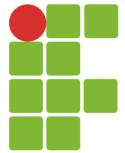
Estruturas de Decisão

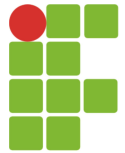
Algoritmos e Programação de Computadores – ABI/LFI/TAI

Prof. Daniel Saad Nogueira Nunes

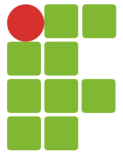
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

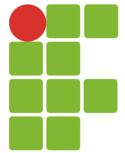


Sumário



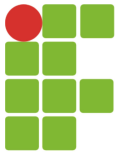
Introdução

- Na última aula verificamos que variáveis e constantes são expressões.
- As operações aritméticas também são consideradas expressões.
- Outro tipo de expressão, são as expressões relacionais, as quais possuem dois tipos de retorno:
  - 0 (Falso).
  - 1 (Verdadeiro).

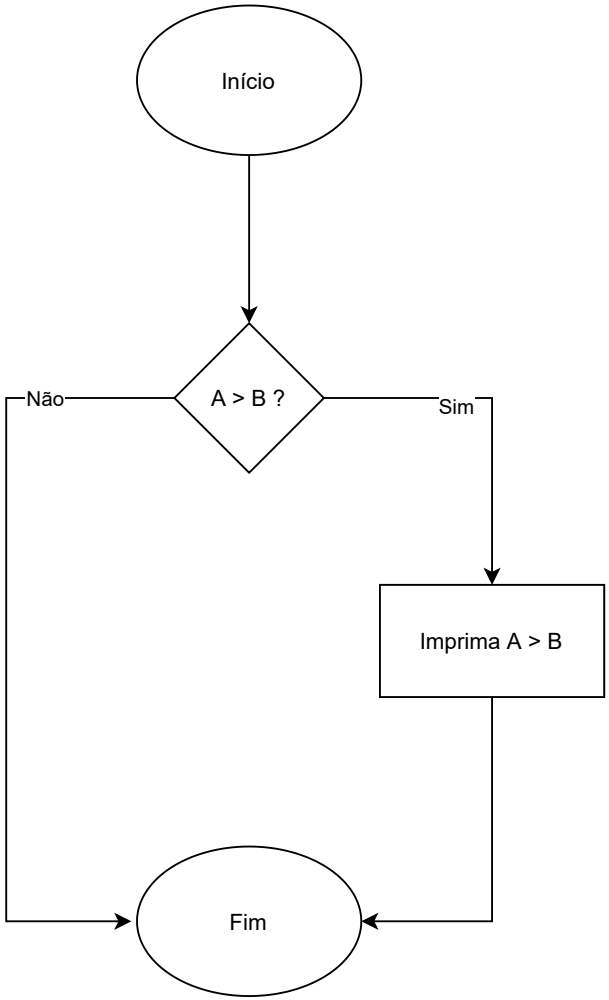


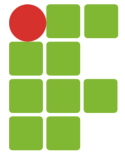
Introdução

- Podemos usar os **operadores relacionais** para realizar comparações entre duas expressões.
- O resultado destas comparações pode ser utilizado, através de **estruturas condicionais** para desviar o fluxo do código, fazendo com que ele se comporte de uma determinada maneira caso uma condição específica seja atingida.



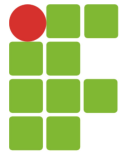
Introdução



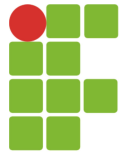


Introdução

- Para escrever expressões mais complexas, podemos utilizar os **operadores lógicos**, que são capazes de ligar duas ou mais expressões de acordo com uma determinada finalidade:
  - **NÃO** `expr`.
  - `expr` **E** `expr`.
  - `expr` **OU** `expr`.
- As expressões criadas com estes operadores também podem ser utilizadas com finalidade de desvio do fluxo do código, fazendo com que ele se comporte de uma determinada forma quando certas condições forem atingidas.

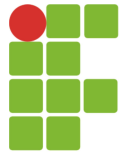


Sumário

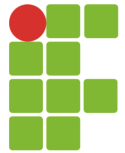


Operadores Relacionais

- Veremos agora os operadores relacionais suportados pela linguagem C.



Sumário



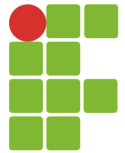
Operadores Relacionais: Igualdade

Igualdade

O operador de igualdade, quando aplicado sobre duas expressões, retorna 1 (verdadeiro), quando elas são iguais, ou 0 (falso), quando elas são diferentes.

