

INTRODUÇÃO



LINGUAGEM C
(PARTE 2)

DIA 3

TÓPICOS DE HOJE



Programming

- ESTRUTURAS
 - CONDICIONAIS
 - REPETIÇÃO
- ARRAYS
 - VETOR
 - MATRIZ

ESTRUTURAS

Conceito

- Estruturas condicionais e de repetição em C permitem controlar o fluxo de execução do programa, tomando decisões ou repetindo blocos de código com base em condições ou contadores.
- As estruturas condicionais permitem executar diferentes blocos de código dependendo de certas condições.
- Ex: if, else, else if, switch.
- As estruturas de repetição permitem executar um bloco de código várias vezes.
- Ex: for, while, do while.

ESTRUTURAS CONDICIONAIS

If

- O if é uma estrutura condicional que permite executar um bloco de código apenas se uma determinada condição for verdadeira. Ele é a base para a tomada de decisões dentro de um programa, permitindo que diferentes caminhos de execução sejam seguidos dependendo de certos critérios.
- Sintaxe:

```
1  if (condicao) {  
2      // Código a ser executado se a condição for verdadeira  
3  }  
4
```

ESTRUTURAS CONDICIONAIS

Else

- O else pode complementar uma estrutura com if, caso a condição do if não seja satisfeita.
- Sintaxe:

```
1  if (condicao) {  
2      // Código a ser executado se a condição for verdadeira  
3  } else {  
4      // Código a ser executado se a condição for falsa  
5  }  
6
```

ESTRUTURAS CONDICIONAIS

Else If

- O else if é utilizado quando existem múltiplas condições a serem verificadas, sendo utilizado entre o if e o else.
- Sintaxe:

```
1  if (condicao1) {  
2      // Código a ser executado se condicao1 for verdadeira  
3  } else if (condicao2) {  
4      // Código a ser executado se condicao2 for verdadeira  
5  } else {  
6      // Código a ser executado se nenhuma das condições for verdadeira  
7  }
```

ESTRUTURAS CONDICIONAIS

Switch

- O switch é uma estrutura condicional que permite selecionar e executar um bloco de código entre várias opções, com base no valor de uma expressão. Ele é particularmente útil quando é preciso testar uma única variável ou expressão contra vários valores possíveis.
- Sintaxe:

```
1  switch (expressao) {  
2      case valor1:  
3          // Código a ser executado se expressao == valor1  
4          break;  
5      case valor2:  
6          // Código a ser executado se expressao == valor2  
7          break;  
8      // Outros casos...  
9      default:  
10          // Código a ser executado se expressao não corresponder a nenhum dos casos anteriores  
11          break;  
12 }
```

ESTRUTURAS CONDICIONAIS

Exemplo

- O exemplo a seguir aplica todos os conceitos abordados até agora. São considerados 3 tipos de clientes. Para cada tipo de cliente, haverá uma política de desconto específica, onde ao comprar uma certa quantidade mínima de itens, é possível obter o desconto no valor total da compra.
- O switch é usado para distinguir os tipos de cliente, e if, else e else if para diferenciar as políticas de desconto.
- No final, é calculado o valor total da compra, já com o desconto aplicado(se houver).

ESTRUTURAS CONDICIONAIS

Exemplo

```
1 #include <stdio.h>
2
3 int main() {
4     int tipo_cliente;
5     int quantidade;
6     float preco_unitario = 50.0;
7     float total, desconto;
8
9     printf("Digite o tipo de cliente (1- Regular, 2- Premium, 3- VIP): ");
10    scanf("%d", &tipo_cliente);
11
12    printf("Digite a quantidade de itens comprados: ");
13    scanf("%d", &quantidade);
14
15    // Calculo do total sem desconto
16    total = preco_unitario * quantidade;
17
```

ESTRUTURAS CONDICIONAIS

Exemplo

```
18 // Aplicando desconto baseado no tipo de cliente
19 switch (tipo_cliente) {
20     case 1: // Cliente Regular
21         if (quantidade >= 10) {
22             desconto = 0.05; // 5% de desconto
23         } else {
24             desconto = 0.0; // Sem desconto
25         }
26         break;
27     case 2: // Cliente Premium
28         if (quantidade >= 10) {
29             desconto = 0.10; // 10% de desconto
30         } else if (quantidade >= 5) {
31             desconto = 0.05; // 5% de desconto
32         } else {
33             desconto = 0.02; // 2% de desconto
34         }
35     break;
```

