(j=0; j<3; j++){
            if(matriz[i][j] > maior)
                maior = matriz[i][j];
            if(matriz[i][j] < menor)
                menor = matriz[i][j];
            }
        }

    // Exibição dos resultados
    printf("\nMaior valor da matriz: %d\n", maior);
    printf("Menor valor da matriz: %d\n", menor);

    return 0;
}
```

Observe a seguir a explicação simplificada:

- O primeiro par de laço for percorre todas as posições da matriz e lê os nove valores digitados pelo usuário.
- O segundo par de laço for percorre novamente toda a matriz:
  - Se um valor for superior ao maior registrado até o momento, ele é atualizado.
  - Se for inferior ao menor, também é atualizado.
- Por fim, o programa mostra na tela os valores finalizados após as comparações.

Saída na tela: simulação com valores digitados pelo usuário.

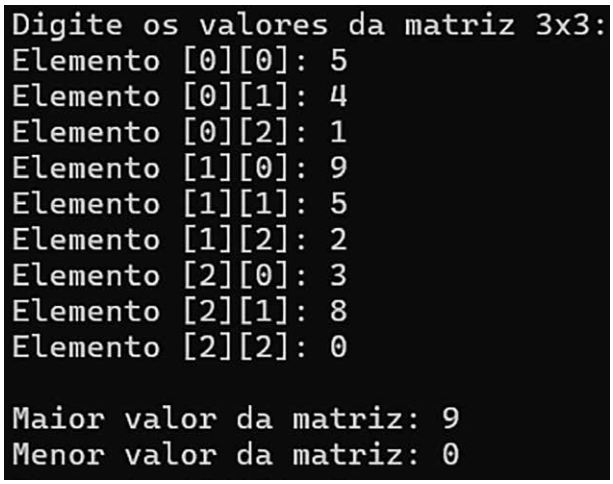


Figura 47

Lembrete

Matriz é uma estrutura de dados em forma de tabela (duas dimensões: linhas e colunas). Em C, é representada como um array bidimensional.

Matriz é muito útil quando se deseja representar dados tabulados, como notas de alunos, mapas, tabelas de preços, jogos e outros cenários onde dois critérios se combinam. A declaração de uma matriz em C segue o padrão: tipo nome[linhas][colunas];

Por exemplo, int matriz[3][4]; cria uma matriz de inteiros com três linhas e quatro colunas.

Entre as operações mais comuns com matrizes, podemos citar: a soma de todos os elementos, o cálculo de média por linha ou coluna, a busca de valores específicos, a identificação do maior e do menor valor, a soma entre duas matrizes, tabelas de produtos e até aplicações lúdicas como jogos.

Saiba mais

Consulte a obra indicada a seguir e aprenda mais sobre a linguagem C:

KERNIGHAN, B. W.; RITCHIE, D. M. *C completo e total*. 2. ed. São Paulo: Pearson Makron Books, 2004.

Resumo

Nesta unidade, estudamos as estruturas de dados, vetores e matrizes na linguagem C.

Vetores são arrays unidimensionais que armazenam vários elementos do mesmo tipo, acessados por índices iniciando em zero. Aprendemos a declarar, preencher (manual ou via for), imprimir, buscar elementos, calcular média, somar valores, identificar maior/menor número e contar pares. Foram apresentados exemplos práticos com vetores de inteiros, nomes de produtos e preços.

Matrizes são arrays bidimensionais (linha × coluna), ideais para representar tabelas, notas de alunos e mapas. Estudamos como declarar e percorrer matrizes com laços aninhados, realizar somas, médias por linha, busca de valores, contagem de pares/ímpares e identificar o maior e o menor valor. Também vimos boas práticas como o uso de #define para tornar o código mais legível e flexível.

Esta unidade reforça o uso de laços for, o controle por índices e a organização lógica dos dados em estruturas sequenciais ou tabulares, preparando o aluno para resolver problemas reais com múltiplos dados.

Exercícios

Questão 1. Avalie as afirmativas a seguir sobre o uso de funções na linguagem C.

I – Na linguagem C, os elementos de um array (ou vetor) são armazenados em posições de memória contíguas.

II – Na linguagem C, o primeiro elemento de um array (vetor) tem índice igual a 1, diferentemente de outras linguagens de programação, nas quais os vetores começam no índice igual a zero.

III – A linguagem C não checa automaticamente os limites de acesso de um vetor, o que significa que é possível acessar posições inválidas, o que gera erros em um programa. Por esse motivo, os programadores devem ser muito cuidadosos ao trabalhar com vetores na linguagem C, procurando garantir que o índice de acesso a um vetor sempre esteja dentro dos limites válidos, que dependem do tamanho do vetor.

É correto apenas o que se afirma em:

A) I.

B) II.

C) III.

D) I e II.

E) I e III.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: sabemos que os elementos de um array (também chamado de vetor) são armazenados em posições contíguas de memória, o que permite a sua rápida localização e o seu rápido acesso, uma vez que a posição de memória de qualquer elemento pode ser facilmente calculada pela posição inicial, pelo tamanho de cada elemento e pelo índice do elemento desejado. Adicionalmente, sabemos que a linguagem C não faz nenhuma checagem automática de limites de acesso de um vetor.

II – Afirmativa incorreta.

Justificativa: o índice do primeiro elemento de um vetor na linguagem C é zero.

III – Afirmativa correta.

Justificativa: é o próprio programador que deve garantir o acesso apenas às posições de memórias válidas.

Questão 2. Considere o programa a seguir, escrito na linguagem C.

```
#include<stdio.h>
int main(){
        int vet1[3]={2,1,0};
        char vet2[3]={'a','b','c'};

        printf("Resultado: %c \n", vet2[vet1[1]]);
        return 0;
}
```

Após a execução do programa, qual será a saída na tela?

A) Resultado: abc.

B) Resultado: \n.

C) Resultado: a.

D) Resultado: b.

E) Resultado: c.

Resposta correta: alternativa D.

Análise da questão

Lembremos que a linha do "printf" deve imprimir apenas um dos elementos (no caso, um caractere ou uma letra) do vetor "vet2". Esse vetor tem três elementos, e a posição vai depender do valor da segunda posição do vetor "vet1" (como na linguagem C o primeiro elemento é acessado pelo índice zero, o segundo elemento é acessado pelo índice 1). O segundo elemento do vetor vet1 é o número 1, de forma que esse é o índice que será utilizado para selecionar o elemento do vetor vet2. No caso, isso corresponde à letra "b", o que significa que o texto mostrado na tela será "Resultado: b". Isso corresponde à alternativa D.