Na linguagem C, usamos os símbolos ==

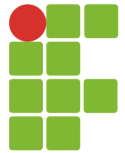
- Exemplo: `9 == 9 // retorna 1 (verdadeiro)` .
- Exemplo: `0 == 5 // retorna 0 (falso)` .



Operadores Relacionais: Igualdade

Exemplo: Igualdade

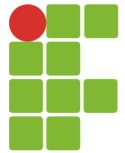
```
1  int main(void) {
2      42 == 42; // Qual o valor da expressão?
3      return 0;
4  }
```



Operadores Relacionais: Igualdade

Exemplo: Igualdade

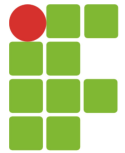
```
1  int main(void) {
2      21 == 42; // Qual o valor da expressão?
3      return 0;
4  }
```



Operadores Relacionais: Igualdade

Exemplo: Igualdade

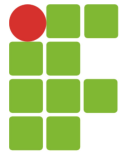
```
1  int main(void) {
2      2 + 3 == 1 + 4; // Qual o valor da expressão?
3      return 0;
4  }
```



Operadores Relacionais: Igualdade

Exemplo: Igualdade

```
1  int main(void) {
2      int a = 2, b = 3, c = 1, d = 4;
3      a + c == b + d; // Qual o valor da expressão?
4      return 0;
5  }
```

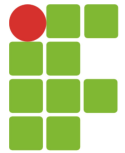


Operadores Relacionais: Igualdade

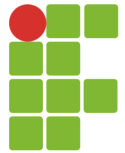
Atribuição vs Igualdade

Não confundir os símbolos de atribuição (=) e igualdade (==). Apesar de parecidos, eles possuem significados totalmente diferentes.

- = é utilizado para atribuir a uma variável o valor de uma expressão.
- == é utilizado para comparar duas expressões.



Sumário



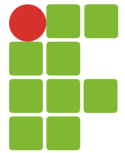
Operadores Relacionais: Diferença

Diferença

O operador de diferença, quando aplicado sobre duas expressões, retorna 1 (verdadeiro), quando elas são diferentes, ou 0 (falso), quando elas são iguais.

Na linguagem `C`, usamos os símbolos `!=`

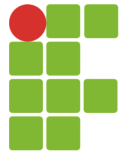
- Exemplo: `9 != 9 // retorna 0 (falso)` .
- Exemplo: `0 != 5 // retorna 1 (verdadeiro)` .



Operadores Relacionais: Diferença

Exemplo: Diferença

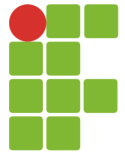
```
1  int main(void) {
2      42 != 42; // Qual o valor da expressão?
3      return 0;
4  }
```



Operadores Relacionais: Diferença

Exemplo: Diferença

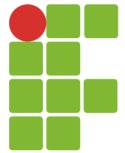
```
1  int main(void) {
2      21 != 42; // Qual o valor da expressão?
3      return 0;
4  }
```



Operadores Relacionais: Diferença

Exemplo: Diferença

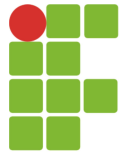
```
1  int main(void) {
2      2 + 3 != 1 + 4; // Qual o valor da expressão?
3      return 0;
4  }
```



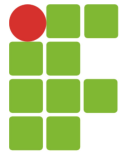
Operadores Relacionais: Diferença

Exemplo: Diferença

```
1  int main(void) {
2      int a = 2, b = 3, c = 1, d = 4;
3      a + b != c + d; // Qual o valor da expressão?
4      return 0;
5  }
```



Sumário



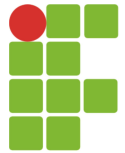
Operadores Relacionais: Maior

Maior

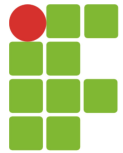
O operador maior, quando aplicado sobre duas expressões, retorna 1 (verdadeiro), quando a da esquerda é maior que a da segunda, ou 0 (falso), caso contrário.

Na linguagem C, usamos o símbolo >

- Exemplo: `9 > 9 // retorna 0 (falso)` .
- Exemplo: `5 > 0 // retorna 1 (verdadeiro)` .



Sumário



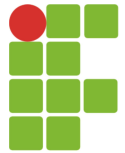
Operadores Relacionais: Maior ou Igual

Maior ou Igual

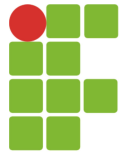
O operador maior ou igual, quando aplicado sobre duas expressões, retorna 1 (verdadeiro), quando a da esquerda é maior ou igual que a da direita, ou 0 (falso), caso contrário.

