Lógica de Programação

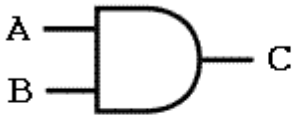
Módulo 03

Estruturas de Seleção

Introdução

Lembra dos operadores lógicos?

AND



Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

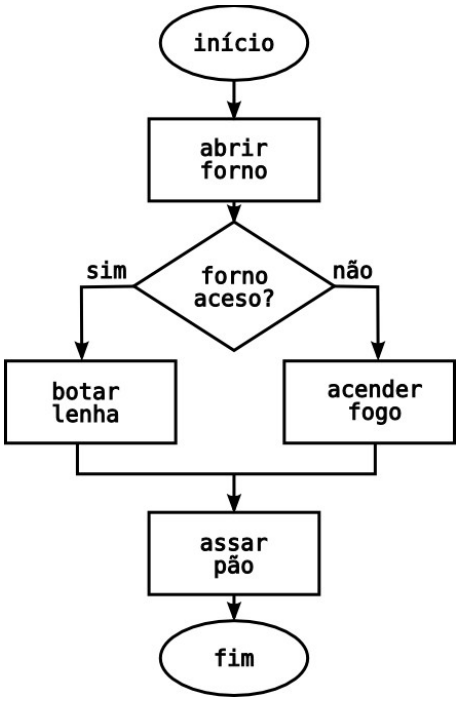
OR

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

NOT

Input	Output
A	C
0	1
1	0

Normalmente quando usamos os operadores lógicos estamos querendo alterar o fluxo do algoritmo. Como assim? Observe o fluxograma a seguir.



Nós estamos querendo assar um pão. Para isso precisamos verificar se o forno está acesso. É nesse momento que haverá uma mudança de fluxo no algoritmo, pois existem duas ações a serem realizadas, mas apenas uma delas será feita dependendo do estado atual do forno. Se ele está acesso (TRUE) então bote mais lenha, senão (FALSE) então acenda o fogo. Só então é que vamos assar o pão.

Até o momento vínhamos concentrando nossos estudos em algoritmos sequenciais, que não sofrem alteração no seu fluxo. Agora veremos como esse fluxo pode ser alterado, o que normalmente ocorre em boa parte dos algoritmos.

Estrutura de Seleção/Decisão SE-ENTÃO

A estrutura de decisão **SE** normalmente vem acompanhada de um comando, ou seja, *se determinada condição for satisfeita pelo comando **SE** então execute determinado comando*. Essas estruturas podem ser simples, compostas ou aninhadas.

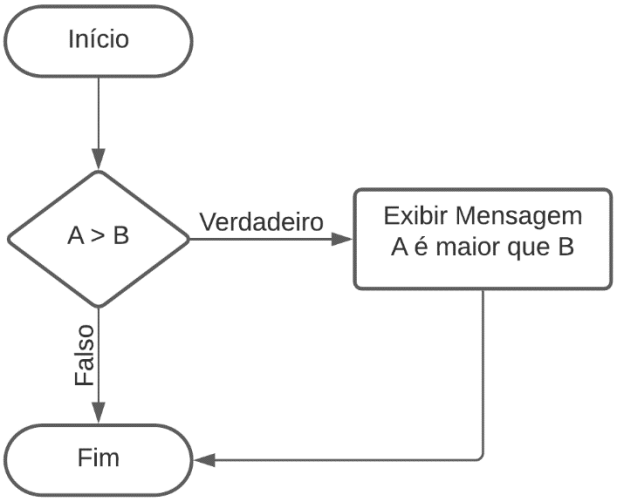
Estrutura de Seleção Simples

Observe a sintaxe para uma estrutura de seleção simples.

```
SE (condição) ENTÃO
      [instruções]
FIM-SE
```

Características Gerais

- A condição é verificada a cada passagem pela estrutura **SE.**
- Se a condição for satisfeita (TRUE), são executadas as instruções que vem depois do **ENTÃO.**
- Se a condição NÃO for satisfeita (FALSE), as instruções que vem depois do **ENTÃO não são** executadas. O algoritmo segue seu fluxo para depois do **FIM-SE**.
- O **SE** sempre executará o bloco de comando ou instrução única se a condição entre parênteses retornar um resultado booleano verdadeiro. Caso contrário, o bloco de comando ou a instrução única não serão executados.



