INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PARAÍBA Campus Campina Grande	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA – CAMPUS CAMPINA GRANDE**		
	CURSO:	**CURSO SUPERIOR ENG. DA COMPUTAÇÃO**	
	PERÍODO:	**P1**	**TURMA: N**
	DISCIPLINA:	**ALGORITMOS E LINGUAGEM DE PROGRAMAÇÃO**	
	PROFESSOR:	**CÉSAR ROCHA VASCONCELOS**	SEMESTRE LETIVO **2018.1**

Módulo 03 – ESTRUTURAS DE DECISÃO E REPETIÇÃO

No módulo anterior, foi possível definir (através de regras, tipos de dados, etc.) toda uma pseudolinguagem algorítmica que se aproximava bastante de uma linguagem de programação real. Ela foi utilizada no estudo e na construção de diversos algoritmos. Entretanto, neste módulo, iremos incrementar nossa pseudolinguagem por meio de novos blocos fundamentais e muito comuns na construção de algoritmos: estruturas de **decisão** e de **repetição**.

1. ESTRUTURA SEQÜENCIAL

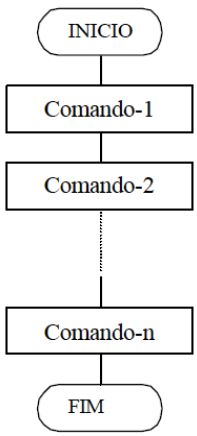
Até aqui, nossos programas foram escritos de forma a serem executados seqüencialmente, não havendo desvios. Todas as instruções eram executadas uma vez.

Veja um exemplo:

```
A, B, C : inteiro
Escreva("Insira o valor de A:")
Leia(A)
Escreva("Insira o valor de B:")
Leia(B)
Escreva("Insira o valor de C:")
Leia(C)

C ← (A + B) * B
Escreva (A, B, C)
```

Fluxograma:



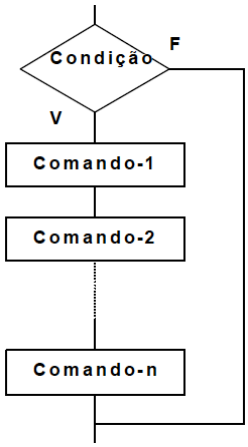
2. ESTRUTURA DE DECISÃO

Ao contrário da execução seqüencial mostrada há pouco, o conjunto de instruções a ser executado no algoritmo dependerá de uma condição. Neste tipo de estrutura, também conhecida como estrutura de seleção ou condicional, é feita a escolha do grupo de ações a ser executado quando determinadas condições (expressões lógicas) são ou não satisfeitas[1].

Sintaxe:

SE (Condição) **ENTAO**
    < lista de comandos >
FIMSE

Fluxograma:



A condição só pode ser uma expressão ou variável que resulte num valor Lógico. Também chamada de **decisão simples**.

Os.: note que a condição dever vir antes do bloco de instruções que será executado. Além disso, para cada SE, deve haver um FIMSE que marca o final do bloco de comandos.

Na elaboração das condições, poderão ser usados os OPERADORES RELACIONAIS, como também os OPERADORES LÓGICOS (E, OU, NÃO, OU Exclusivo) na composição de expressões mais complexas.

Há também as chamadas estruturas de **decisão compostas**. Segundo [1], este tipo de estrutura é utilizada quando se deseja executar um entre dois comandos (ou uma entre duas seqüências de comandos) dependendo, sempre, do resultado de uma condição.