Na linguagem C, usamos o símbolo >=

- Exemplo: `9 >= 9 // retorna 1 (verdadeiro)` .
- Exemplo: `0 >= 5 // retorna 0 (falso)` .



Sumário



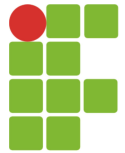
Operadores Relacionais: Menor

Menor

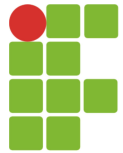
O operador menor, quando aplicado sobre duas expressões, retorna 1 (verdadeiro), quando a da esquerda é menor que a da direita, ou 0 (falso), caso contrário.

Na linguagem C, usamos o símbolo <

- Exemplo: `9 < 9 // retorna 0 (falso)` .
- Exemplo: `0 < 5 // retorna 1 (verdadeiro)` .



Sumário



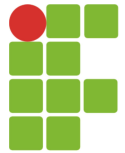
Operadores Relacionais: Menor ou Igual

Menor ou Igual

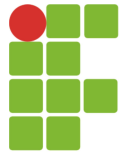
O operador menor ou Igual, quando aplicado sobre duas expressões, retorna 1 (verdadeiro), quando a da esquerda é menor ou igual que a da direita, ou 0 (falso), caso contrário.

Na linguagem C, usamos o símbolo <=

- Exemplo: `9 <= 9 // retorna 1 (verdadeiro)` .
- Exemplo: `5 <= 0 // retorna 0 (falso)` .

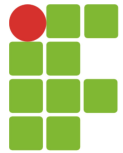


Sumário



Comparação de Números Reais

- Os operadores relacionais apresentados podem ser utilizados para números inteiros, caracteres ou reais.
- Contudo, devido à natureza aproximada da representação computacional dos números reais, a comparação pode não dar o valor esperado, devido à erros de precisão ou arrendondamento.

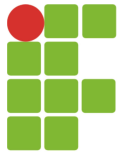


Comparação de Números Reais

Exemplo: Comparação de Reais

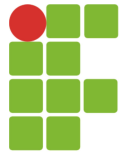
```c
#include <stdio.h>

int main(void) {
    double a = 1.0;
    double b = (0.3 * 3) + 0.1;
    int valor_expr = (a == b); // Qual o valor da expressão?
    printf("%.20f %.20f %d\n", a, b, valor_expr);
    return 0;
}
```

- 0! Falso.
- a vale 1.00000000000000000000.
- b vale 0.99999999999999988898.



Comparação de Números Reais

- O operador == não nos dá o resultado esperado, devido à natureza da representação em ponto flutuante.
- Uma possível solução é usar um valor ϵ bastante pequeno de forma que, se $|a - b| < \epsilon$.
- O valor de ϵ deve ser escolhido de acordo com a sua aplicação.
- Usamos o comando `fabs(expr)` do cabeçalho `<math.h>` para obter o valor absoluto da expressão `expr`.

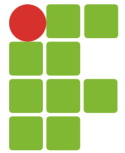


Comparação de Números Reais

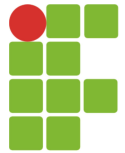
Exemplo: Comparação de Reais

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

int main(void) {
    double a = 1.0;
    double b = (0.3 * 3) + 0.1;
    const double epsilon = 1e-6;                // 0.000001
    int valor_expr = (fabs(a - b) < epsilon); // Qual o valor da expressão?
    printf("%.20f %.20f %d\n", a, b, valor_expr);
    return 0;
}
```

- 1! Verdadeiro.

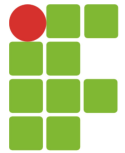


Sumário

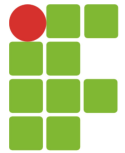


Operadores Lógicos

- Os operadores lógicos conseguem ser aplicados em uma, duas ou mais expressões, para expressar uma ideia mais complexa a ser avaliada.
- Operadores:
  - NÃO (!).
  - E (&&).
  - OU (||).



Sumário



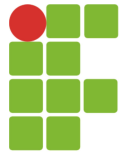
Operadors Lógicos: Negação

Negação

O operador lógico de negação (NÃO), quando aplicado a uma expressão:

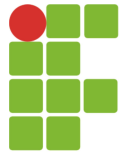
- Retorna verdadeiro quando a expressão é falsa.
- Retorna falso quando a expressão é verdadeira.

Utilizamos o símbolo ! para denotar o operador de negação.



Operadores Lógics: Negação

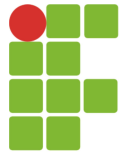
`expr`	`!expr`
0	1
1	0



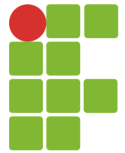
Operadores Lógicos: Negação

Verificação de um Número Ímpar