ESTRUTURAS CONDICIONAIS

Exemplo

```
36     case 3: // Cliente VIP
37         if (quantidade >= 10) {
38             desconto = 0.15; // 15% de desconto
39         } else if (quantidade >= 5) {
40             desconto = 0.10; // 10% de desconto
41         } else {
42             desconto = 0.05; // 5% de desconto
43         }
44         break;
45     default:
46         printf("Tipo de cliente inválido!\n");
47         return 1; // Encerrar o programa em caso de tipo de cliente inválido
48     }
49
50     // Calculando o total com desconto
51     total = total - (total * desconto);
52
53     printf("Total após o desconto: R$ %.2f\n", total);
```

ESTRUTURAS DE REPETIÇÃO

For

- O for é uma estrutura de repetição que é utilizada quando se sabe o número exato de vezes que o loop deve ser executado. Ele permite inicializar uma variável de controle, definir uma condição de continuação e incrementar ou decrementar a variável de controle em cada iteração.
- Sintaxe:

```
1  for (inicializacao; condicao; incremento) {  
2      // Código a ser executado em cada iteração  
3  }
```

ESTRUTURAS DE REPETIÇÃO

While

- O while é usado quando não se sabe exatamente quantas vezes o loop deve ser executado, mas se deseja continuar enquanto uma condição é verdadeira.
- Sintaxe:

```
1  while (condicao) {  
2      // Código a ser executado enquanto a condição for verdadeira  
3 }
```

ESTRUTURAS DE REPETIÇÃO

Do While

- O do-while é semelhante ao while, mas a condição é avaliada após a execução do bloco de código. Isso garante que o código dentro do loop seja executado pelo menos uma vez, independentemente da condição.
- Sintaxe:

```
1  do {  
2      // Código a ser executado em cada iteração  
3  } while (condicao);
```

ESTRUTURAS DE REPETIÇÃO

Exemplo

- O exemplo a seguir soma números de 1 a 10 usando as estruturas for, while, e do while.

ESTRUTURAS DE REPETIÇÃO

Exemplo

```
1 #include <stdio.h>
2
3 int main() {
4     int soma = 0;
5
6     // Usando o loop 'for' para calcular a soma de 1 a 10
7     for (int i = 1; i <= 10; i++) {
8         soma += i;
9     }
10    printf("Soma usando 'for': %d\n", soma);
11
12    // Reinicializa a variável soma para 0
13    soma = 0;
14    int i = 1;
15
16    // Usando o loop 'while' para calcular a soma de 1 a 10
17    while (i <= 10) {
18        soma += i;
19        i++;
20    }
```

ESTRUTURAS DE REPETIÇÃO

Exemplo

```
21  printf("Soma usando 'while': %d\n", soma);  
22  
23  // Reinicializa a variável soma e o contador i para 0  
24  soma = 0;  
25  i = 1;  
26  
27  // Usando o loop 'do-while' para calcular a soma de 1 a 10  
28  ~ do {  
29      soma += i;  
30      i++;  
31  } while (i <= 10);  
32  printf("Soma usando 'do-while': %d\n", soma);  
33  
34  return 0;  
35 }
```

ESTRUTURAS DE REPETIÇÃO

Exercícios

- 1- Escreva um programa que gere a sequência de Fibonacci até um valor limite fornecido pelo usuário.
- 2- Escreva um programa que receba um número inteiro e verifique se ele é um palíndromo (um número que é igual quando lido da esquerda para a direita e da direita para a esquerda).
- 3- Escreva um programa que implemente uma calculadora. O usuário pode escolher entre somar, subtrair, multiplicar, dividir, calcular o módulo ou sair do programa. O programa deve pedir dois números inteiros para as operações e utilizar a estrutura switch para escolher a operação correta. O programa continuará solicitando operações até que o usuário escolha a opção de sair.

ARRAYS

Conceito

- Arrays são estruturas de dados fundamentais em C que permitem armazenar múltiplos valores do mesmo tipo em uma única variável. Eles são usados para organizar e manipular grandes conjuntos de dados de maneira eficiente.
- Um array é uma coleção de elementos, todos do mesmo tipo de dado (como int, float, char, etc.), que são armazenados contiguamente na memória. Cada elemento em um array pode ser acessado individualmente usando um índice.
- Ex: Arrays Unidimensionais (Vetores), Arrays Multidimensionais (Matrizes), Arrays Dinâmicos.