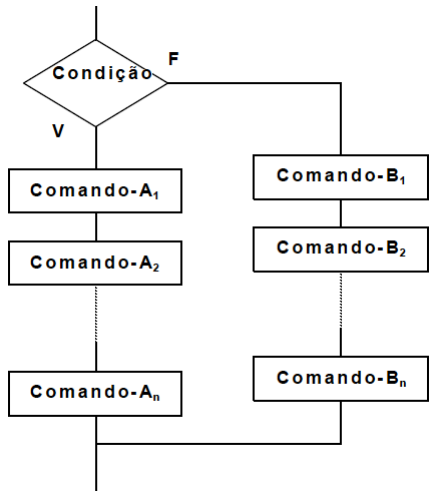
A estrutura de decisão composta é representada pelo comando **SE...ENTAO...SENAO... FIMSE** e possui a seguinte forma:

Sintaxe:

SE (Condição) **ENTAO**
    < lista de comandos >
SENAO
    < lista de comandos >
FIMSE

Da mesma forma que nas estruturas simples, a condição em uma **decisão composta** só pode ser uma expressão ou variável que resulte num valor Lógico.

Fluxograma:



Note que, apesar de haver duas seqüências de instruções, apenas um bloco de instruções será escolhido: se a avaliação da condição resultar no valor lógico verdadeiro (ou V), o primeiro bloco será executado. Caso contrário, (se a avaliação resultar no valor falso) o segundo bloco será selecionado para execução.

2.1 Relembrando...

Tabela-Verdade para os operadores:

E			OU			NÃO	
V	V	**V**	V	V	**V**	V	**F**
V	F	**F**	V	F	**V**	F	**V**
F	V	**F**	F	V	**V**		
F	F	**F**	F	F	**F**		

Exemplos de expressões lógicas compostas:

```
(IDADE > 18) E (SALARIO < 2000)
(X > 12) OU (NÃO t)      (suponha t como sendo do tipo lógico)
(X <> 10) E (A > B) OU (NAO C = D)
```

2.2 Exemplos de estruturas de decisão:

a)
```
SE (VENDAS > 1500.00) ENTAO
   COMISSAO ← 0.10
   PREMIO ← 200.00
SENAO
   COMISSAO ← 0.08
   PREMIO ← 50.00
FIMSE
```

b)
```
SE (IDADE >= 60) OU (POSSUI_FILHOS = VERDADEIRO) ENTAO
   PAGAMENTO ← 330
SENAO
   PAGAMENTO ← 0.60
FIMSE
```

c)
```
SE (ROTA_COLISAO) ENTAO
   Escreva( "Vai bater!")
FIMSE
```

No exemplo acima, ROTA_COLISAO é do tipo Lógico. Portanto, não é necessário fazer ROTA_COLISAO = VERDADEIRO. Para o computador, se você não escrever explicitamente a comparação da variável lógica, ele irá assumir que o bloco só será executado se ROTA_COLISAO tiver valor lógico verdadeiro. Note que o mesmo raciocínio poderia ser aplicado para o exemplo anterior em POSSUI_FILHOS. Com isso, pode-se concluir que se a variável for do tipo Lógico, ela poderá aparecer sem uma comparação explícita na expressão. Mas saiba que compará-la com V não está errado, só é redundante.

2.3 Um exemplo completo (Note que o algoritmo não trata a ocorrência de números iguais)

```
Algoritmo "Maior-Número"
// Programa capaz de ler dois números inteiros e decidir qual deles é o maior.
var
   //Declaração das variáveis
   N1, N2, MAIOR : INTEIRO
inicio
   //Ler os dois números
   Escreva ("Insira o primeiro número:")
   Leia( N1 )
   Escreva ("Insira o segundo número:")
   Leia( N2 )
   //Determinar o maior
   Se N1 > N2 Entao
     MAIOR ← N1
   Senao
     MAIOR ← N2
   Fimse
   //Imprimir o resultado
   Escreva ("O maior número é: ", MAIOR)
fimalgoritmo
```