```
1  #include <stdio.h>
2
3  int main(void) {
4      int n;
5      scanf("%d", &n);
6      int impar = !(n % 2 == 0);
7      printf("%d\n", impar);
8      return 0;
9  }
```



Sumário



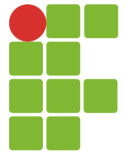
Operadors Lógicos: Conjunção

E

O operador lógico de conjunção (E), quando aplicado a duas expressões:

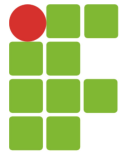
- Retorna verdadeiro quando **ambas** as expressões são verdadeiras.
- Retorna falso quando pelo menos uma das expressões é falsa.

Utilizamos os símbolos && para denotar o operador de conjunção.



Operadores Lógicos: Conjunção

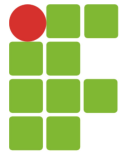
`expr1`	`expr2`	`expr1 && expr2`
0	0	0
0	1	0
1	0	0
1	1	1



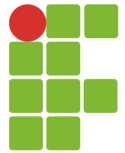
Operadores Lógicos: Conjunção

Verificação de um Número Divisível por 6

```
1  #include <stdio.h>
2
3  int main(void) {
4      int numero;
5      scanf("%d", &numero);
6      int divisivel_por_6 = (numero % 2 == 0) && (numero % 3 == 0);
7      printf("%d\n", divisivel_por_6);
8      return 0;
9  }
```



Sumário



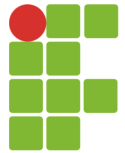
Operadors Lógicos: Disjunção

OU

O operador lógico de conjunção (OU), quando aplicado a duas expressões:

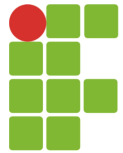
- Retorna verdadeiro quando **ao menos uma** das expressões é verdadeira.
- Retorna falso quando ambas as expressões são falsas.

Utilizamos os símbolos || para denotar o operador de disjunção.



Operadores Lógicos: Disjunção

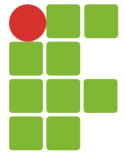
expr1	expr2	expr1 || expr2
0	0	0
0	1	1
1	0	1
1	1	1



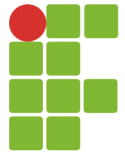
Operadores Lógicos: Disjunção

Verificação de um Número Divisível por 5 ou por 10

```c
#include <stdio.h>

int main(void) {
    int numero;
    scanf("%d", &numero);
    int divisivel_por_5_ou_10 = (numero % 5 == 0) || (numero % 10 == 0);
    printf("%d\n", divisivel_por_5_ou_10);
    return 0;
}
```

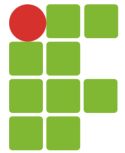


Sumário



Operadores Lógicos: Associatividade

- É possível compor expressões com vários operadores lógicos.
- Neste caso, nenhum operador possui precedência sobre o outro.
- Contudo, a associatividade é da esquerda para a direita.

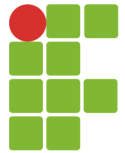


Operadores Lógicos: Associatividade

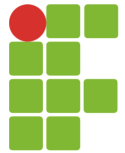
- Para verificar se (um número é ímpar) ou (divisível por 5 e 7 simultaneamente), poderíamos escrever a seguinte expressão:

```
!(n % 2 == 0) || ((n % 5 == 0) && (n % 7 == 0))
```

- Os parênteses aqui são indispensáveis, pois sem eles, a expressão seria lida como: "(o número é ímpar ou é divisível por 5) e é divisível por 7".
- Sem os parênteses, caso o número seja 41, temos que o número é ímpar ou divisível por 5, mas não é divisível por 7, portanto a expressão seria falsa.
- Com os parênteses, o número é ímpar, o que já configura a expressão verdadeira.

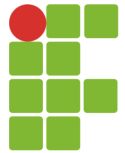


Sumário

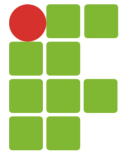


Estruturas de Decisão

- As estruturas de decisão recebem uma expressão e, dependendo do valor verdade dela, é capaz de desviar o código para diferentes trechos.
- Podemos empregar as estruturas de decisão para que o programa se comporte de maneiras diferentes quando determinadas condições forem atingidas.
- Iremos examinar agora as estruturas de decisão da linguagem `C`.



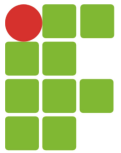
Sumário



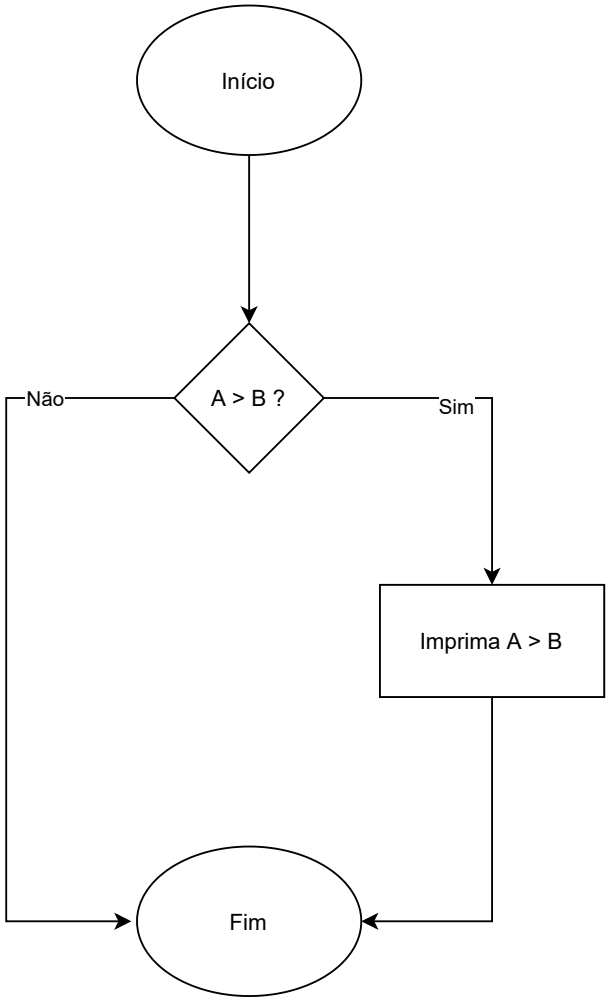
Se Então

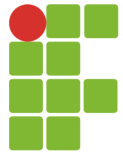
Se então

- A estrutura **Se, então** da linguagem C verifica o valor da expressão e (**Se**) e, caso ela seja verdadeira, executa o bloco de código relacionado à estrutura (**Então**).
- Após finalizar a execução do bloco de código, o programa continua a execução de acordo com as instruções que estão abaixo do bloco de código.



Se Então





Se Então: Sintaxe

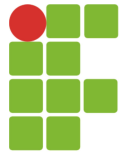
Sintaxe

- Único comando:

```
1  if (expr)
2      comando_1;
```

- Múltiplos comandos:

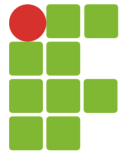
```
1  if (expr) {
2      comando_1;
3      comando_2;
4      ...
5      comando_n;
6  }
```



Se Então: Sintaxe

Observação: Uso das Chaves

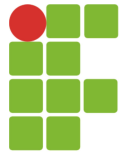
- É um erro de **lógica** não colocar chaves quando o bloco de comandos do **Se, Então** possui mais de um comando.
- O código pode até compilar, mas o resultado não será o esperado.
- Inclusive, as chaves podem ser utilizadas até quando se tem apenas um único comando no bloco.



Se Então

Exemplo: Verifica se um Número é Maior que o Outro

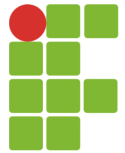
```c
#include <stdio.h>

int main(void) {
    int a, b;
    printf("Digite dois valores, a e b: ");
    scanf("%d %d", &a, &b);
    if (a > b)
        printf("a = %d é maior que b = %d\n", a, b);
    return 0;
}
```



Se Então

Exemplo: Verifica se um Número é Par

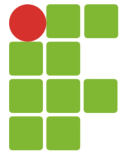
```
1  #include <stdio.h>
2
3  int main(void) {
4      int n;
5      printf("Digite um número: ");
6      scanf("%d", &n);
7      if (n % 2 == 0)
8          printf("%d é par\n", n);
9      return 0;
10 }
```



Se Então

Exemplo: Verifica se um Número é Múltiplo de 6

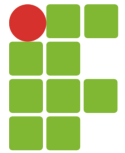
```c
#include <stdio.h>

