

# UNIDADE II: ESTRUTURAS CONDICIONAIS

- Estrutura Sequencial
- Estruturas Condicionais

# Estrutura Sequencial

## • Forma geral:

```
<comando 1>;  
<comando 2>;  
<comando 3>;  
...  
<comando n>;
```

Os comandos são separados por ponto e vírgula e executados de forma sequencial, ou seja, na ordem em que eles aparecem

# Estrutura Sequencial

● **Exemplo 01:** Ler os valores dos catetos de um triângulo retângulo e mostrar a hipotenusa

## Algoritmo

```
real a, b, c;  
escrever: "Entrar com 1o cateto:";  
ler b;  
escrever: "Entrar com 2o cateto:";  
ler c;  
a = raiz(pow(b,2) + pow(c,2));  
imprimir "Hipotenusa: " + a;
```

## Fim Algoritmo

# Estrutura Condicional

- Em nosso dia a dia, quase sempre, temos que tomar decisões

**Se** fizer sol, **então**, ...

**Se** idade maior que 18, **então**, ...

**Se** eu ganhar na mega sena, **então**, ...

**Se** o meu time ganhar, **então**, ...

**Se** eu passar em cálculo, **então**, ...

**A programação é totalmente relacionada à tomada de decisões**

# Estrutura Condicional

● **Exemplo 01:** Ler os valores dos catetos de um triângulo retângulo e mostrar a hipotenusa

## Algoritmo

```
real numero;  
escrever "Digite numero real: ";  
ler numero;  
  
se (numero > 45) então  
    escrever: numero;  
fim se
```

Fim algoritmo

**se** (expressão) **então**

|

lista de comandos

**fim se**

## *if*: Comando Se em C-like

```
if (expressão) {
```

```
    lista de comandos
```

```
}
```



## Comando Se – Senão

**se** (expressão) **então**

|

lista de comandos 1

**senão**

|

lista de comandos 2

**fim se**

## *if-else*: Comando Se – Senão em C-like

```
if (expressão) {  
    lista de comandos 1  
} else {  
    lista de comandos 2  
}
```

## Comando Se – Senão – Se

```
se (expressão1) então
    |      lista de comandos 1
senão se (expressão2) então
    |      lista de comandos 2
senão se (expressão3) então
    :      lista de comandos 3
    :
    :      ...
senão
    |      lista de comandos n
fim se
```

# *if-else-if*: Comando Se – Senão – Se em C-like

```
if (expressão1) {  
    lista de comandos 1  
} else if (expressão2) {  
    lista de comandos 2  
} else if (expressão3) {  
    lista de comandos 3  
    ...  
} else {  
    lista de comandos n  
}
```

- Leia 3 números inteiros, selecione o menor e o maior e imprima os seus respectivos valores na tela.
- Leia dois números. Se um deles for maior que 45, realize a soma dos mesmos. Caso contrário, se os dois forem maior que 20, realize a subtração do maior pelo menor, senão, se um deles for menor do que 10 e o outro diferente de 0 realize a divisão do primeiro pelo segundo. Finalmente, se nenhum dos casos solicitados for válido, mostre seu nome na tela.
- Seja uma partida de futebol, leia os números de gols do mandante e do visitante e imprima quem foi o vencedor ou se foi empate.

● O banco do Zé abriu uma linha de crédito para os seus clientes. O valor máximo da prestação não poderá ultrapassar 40% do salário bruto. Fazer um algoritmo que permita entrar com o salário bruto e o valor da prestação e informar se o empréstimo será concedido.

# Aninhamento do Comando Se

```
se ( expressão ) então
    se ( expressão ) então
        ...
    senão
        ...
fim se
senão
    se ( expressão ) então
        ...
    senão
        se ( expressão ) então
            ...
        fim se
    fim se
fim se
```

# Algumas Considerações sobre o Comando Se

- O { e } é obrigatório quando o if ou o else tiver mais de um comando
- Quando eles tiverem exatamente um comando, o { e } é facultativo
- Uma ótima prática de programação é sempre utilizá-los
  - Onde se lê ótima prática de programação entende-se sempre faça isso
- CUIDADO com ifs aninhados



- O else abaixo pertence a qual if?

```
if (n > 0)
    if (a > b)
        z = a;
else
    z = b;
```

- O else abaixo pertence a qual if?

```
if (n > 0)
    if (a > b)
        z = a;
else
    z = b;
```