No fluxograma acima podemos observar a estrutura de seleção simples. Se A for maior que B, então uma mensagem será exibida informando que A é maior que B. Pronto. E se não for? Não estamos realizando nenhum procedimento caso a resposta seja negativa.

Observe o mesmo fluxo em pseudocódigo:

```
Algoritmo verifica_numero
Início
  var A, B: inteiro
  A ← 10
  B ← 20
  Se (A > B) Então
    Escreva "A é maior que B.";
  Fim-Se
Fim-Algoritmo
```

Como poderíamos alterar o algoritmo realizar algum procedimento caso A não seja maior que B? Vamos observar como funcionam as Estruturas de Seleção Compostas.

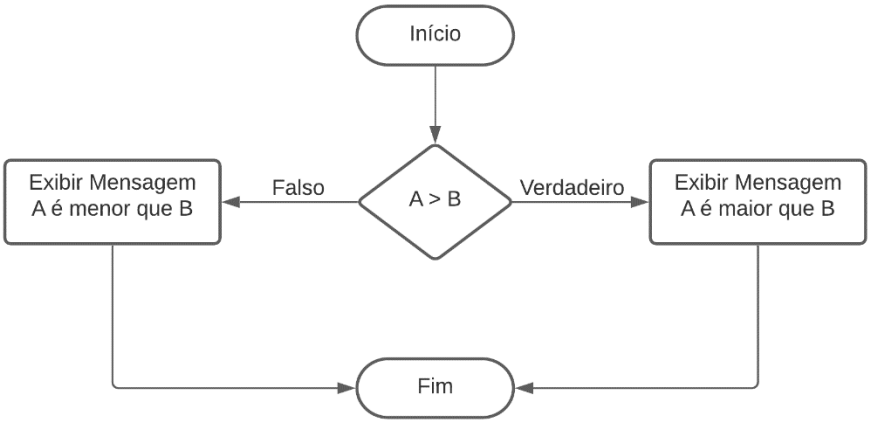
Estrutura de Seleção Composta

Observe a sintaxe para uma estrutura de seleção composta.

```
SE (condição) ENTÃO
      [instruções do SE]
SENÃO
      [instruções do SENÃO]
FIM-SE
```

Características Gerais

- A condição é verificada a cada passagem pela estrutura **SE**.
- Se a condição for satisfeita (TRUE), são executadas as instruções entre **ENTÃO** e **SENÃO**.
- Se a condição NÃO for satisfeita (FALSE), são executadas as instruções entre **SENÃO** e **FIM-SE**;
- As instruções do **SENÃO** serão executadas somente quando o valor da condição do **SE** for FALSE.



Podemos observar agora nosso fluxograma alterado para uma estrutura de seleção composta. Se o resultado do teste for positivo, faremos alguma coisa e se for negativo também realizaremos algum procedimento.

Observe o mesmo fluxo em pseudocódigo:

```
Algoritmo verifica_numero
Início
  var A, B: inteiro
  A ← 30
  B ← 20
  Se (A > B) Então
    Escreva "A é maior que B.";
  Senão
    Escreva "A é menor que B.";
  Fim-Se
Fim-Algoritmo
```

É importante frisar que muitas linguagens permitem a abertura de um bloco após o e **SE** após o **SENÃO**, assim como permitem também que um bloco não seja aberto caso as instruções possuam apenas uma linha. Vamos usar o mesmo algoritmo acima para incluir

mais instruções no **SE** e no **SENÃO** e vamos usar chaves "{ }" para abrir e fechar blocos.

```
Algoritmo verifica_numero
{ // essa chave ficou no lugar do "Início"
  var A, B: inteiro
  A ← 30
  B ← 20
  Se (A > B) Então
  {
    Escreva "A é maior que B.";
    A ← A*10;
  }
  Senão
  {
    Escreva "A é menor que B.";
    A ← B*10;
  } // essa chave fico no lugar do "FimSe"
} // essa chave ficou no lugar do "Fim-Algoritmo"
```

Fizemos uma pequena alteração para mostrar como boa parte das linguagens trabalham quando abrem e fecham blocos, usando chaves. Você deve, no entanto, evitar essa sintaxe quando for criar seus algoritmos. Estamos fazendo isso aqui para mostrar dois pontos.