int main(void) {
    int n;
    printf("Digite um número: ");
    scanf("%d", &n);
    if (n % 2 == 0 && n % 3 == 0)
        printf("%d é múltiplo de 6\n", n);
    return 0;
}
```



Se Então

Exemplo: Verifica se um Número é Múltiplo de 2 ou de 5

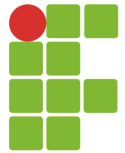
```
1  #include <stdio.h>
2
3  int main(void) {
4      int n;
5      printf("Digite um número: ");
6      scanf("%d", &n);
7      if (n % 2 == 0 || n % 5 == 0)
8          printf("%d é múltiplo de 2 ou de 5\n", n);
9      return 0;
10 }
```



Se Então

Exemplo: Verifica se um Número é Par ou se Ele é Múltiplo de 5 e 7

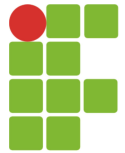
```
1  #include <stdio.h>
2
3  int main(void) {
4      int n;
5      printf("Digite um número: ");
6      scanf("%d", &n);
7      if(n%2 == 0 || ((n%5 == 0) && (n%7==0)))
8          printf("%d é par ou múltiplo de 5 e 7\n",n);
9      return 0;
10 }
```



Se Então

Exemplo: Troca o valor de Dois Números caso o Primeiro seja Maior que o Segundo

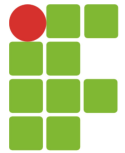
```c
#include <stdio.h>
int main(void) {
    int a, b;
    printf("Digite dois números: ");
    scanf("%d %d", &a, &b);
    if (a > b) {
        int aux = a;
        a = b;
        b = aux;
    }
    printf("%d %d\n", a, b);
    return 0;
}
```



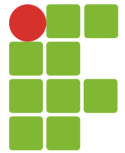
Se Então: Indentação

Boa Prática de Programação: Indentação

- Independente do bloco da estrutura `if` possuir um ou múltiplos comandos, estes devem estar indentados para maior legibilidade de código.

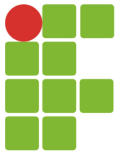


Sumário

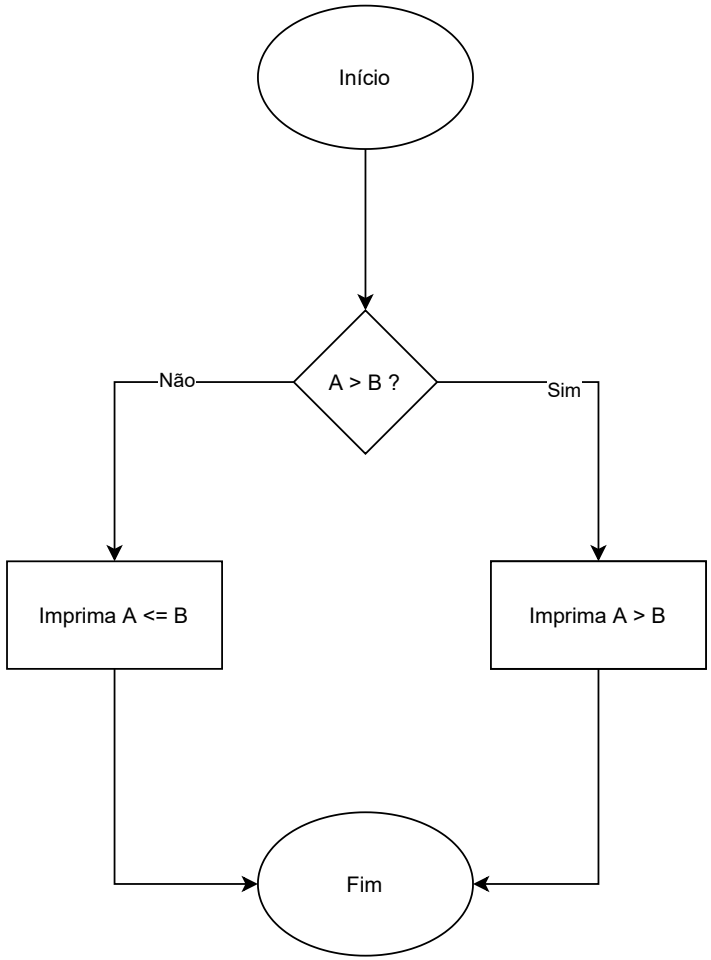


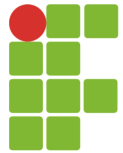
Senão

- Vimos a estrutura **Se, Então**, que executa um bloco de código caso a expressão seja verdadeira.
- E se quisermos que o código faça uma ação quando a expressão seja verdadeira e outra ação caso ela seja falsa?
- Usamos a estrutura **Senão**.
- Após a execução do código de bloco do `if` se a expressão for verdadeira, ou do `else` caso a expressão seja falsa, o código continua sua execução a partir dos comandos que estão abaixo da estrutura `if else`.



Senão



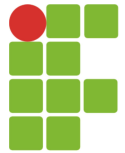


Senão: Sintaxe

Sintaxe

- Único comando:

```
1  if (expr){
2      comando_1;
3      ...
4      comando_n;
5  }
6  else
7      comando_else_1;
```

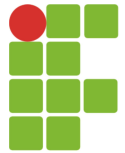


Senão: Sintaxe

Sintaxe

- Múltiplos comandos:

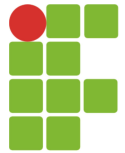
```
1  if (expr){
2      comando_1;
3      ...
4      comando_n;
5  }
6  else{
7      comando_else_1;
8      ...
9      comando_else_n;
10 }
```



Senão: Sintaxe

Observação: Uso das Chaves

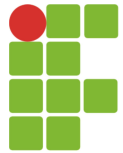
- É um erro de **lógica** não colocar chaves quando o bloco de comandos do **Senão** possui mais de um comando.
- O código pode até compilar, mas o resultado não será o esperado.
- Inclusive, as chaves podem ser utilizadas até quando se tem apenas um único comando no bloco.



Senão

Exemplo: Par ou Ímpar?

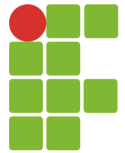
```c
#include <stdio.h>

int main(void) {
    int n;
    printf("Digite um número: ");
    scanf("%d", &n);
    if (n % 2 == 0)
        printf("%d é par\n", n);
    else
        printf("%d é ímpar\n");
    return 0;
}
```



Senão

Exemplo: Maior ou menor?

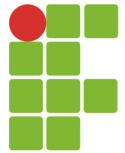
```
1  #include <stdio.h>
2
3  int main(void) {
4      int a, b;
5      printf("Digite dois números: ");
6      scanf("%d %d", &a, &b);
7      if (a >= b)
8          printf("O primeiro número é maior ou igual ao segundo.\n");
9      else
10         printf("O primeiro número é menor que o segundo.\n");
11     printf("Os números digitados foram: %d %d\n", a, b);
12     return 0;
13 }
```



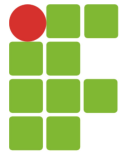
Se Então: Indentação

Boa Prática de Programação: Indentação

- A mesma observação da estrutura `if` serve para a estrutura `else`.
- Os comandos do bloco de código `else` devem estar devidamente indentados.

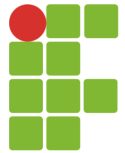


Sumário



Estruturas Aninhadas

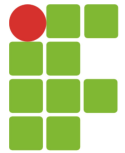
- É possível utilizar uma estrutura `if/else` dentro de outra estrutura `if/else`.
- Com isso, conseguimos criar uma lógica mais complexa nas nossas comparações.



Estruturas Aninhadas

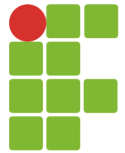
Suponha que queiramos que o programa leia um número inteiro e imprima uma mensagem personalizada para cada um dos seguintes cenários:

1. Número par e menor que 50.
2. Número par e maior ou igual a 50.
3. Número ímpar e menor que 50.
4. Número ímpar e maior ou igual a 50.



Estruturas Aninhadas

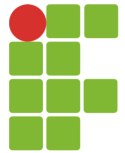
```c
#include <stdio.h>

int main(void) {
    int n;
    printf("Digite um número : ");
    scanf("%d", &n);
    if (n % 2 == 0) { // n é par
        if (n < 50) {
            printf("O número é par e menor que 50\n");
        }
        else {
            printf("O número é par e maior ou igual do que 50\n");
        }
    }
    else { // n é ímpar
        if (n < 50) {
            printf("O número é ímpar e menor que 50\n");
        }
        else {
            printf("O número é ímpar e maior ou igual do que 50\n");
        }
    }
    return 0;
}
```



Estruturas Aninhadas

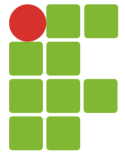
Suponha agora que queiramos ler um número e dizer se:

- Ele é menor ou igual a 100.
- Ele é maior que 100 e menor ou igual a 200.
- Ele é maior que 200 e menor ou igual a 300.
- Ele não se enquadra em nenhuma das opções.



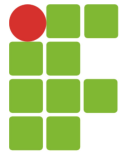
Estruturas aninhadas

- Uma estratégia é utilizar 4 estruturas `if`.
- Contudo o desempenho não será bom, já que sempre 4 comparações serão realizadas.



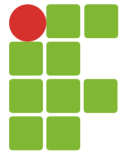
Estruturas Aninhadas