ARRAYS UNIDIMENSIONAIS

Vetores

- Um vetor é uma estrutura que armazena múltiplos valores de um mesmo tipo de dado em uma única variável. Cada valor no vetor pode ser acessado individualmente por meio de um índice, que representa sua posição dentro do vetor.
- Para declarar um vetor em C, você precisa definir o tipo de dado dos elementos, o nome do vetor e o tamanho (número de elementos que o vetor pode armazenar).
- Exemplo:

```
1 int numeros[5];
```

ARRAYS UNIDIMENSIONAIS

Vetores

- Inicialização: Pode se inicializar um vetor no momento da declaração ou posteriormente. Se não fornecer uma inicialização, os elementos do vetor serão preenchidos com valores padrão (zero para tipos numéricos).
- Inicialização no momento da definição:

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

- Inicialização posterior:

```
1 int numeros[5];
2 numeros[0] = 1;
3 numeros[1] = 2;
4 numeros[2] = 3;
5 numeros[3] = 4;
6 numeros[4] = 5;
```

ARRAYS UNIDIMENSIONAIS

Vetores

- Inicialização por meio de estruturas de repetição:

```
1 #include <stdio.h>
2
3 int main() {
4     int vetor[10]; // Declara um vetor de 10 inteiros
5
6     // Inicializa o vetor com valores de 1 a 10
7     for (int i = 0; i < 10; i++) {
8         vetor[i] = i + 1; // Atribui i + 1 ao elemento vetor[i]
9     }
10
11    // Imprime o vetor para verificar a inicialização
12    printf("Vetor inicializado:\n");
13    for (int i = 0; i < 10; i++) {
14        printf("%d ", vetor[i]);
15    }
16    printf("\n");
17
18    return 0;
19 }
```

ARRAYS UNIDIMENSIONAIS

Vetores

- Acesso aos elementos:
- Os elementos do vetor são acessados usando índices. Os índices começam em 0 e vão até tamanho - 1.

```
1 #include <stdio.h>
2
3 int main() {
4
5     int numeros[5] = {1, 2, 3, 4, 5};
6     printf("%d\n", numeros[2]); // Imprime 3
7
8     return 0;
9 }
```

ARRAYS UNIDIMENSIONAIS

Vetores

- Acesso aos elementos por estruturas de repetição:

```
1 #include <stdio.h>
2
3 int main() {
4
5     int numeros[5] = {1, 2, 3, 4, 5};
6     for (int i = 0; i < 5; i++) {
7         printf("%d\n", numeros[i]); // Imprime cada elemento do vetor
8     }
9     return 0;
10 }
```

ARRAYS UNIDIMENSIONAIS

Exemplo

- O exemplo a seguir calcula a média de um conjunto de notas de alunos.
- O vetor é utilizado para armazenar as notas dos alunos.

```
1 #include <stdio.h>
2
3 int main() {
4     int notas[5]; // Declara um vetor para armazenar 5 notas
5     int soma = 0; // Variável para armazenar a soma das notas
6     float media; // Variável para armazenar a média das notas
7
8     // Solicita ao usuário para inserir as 5 notas
9     printf("Digite as notas dos 5 alunos:\n");
10    for (int i = 0; i < 5; i++) {
11        printf("Nota do aluno %d: ", i + 1);
12        scanf("%d", &notas[i]); // Lê a nota e armazena no vetor
13        soma += notas[i];      // Adiciona a nota à soma
14    }
15 }
```

ARRAYS UNIDIMENSIONAIS

Exemplo

```
16     // Calcula a média das notas
17     media = (float)soma / 5;
18
19     // Exibe as notas e a média
20     printf("\nNotas inseridas:\n");
21     for (int i = 0; i < 5; i++) {
22         printf("Aluno %d: %d\n", i + 1, notas[i]);
23     }
24
25     printf("\nA média das notas é: %.2f\n", media);
26
27     return 0;
28 }
```