2.4 SE's aninhados (ou agrupados)

Neste exemplo, é apresentado um algoritmo para calcular o maior dentre três números (X, Y e Z). O processo requer dois passos de comparação (dois SE's, um dentro do outro). Note que o algoritmo não trata a ocorrência de números iguais. Observe:

```
Algoritmo "Maior_de_três"
// lê três números inteiros e determina o maior deles.
var
//Declarar variáveis
N1, N2, N3, MAIOR : inteiro
inicio
//Ler os números
Escreva ("Insira o primeiro número:")
Leia( N1 )
Escreva ("Insira o segundo número:")
Leia( N2 )
Escreva ("Insira o terceiro e último número:")
Leia( N3 )
```

```
//Determinar o maior deles
SE N1 >= N2 ENTAO
   SE N1 >= N3 ENTAO
      MAIOR <- N1  //N1 > N2  e N1 > N3
   SENAO
      MAIOR <- N3  //N3 > N1  e N3 > N2
   FIMSE
SENAO
   SE N2 >= N3 ENTAO
      MAIOR <- N2  //N2 > N1  e N2 > N3
   SENAO
      MAIOR <- N3  //N3 > N2  e N3 > N1
   FIMSE
FIMSE
//Imprimir o resultado final
Escreva ("O maior número é: ", MAIOR)
fimalgoritmo
```

Observe que você poderia ter colocado o comando Escreva em lugar dos comandos de atribuição para a variável MAIOR. Com isso, poderíamos dispensar a variável MAIOR e também eliminar o passo do final do programa logo após a condição de teste. Veja que um mesmo algoritmo pode ter muitas variantes.

Veja uma outra versão, a seguir:

```
Algoritmo "Maior_de_três versão II"
//lê três números e determina o maior deles
var
N1, N2, N3, MAIOR : INTEIRO
inicio
{Ler os números}
//Ler os números
Escreva ("Insira o primeiro número:")
Leia( N1 )
Escreva ("Insira o segundo número:")
Leia( N2 )
Escreva ("Insira o terceiro e último número:")
Leia( N3 )

//Determinar o maior deles
SE (N1 > N2) E (N1 > N3) ENTAO
   MAIOR <- N1  //N1 > N2  e N1 > N3
SENAO
   SE (N2 > N3) ENTAO
      MAIOR <- N2  //N2 > N1  e N2 > N3
   SENAO
      MAIOR <- N3  //N3 > N1  e N3 > N2
   FIMSE
FIMSE
//Imprimir o resultado final
Escreva ("O maior número é:" , MAIOR)
//Término
fimalgoritmo
```

Observe que, se a condição do primeiro SE for falso, podemos, automaticamente, tirar uma conclusão: o maior número não é N1. Por isso, fizemos a comparação entre os outros dois números no SE aninhado.

Questão em sala:

1. Escreva um algoritmo que receba um número inteiro e determine se ele é par ou ímpar.
2. Modifique a primeira versão do algoritmo maior de dois números para que ele trate a situação em que o usuário digitar dois números iguais.

3. ESTRUTURA DE REPETIÇÃO (LAÇO)

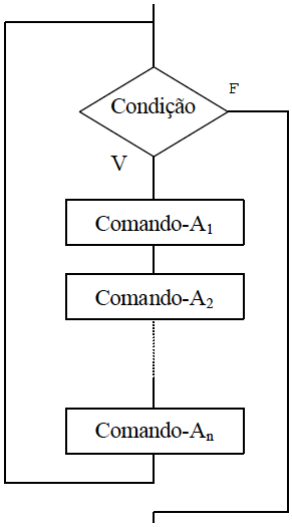
Permite que uma seqüência de comandos seja executada repetidamente até que uma determinada condição de interrupção seja satisfeita. As linguagens de programação reais possuem mais de um tipo de estruturas de repetição, a saber:

3.1 ENQUANTO .. FAÇA

Sintaxe:

ENQUANTO (Condição) **FACA**
    < lista de comandos >
FIMENQUANTO

Fluxograma:



Como o teste da condição é realizado no início do laço, a seqüência de comandos será executada zero ou mais vezes, dependendo da avaliação da condição.