```c
#include <stdio.h>

int main(void) {
    int n;
    printf("Digite um número : ");
    scanf("%d", &n);
    if (n <= 100) {
        printf("O número é menor ou igual a 100\n");
    }
    if (n > 100 && n <= 200) {
        printf("O número é maior que 100 e menor ou igual a 200\n");
    }
    if (n > 200 && n <= 300) {
        printf("O número é maior que 200 e menor ou igual a 300\n");
    }
    if (n > 300) {
        printf("O número não se enquadra em nenhuma das opções\n");
    }
    return 0;
}
```



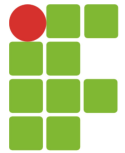
Estruturas aninhadas

- Uma estratégia melhor é estruturas aninhadas.
- Sempre faremos o número de comparações estritamente necessários.



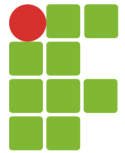
Estruturas Aninhadas

```c
#include <stdio.h>
int main(void) {
    int n;
    printf("Digite um número : ");
    scanf("%d", &n);
    if (n <= 100) {
        printf("O número é menor ou igual a 100\n");
    }
    else { // n > 100
        if (n <= 200) {
            printf("O número é maior que 100 e menor ou igual a 200\n");
        }
        else { // n > 200
            if (n <= 300) {
                printf("O número é maior que 200 e menor ou igual a 300\n");
            }
            else { // n > 300
                printf("O número não se enquadra em nenhuma das opções\n");
            }
        }
    }
    return 0;
}
```



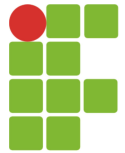
Estrutura Aninhadas

- Neste tipo de encadeamento, em que logo após um `else` existe um único `if` (escada `else-if`), podemos simplificar o nosso código.
- Escrevemos a estrutura `else if`.

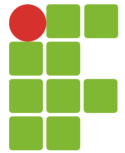


Estruturas Aninhadas

```c
#include <stdio.h>
int main(void) {
    int n;
    printf("Digite um número : ");
    scanf("%d", &n);
    if (n <= 100) { // n <= 100
        printf("O número é menor ou igual a 100\n");
    }
    else if (n <= 200) { // n > 100 e n <= 200
        printf("O número é maior que 100 e menor ou igual a 200\n");
    }
    else if (n <= 300) { // n > 200 e n <= 300
        printf("O número é maior que 200 e menor ou igual a 300\n");
    }
    else { // n > 300
        printf("O número não se enquadra em nenhuma das opções\n");
    }
    return 0;
}
```

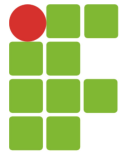


Sumário



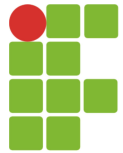
Operador Ternário

- O operador ternário da linguagem `C` possui a seguinte sintaxe:
  `expr_a ? expr_b : expr_c` ;
- Caso `expr_a` seja verdadeira, então o operador ternário irá retornar a expressão `expr_b`.
- Senão ele irá retornar a expressão `expr_c`.



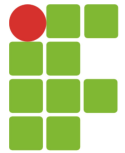
Operador Ternário

- O código a seguir lê dois números inteiros e os compara de modo a atribuir à variável `menor` o menor dos dois números através do operador ternário.



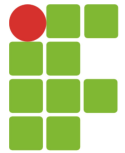
Operador Ternário

```c
#include <stdio.h>

int main(void) {
    int a, b;
    printf("Digite dois números : ");
    scanf("%d %d", &a, &b);
    int menor = a < b ? a : b;
    printf("O menor dos valores digitados é %d\n", menor);
    return 0;
}
```



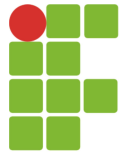
Operador Ternário

- Em seguida, temos o código equivalente utilizando a estrutura `if/else`.
- Repare que o código fica um pouco mais extenso, apesar de a linguagem de máquina gerada em ambos os códigos ser a mesma (muito provavelmente).

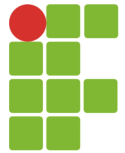


Operador Ternário

```c
#include <stdio.h>

int main(void) {
    int a, b;
    int menor;
    printf("Digite dois números : ");
    scanf("%d %d", &a, &b);
    if (a < b) {
        menor = a;
    }
    else {
        menor = b;
    }
    printf("O menor dos valores digitados é %d\n", menor);
    return 0;
}
```



Sumário



Considerações

- Por motivos didáticos, consideramos nesta aula que quando uma expressão é verdadeira, o seu valor é 1, e quando ela é falsa, o seu valor é 0.
- Contudo, na linguagem `C`, qualquer valor diferente de 0 é considerado verdadeiro, assim, se uma expressão for avaliada em um valor diferente de 0, ela será considerada verdadeira.