ARRAYS MULTIDIMENSIONAIS

Matrizes

- Uma matriz é um tipo de array multidimensional, onde os elementos são organizados em uma grade de duas ou mais dimensões. A matriz mais comum é a matriz bidimensional, que pode ser visualizada como uma tabela de valores com linhas e colunas.
- Uma matriz é uma coleção de elementos de mesmo tipo, organizados em um formato de linhas e colunas. Cada elemento da matriz pode ser acessado usando dois índices: um para a linha e outro para a coluna.
- Para declarar uma matriz em C, você precisa especificar o tipo de dado, o nome da matriz, o número de linhas e o número de colunas.

```
tipo nome_da_matriz[numero_de_linhas][numero_de_colunas];
```

ARRAYS MULTIDIMENSIONAIS

Matrizes

- Inicialização: Assim como vetores, as matrizes podem ser inicializadas no momento da declaração.

```
1 < int matriz[3][3] = {  
2 |     {1, 2, 3},  
3 |     {4, 5, 6},  
4 |     {7, 8, 9}  
5 };
```

- Posterior: A inicialização posterior pode ser feita por meio de estruturas de repetição.

ARRAYS MULTIDIMENSIONAIS

Matrizes

- Inicialização Posterior:

```
1 int main() {
2     int matriz[3][3]; // Declara uma matriz 3x3
3
4     // Inicializa a matriz com o produto dos índices de linha e coluna
5     for (int i = 0; i < 3; i++) {
6         for (int j = 0; j < 3; j++) {
7             matriz[i][j] = i * j; // Atribui o produto de i e j ao elemento matriz[i][j]
8         }
9     }
10 }
```

ARRAYS MULTIDIMENSIONAIS

Matrizes

- Acesso aos elementos: Cada elemento da matriz é acessado usando dois índices: o primeiro índice para a linha e o segundo índice para a coluna.
- Manipulação de elementos: Para manipular os elementos de uma matriz, geralmente se utiliza laços for aninhados, onde o laço externo percorre as linhas e o laço interno percorre as colunas (conforme visto no exemplo de inicialização posterior).

ARRAYS MULTIDIMENSIONAIS

Exemplo

- O exemplo a seguir demonstra a soma de duas matrizes.

```
1 #include <stdio.h>
2
3 int main() {
4     int A[3][3], B[3][3], C[3][3]; // Declaração de três matrizes 3x3
5
6     // Inicializa a matriz A com valores de 1 a 9
7     printf("Matriz A:\n");
8     int valor = 1;
9     for (int i = 0; i < 3; i++) {
10         for (int j = 0; j < 3; j++) {
11             A[i][j] = valor++;
12             printf("%d ", A[i][j]);
13         }
14         printf("\n");
15     }
16 }
```

ARRAYS MULTIDIMENSIONAIS

Exemplo

```
17 // Inicializa a matriz B com valores de 9 a 1
18 printf("\nMatriz B:\n");
19 valor = 9;
20 for (int i = 0; i < 3; i++) {
21     for (int j = 0; j < 3; j++) {
22         B[i][j] = valor--;
23         printf("%d ", B[i][j]);
24     }
25     printf("\n");
26 }
27
28 // Soma as matrizes A e B e armazena o resultado na matriz C
29 for (int i = 0; i < 3; i++) {
30     for (int j = 0; j < 3; j++) {
31         C[i][j] = A[i][j] + B[i][j];
32     }
33 }
```

ARRAYS MULTIDIMENSIONAIS

Exemplo

```
35     // Imprime a matriz resultante C
36     printf("\nMatriz C (soma de A e B):\n");
37     for (int i = 0; i < 3; i++) {
38         for (int j = 0; j < 3; j++) {
39             printf("%d ", C[i][j]);
40         }
41         printf("\n");
42     }
43
44     return 0;
45 }
```

ARRAYS MULTIDIMENSIONAIS

Exercícios

- 1- Escreva um programa que receba um vetor de 20 números inteiros e determine o maior e o menor elemento desse vetor.
- 2- Escreva um programa que receba duas matrizes esparsas (onde a maioria dos elementos são zeros) de dimensões $n \times m$ e $m \times p$, e calcule o produto entre elas. O programa deve otimizar o cálculo ignorando os elementos zero.

Dicas:

- Utilize uma estrutura que armazene apenas os elementos não nulos (como tuplas de linha, coluna, valor).
- O programa deve calcular o produto sem percorrer as células onde há zeros.

CONGRATULATIONS



MUITO OBRIGADO!



DIN Departamento de
Informática