Sempre associamos  
o else ao if mais interno

• E agora?

```
if (n > 0) {  
    if (a > b)  
        z = a;  
} else  
    z = b;
```

# Operador Ternário: Comando Condicional Enxuto

(expressão) ? valor1 : valor2 ;

# Operador Ternário: Comando Condicional Enxuto

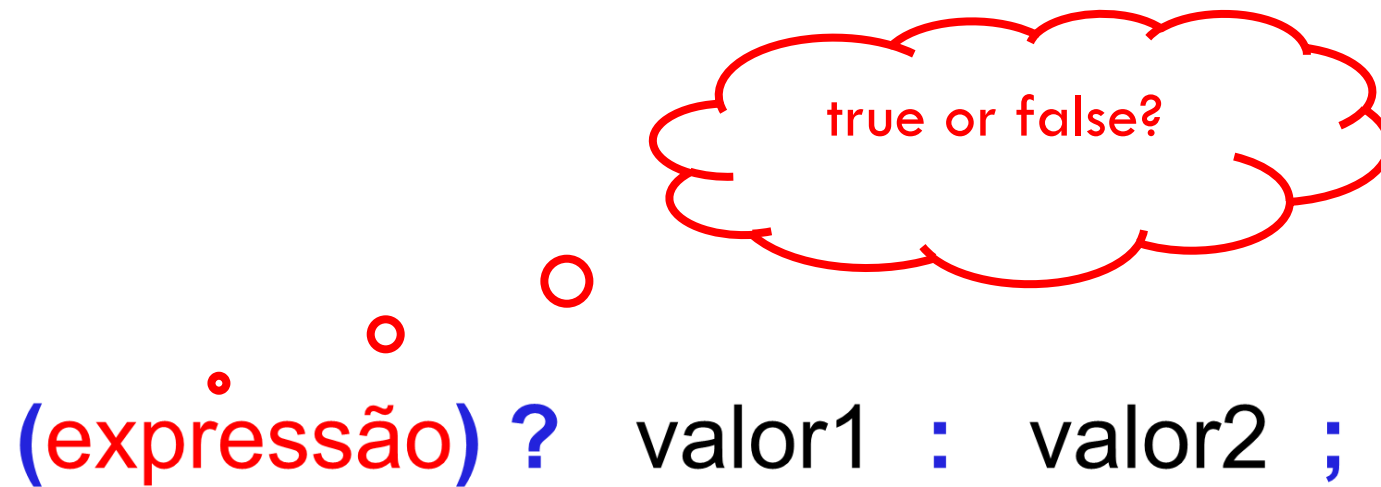


(expressão) ? valor1 : valor2 ;

# Operador Ternário: Comando Condicional Enxuto

(expressão) ? valor1 : valor2 ;

# Operador Ternário: Comando Condicional Enxuto




The diagram illustrates the ternary operator syntax: `(expressão) ? valor1 : valor2 ;`. The word `expressão` is in red, while the other tokens are in blue. Above the opening parenthesis, there are three red circles of increasing size, leading to a red thought bubble. Inside the bubble, the text `true or false?` is written in red, indicating that the expression must evaluate to a boolean value.

`(expressão) ? valor1 : valor2 ;`

true or false?

# Operador Ternário: Comando Condicional Enxuto



The diagram illustrates the flow of the ternary operator. A red arrow points upwards to the opening parenthesis of the expression. Another red arrow points from the closing parenthesis to the question mark. A third red arrow points from the question mark to the word 'valor1'. A fourth red arrow points from 'valor1' to the colon. A fifth red arrow points from the colon to 'valor2'. A sixth red arrow points from 'valor2' to the semicolon. A seventh red arrow points from the semicolon back to the opening parenthesis, completing a loop.

(expressão) ? valor1 : valor2 ;