O primeiro ponto é a questão da indentação. O seu código está indentado quando você insere espaços em branco sempre que se inicia um novo bloco. Veja novamente o mesmo código sem indentação.

```
Algoritmo verifica_numero
{ // essa chave ficou no lugar do "Início"
var A, B: inteiro
A ← 30
B ← 20
Se (A > B) Então
{
Escreva "A é maior que B.";
A ← A*10;
}
Senão
{
Escreva "A é menor que B.";
A ← B*10;
} // essa chave fico no lugar do "FimSe"
} // essa chave ficou no lugar do "Fim-Algoritmo"
```

Ficou mais complicado para ler e compreender o código né? Sem dúvida. É uma

boa prática indentar o seu código. Faça isso sempre para manter a legibilidade do código.

O segundo ponto é a questão de abrir ou não um bloco quando se utiliza uma estrutura de seleção. Em boa parte das linguagens, se você tiver apenas uma instrução depois da estrutura de seleção, não será obrigado a abrir o bloco. Como assim? Observe.

```
Algoritmo verifica_numero
{ // essa chave ficou no lugar do "Início"
  var A, B: inteiro
  A ← 30
  B ← 20
  Se (A > B) Então
  {
    Escreva "A é maior que B.";
    A ← A*10;
  }
  Senão
    Escreva "A é menor que B.";
} // essa chave ficou no lugar do "Fim-Algoritmo"
```

Veja que no **SENAO** agora temos apenas uma instrução, ao passo que no **SE** temos duas. Observe também que no **SE** nós abrimos um bloco com a chave e fechamos o bloco antes do **SENAO**. Já no **SENAO**, como temos apenas uma instrução, nós não abrimos um bloco. A maioria das linguagens vai rodar esse código sem nenhum problema. Mas compare o código acima com o código abaixo e veja qual dos dois é mais legível.

```
Algoritmo verifica_numero
{ // essa chave ficou no lugar do "Início"
  var A, B: inteiro
  A ← 30
  B ← 20
  Se (A > B) Então
  {
    Escreva "A é maior que B.";
    A ← A*10;
  }
  Senão
  {
    Escreva "A é menor que B.";
  }
} // essa chave ficou no lugar do "Fim-Algoritmo"
```

Prefira sempre abrir blocos usando o recurso da linguagem para deixar seu código mais legível. Preze sempre pela legibilidade do seu código!

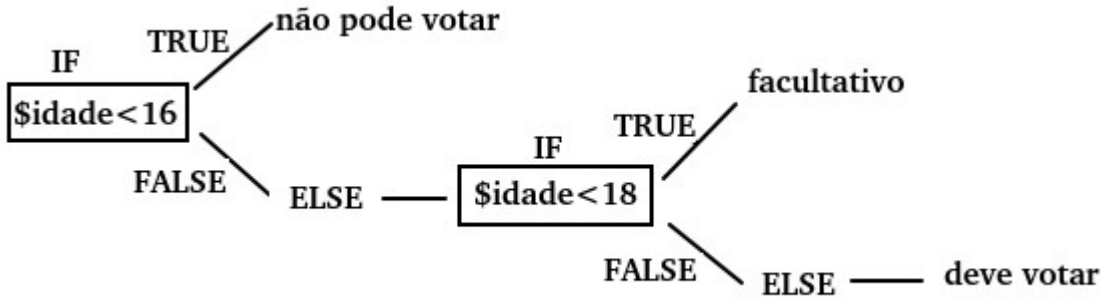
Estrutura de Seleção Aninhada

Observe a sintaxe para uma estrutura de seleção aninhada.

```
SE (condição-geral) ENTÃO
      SE (condição) ENTÃO
            [instruções do SE-GERAL]
      SENÃO
            [instruções do SENÃO]
      FIM-SE
SENÃO
      [instruções do SENÃO]
FIM-SE
```

Características Gerais

- É utilizada, em geral, quando é necessário realizar várias comparações com a mesma variável.
- É chamada de aninhada porque na sua representação fica uma seleção dentro de outra seleção.
- Também é conhecida como **seleção encadeada**.
- Permite fazer a escolha de apenas um entre vários comandos possíveis.