Exemplo:

```
  ...
Escreva("Digite sua senha:")
Leia(senha)
ENQUANTO (senha <> 123) FACA
   Escreval( "Senha incorreta. Tente novamente" )
   Escreva("Digite sua senha:")
   Leia(senha)
FIMENQUANTO
Escreva( "Senha 123 foi digitada!" )
```

Observe que:

- A condição é testada no início do bloco a ser repetido;
- As instruções só serão executadas se a condição for satisfeita;
- A condição é avaliada a cada repetição do laço;
- O laço chega ao fim quando a condição for falsa;
- O comando ENQUANTO tem apenas um único ponto de saída: quando a condição for falsa;
- Deve-se ter cuidado especial com as condições iniciais e finais do laço;
- Depois da interrupção, a seqüência de comandos que vier logo após a expressão FIM-ENQUANTO passa a ser executada.

Mais exemplos:

```
Algoritmo "Números de 1 a 100 (com enquanto...faca)"
var j: inteiro
inicio
    j <- 1
    enquanto j <= 100 faca
          escreval ( j )
          j <- j + 1
    fimenquanto
fimalgoritmo
```

```
Algoritmo "Soma dos números inteiros Pares de 100 a 200: é auto-explicativo"
var
 Par, Soma : inteiro
inicio
    //Versão que utiliza variável Par para encontrar o próximo número par
    Soma <- 0
    Par <- 100
    ENQUANTO (Par <= 200) FACA
          Escreval ( Par )
          Soma <- Soma + Par
          Par <- Par + 2
    FIMENQUANTO
    Escreva ("Resultado da soma:", Soma)
fimalgoritmo
```

Caso especial: Laço Condicional Controlado Por Sentinela:

Quando não conhecemos o número de repetições a ser realizado e este for determinado por um valor que será lido (o que chamamos de **flag**), devemos utilizar um controle de repetições por entrada, também conhecido como **controle por sentinela**. Este tipo de laço utiliza a própria entrada para identificar o fim de um conjunto de dados válidos. Ao contrário dos exemplos acima, este laço não precisa (ou não tem como) saber de antemão a quantidade de conjuntos de dados a ser processada. Os laços controlados por contador (exemplos acima) são muito práticos, mas nem sempre temos como saber a quantidade de repetições que desejamos realizar.
Assim, neste tipo de situação, usaremos uma **flag** que, quando for detectada, causará a interrupção da repetição. Veja um exemplo:

Exemplo 01:

```
Algoritmo "Sentinela”
//este algoritmo lê uma quantidade não conhecida de números inteiros e exibe
//a soma do conjunto.
//A condição de saída do laço será a leitura do valor 0 (flag)."
var
NUM,QUANT,SOMA : inteiro
inicio
    SOMA <- 0
    QUANT <- 0
    Escreva( "Insira um numero (ou digite 0 para sair):")
    Leia( NUM )
    Enquanto ( NUM <> 0 )faca
          SOMA <- SOMA + NUM
          QUANT <- QUANT + 1
          Escreva( "Insira um numero (ou digite 0 para sair):")
          Leia( NUM )
    fimenquanto
   Escreval( "No total, foi (foram) inserido(s) ", QUANT, " número(s) no
                                                          programa..." )
   Escreval( "A soma final é: ", SOMA )
   Escreval( "Encerrando o programa...Ate logo!" )
Fimalgoritmo
```

Observe que existem duas leituras da variável do **flag**: uma antes do laço e a outra dentro do laço (no final dele).