true

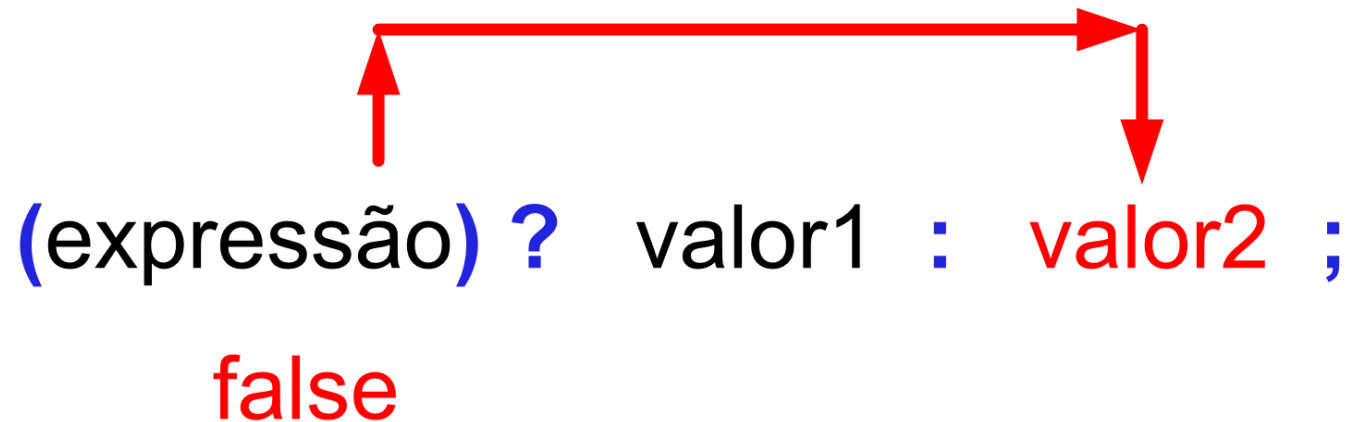


# Operador Ternário: Comando Condicional Enxuto

(expressão) ? valor1 : valor2 ;



# Operador Ternário: Comando Condicional Enxuto



The diagram illustrates the ternary operator syntax: `(expressão) ? valor1 : valor2 ;`. A red arrow points upwards to the opening parenthesis of `(expressão)`. Another red arrow points from the question mark `?` to the start of the horizontal line above `valor1`. A third red arrow points from the end of this horizontal line down to the start of the horizontal line above `valor2`. A fourth red arrow points downwards from the semicolon `;`. The word `false` is written in red below the expression part.

`(expressão) ? valor1 : valor2 ;`  
`false`

# Operador Ternário: Comando Condicional Enxuto

(expressão) ? valor1 : valor2 ;



```
if (a > b) {  
    c = a*a;  
} else {  
    c = b;  
}
```

ou

```
c = (a > b) ? a*a : b;
```

- Faça um programa que leia dois números a e b e mostre o maior deles na tela.

Use o operador ternário

- Faça um programa que leia dois números a e b e mostre o maior deles na tela.

Use o operador ternário

```
int a, b, maior;  
ler a, b;  
maior = (a > b) ? a : b;  
escrever: maior;
```

- Faça um programa que leia três números  $a$ ,  $b$  e  $c$  e mostre o maior deles na tela .

Use o operador ternário

- Faça um programa que leia três números a, b e c e mostre o maior deles na tela .