Observe o fluxograma acima sobre a idade de votação. Como ficaria o pseudocódigo? Vejamos.

```
Algoritmo votacao
Início
  var idade: inteiro
  idade ← 17
  Se (idade < 16) Então
    Escreva "não pode votar";
  Senão
    Se (idade < 18) Então
      Escreva "o voto é facultativo";
    Senão
      Escreva "deve votar";
    Fim-Se
  Fim-Se
Fim-Algoritmo
```

Muito bem, até aqui tudo certo. Agora, imagine que tivéssemos um programa que devesse fazer algo dependendo da Unidade da Federação selecionada. Sabemos que temos 27 UFs no Brasil: AL, AM, CE, SC, SP, RS etc. Como ficaria nossa lógica? Utilizaríamos 27 **SE** ou teríamos uma opção mais apropriada?

Estrutura de Seleção/Decisão ESCOLHA-CASO

A estrutura ESCOLHA-CASO é o que utilizamos para o caso mencionado anteriormente. Em inglês normalmente ela é conhecida como **SWITCH-CASE**. Observe sua sintaxe.

```
Escolha <condição>
  Caso <expressão>
    <instruções>;
    Pare;
  Caso
    <instruções>;
    Pare;
  Padrão
    <instruções>;
    Pare;
Fim-Escolha
```

Características Gerais

- A variável a ser testada deve ser sempre do tipo inteiro ou literal.

- É utilizado para oferecer várias opções ao usuário, deixando que escolha um valor dentre vários.
- A principal vantagem dessa estrutura é que ela evita uma série de testes com a estrutura **SE**.
- Funciona de maneira semelhante ao **SE** encadeado.
- A condição após o SWITCH/ESCOLHA informa o valor que será comparado em cada CASE/CASO.
- No primeiro CASE/CASO é verificado se o valor recebido como parâmetro é igual ao seu valor.
- Se o valor do parâmetro informado for o mesmo (igual) do CASE/CASO, será executado o trecho de código dentro do respectivo CASE/CASO.
- Se o valor do parâmetro informado for diferente do CASE/CASO, será testada a condição do próximo CASE/CASO.
- O comando BREAK/PARE é utilizado para forçar a saída do SWITCH/ESCOLHA ao se entrar em um CASE/CASO.
- Sem o BREAK/PARE, todos os CASE/CASO serão testados, mesmo que algum CASE/CASO já tenha atendido a condição.
- O comando DEFAULT/PADRÃO é opcional e define um fluxo alternativo para as situações não atendidas por nenhum CASE/CASO.
- O trecho de código dentro do DEFAULT/SENÃO será executado apenas quando o valor de nenhum CASE/CASO for igual ao valor do parâmetro informado.

Vamos a um exemplo para ficar mais claro. Aproveitemos o que mencionamos anteriormente e vamos realizar alguma tarefa dependendo da UF passada como parâmetro.

```
Algoritmo uf_informada
Início
  var uf: literal
  Escreva "Qual a UF desejada?";
  Leia(uf);
  Escolha (uf);
  Caso "AL"
    Escreva "Alagoas";
    Pare;
  Caso "AM"
    Escreva "Amazonas";
```

```
    Pare;
  Caso "CE"
    Escreva "Ceará";
    Pare;
  Caso "SC"
    Escreva "Santa Catarina";
    Pare;
  Caso "SP"
    Escreva "São Paulo";
    Pare;
  Caso "RS"
    Escreva "Rio Grande do Sul";
    Pare;
  Padrão
    Escreva "UF Inválida";
    Pare;
  Fim-Escolha
Fim-Algoritmo
```

No algoritmo acima não testamos todas as UFs. Então se o usuário passar RJ como parâmetro ele vai receber a mensagem padrão "UF Inválida". Em casos como esse a utilização do ESCOLHA-CASO é mais indicada, pois o código fica mais legível e fácil de compreender. A seguir podemos observar um exemplo de fluxograma para essa estrutura de seleção.