Use o operador ternário

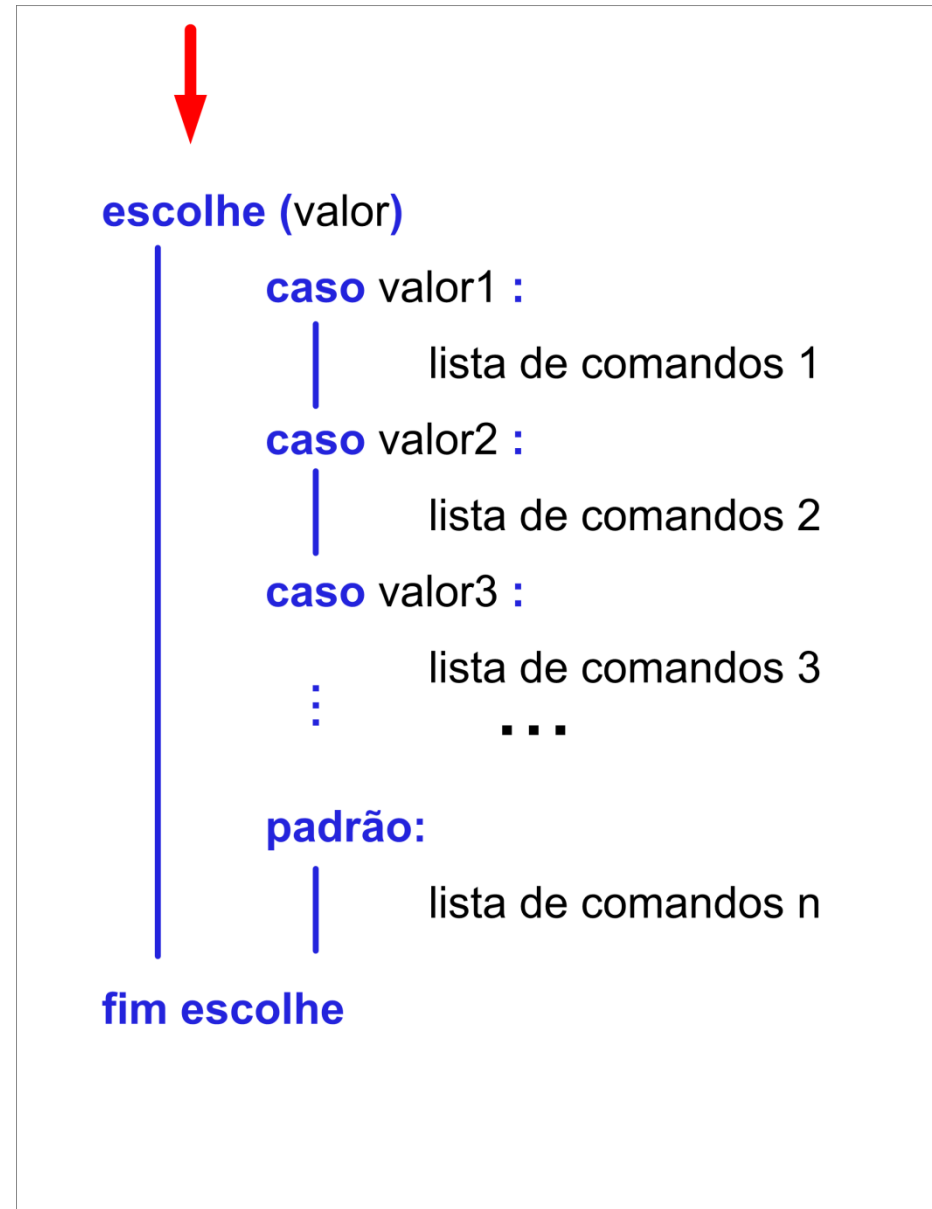
```
int a, b, c, maior;  
ler a, b, c;  
maior = (a > b) ? a : b;  
maior = (maior > c) ? maior : c;  
escrever: maior;
```



# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```


# Comando Escolhe



# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```

## Comando Escolhe

**escolhe** (valor) 

**caso** valor1 :  
        lista de comandos 1

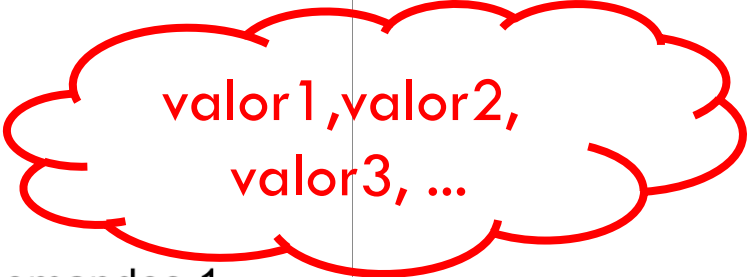
**caso** valor2 :  
        lista de comandos 2

**caso** valor3 :  
        lista de comandos 3

        ⋮  
        ...

**padrão:**  
        lista de comandos n

**fim escolhe**



valor1, valor2,  
valor3, ...

# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```

# Comando Escolhe


```
escolhe (valor)
|
|  caso valor1 :
|      lista de comandos 1
|
|  caso valor2 :
|      lista de comandos 2
|
|  caso valor3 :
|      lista de comandos 3
|      ...
|
|  padrão:
|      lista de comandos n
|
fim escolhe
```

## Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |          lista de comandos 1
|  caso valor2 :
|      |          lista de comandos 2
|  caso valor3 :
|      |          lista de comandos 3
|      |          ...
|  padrão:
|      |          lista de comandos n
|
fim escolhe
```



## Comando Escolhe

**escolhe** (valor) 

**caso** valor1 :  
        lista de comandos 1

**caso** valor2 :  
        lista de comandos 2

**caso** valor3 :  
        lista de comandos 3

        ⋮  
        ...

**padrão:**  
        lista de comandos n

**fim escolhe**

valor1, valor2,  
valor3, ...



# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      lista de comandos 1
|
|  caso valor2 :
|      lista de comandos 2
|
|  caso valor3 :
|      lista de comandos 3
|      ...
|
|  padrão:
|      lista de comandos n
|
fim escolhe
```

# Comando Escolhe


```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```

# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|  caso valor2 :
|      |      lista de comandos 2
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```



## Comando Escolhe

**escolhe** (valor) 

**caso** valor1 :  
        lista de comandos 1

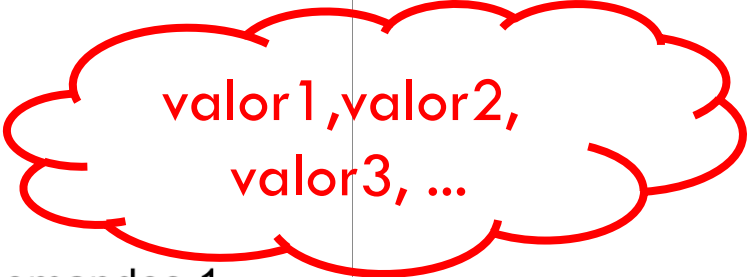
**caso** valor2 :  
        lista de comandos 2

**caso** valor3 :  
        lista de comandos 3

        :  
        ...

**padrão:**  
        lista de comandos n

**fim escolhe**



valor1, valor2,  
valor3, ...

# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```


# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|      |
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```

## Comando Escolhe



## Comando Escolhe

**escolhe** (valor) 

**caso** valor1 :  
        lista de comandos 1

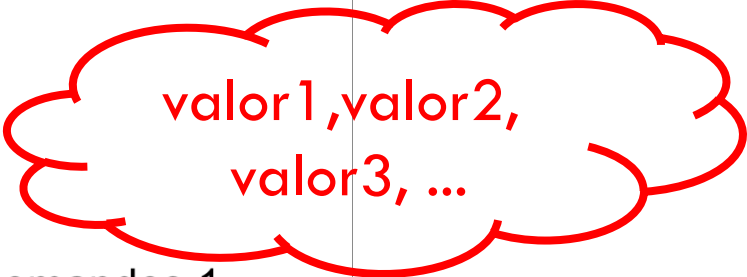
**caso** valor2 :  
        lista de comandos 2

**caso** valor3 :  
        lista de comandos 3

        ⋮  
        ...

**padrão:**  
        lista de comandos n

**fim escolhe**



valor1, valor2,  
valor3, ...



# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      |      lista de comandos 1
|
|  caso valor2 :
|      |      lista de comandos 2
|
|  caso valor3 :
|      |      lista de comandos 3
|      |      ...
|
|  padrão:
|      |      lista de comandos n
|
fim escolhe
```

# Comando Escolhe

```
escolhe (valor)
|
|  caso valor1 :
|      lista de comandos 1
|
|  caso valor2 :
|      lista de comandos 2
|
|  caso valor3 :
|      lista de comandos 3
|      ...
|
|  padrão:
|      lista de comandos n
|
fim escolhe
```



# *switch*: Comando Escolhe em C-like

```
switch (valor) {  
    case valor1 :  
        lista de comandos 1  
        break;  
    case valor2 :  
        lista de comandos 2  
        break;  
    case valor3 : case valor4 :  
        lista de comandos 3  
        break;  
  
    default:  
        lista de comandos n  
}
```

- Faça um programa que leia um caractere, identifique-o e escreva na tela se ele é um ponto, uma vírgula, um ponto e vírgula ou outro sinal. Use o comando switch-case

```
char ch;  
ler ch;  
  
switch( ch ) {  
    case '.':  
        escrever: "Ponto";  
        break;  
    case ',':  
        escrever: "Vírgula";  
        break;  
    case ';':  
        escrever: "Ponto e vírgula";  
        break;  
    default:  
        escrever: "Não é pontuação";  
}
```

- Faça um programa que leia um número inteiro, garanta que o mesmo está entre 1 e 12 e escreva na tela o nome do mês correspondente . Use o comando switch-case

```
int mes;

ler mes;

while ( (mes >= 1 && mes <= 12)
        == false) {
    ler mes;
}

switch (mes) {
    case 1:
        escrever: "janeiro";
        break;

    case 2:
        escrever: "fevereiro";
        break;
```

• • •

```
case 12:
    escrever: "dezembro";
    break;

default:
    escrever: "Mês invalido";
}
```

- Para o programa anterior, comente os **breaks** dos meses 3, 4, 5 e 9, compile seu código e execute seu programa para o mês 2. Descreva o que aconteceu
- Em seguida, execute para o mês 3. Descreva o que aconteceu
- Depois, para o mês 5. Descreva o que aconteceu