Exemplo 02: (mais avançado):

```
Algoritmo "média de alunos"
// este algoritmo lê um número indeterminado de alunos juntamente com suas
// 3 notas e calcula a média destas notas para cada um.
// A condição de parada do laço será a leitura do valor "FIM" (flag)
// para o nome do aluno.
```

```
var
    NOME : caractere
    N1, N2, N3, MEDIA: real
inicio
    //Ler o primeiro conjunto de dados de um estudante
    Escreva ("Digite o nome do aluno (ou FIM para sair):")
    Leia (NOME)

    //Inicio do laço
    ENQUANTO Nome <> "FIM" FACA
          Escreva ("Digite a primeira nota do aluno:")
          Leia (N1)
          Escreva ("Digite a segunda nota do aluno:")
          Leia (N2)
          Escreva ("Digite a terceira nota do aluno:")
          Leia (N3)
          Media <- (N1 + N2 + N3) / 3
          Escreval ("Aluno: ", NOME )
          Escreval ("1a Nota: ", N1:4:1 )
          Escreval ("2a Nota: ", N2:4:1 )
          Escreval ("3a Nota: ", N3:4:1 )
          Escreval ("Media: ", Media:4:1 )

          //Ler o próximo conjunto de dados do estudante
          Escreva ("Digite o nome do aluno (ou FIM para sair):")
          Leia (NOME)
    FIMENQUANTO
fimalgoritmo
```

- A condição de parada do algoritmo é detectar um aluno cujo nome seja 'FIM';
- Observe que, antes do laço terminar, temos que ler um o nome do aluno para podermos testar novamente a condição do laço (pra ver se o usuário ainda quer continuar no programa)
- Se o primeiro conjunto de dados apresentar o nome do aluno como sendo FIM, então o algoritmo termina sem processar nada;
- Note que você poderia utilizar um outro dado como parada do laço: poderia ser "enquanto a nota1 (ou a 2 ou a 3) não for negativa", já que não existe nota negativa;

3.2 PARA ... ATÉ... PASSO K

Quando o número de repetições for previamente conhecido, podemos utilizar um contador para controlar o número de repetições do laço. Um contador é uma variável do tipo inteiro que deve receber (antes do laço) um valor inicial e dentro do laço (no final dele) este contador já será automaticamente incrementado no valor do passo. A interrupção do laço acontecerá quanto o contador encontrar o valor final indicado (ou superar/passar deste valor).

Sintaxe:

PARA <variável> de <valor-inicial> ATE <valor-limite> [passo <incremento>] FACA
<seqüência-de-comandos>
FIMPARA

Onde:

- variável = variável inteira
- Valor-inicial = Valor inicial da variável inteira
- Valor-final = Valor final da variável inteira
- passo = incremento da variável (Se for omitido, considerar-se-á o incremento de um em um)

Exemplo 01: Soma dos Pares de 100 a 200

```
Algoritmo "Soma dos Pares de 100 a 200: é auto-explicativo. Mas, agora, usa
outra estrutura de repetição. O PARA ...ATÉ... PASSO k"

var
num, soma : inteiro
inicio
//Versão que utiliza o passo para encontrar o próximo número par
soma <- 0
Para num de 100 ate 200 passo 2 faca
   Escreval ( num )
   Soma <- Soma + num
fimpara
Escreva ("Resultado da soma:", Soma)
fimalgoritmo
```

- A principal diferença entre essa versão e a anterior (usando ENQUANTO... FAÇA) é a ausência de uma instrução dentro do bloco de comandos responsável por incrementar a variável que controla o laço (o próximo número par). O comando PARA ATÉ PASSO K já faz isso automaticamente no final do laço (e antes do próximo teste);
- Este laço, para aplicações em que sabemos o número de vezes que deveremos realizar uma repetição, é bem mais claro que o ENQUANTO..FAÇA (versão anterior). Mas lembre-se: nada impede que o ENQUANTO..FAÇA seja usado. Porém, nestes casos, o PARA ATÉ PASSO K é bem mais prático.
- O incremento da variável de controle vai depender das necessidades do programador. Pode ser crescente: para x de 1 até 10 passo 2 (de dois em dois)
- É possível especificar o valor inicial maior que o final, neste caso há decremento da variável contadora. Veja: para x de 10 até 1 passo -1 (decrementando de 1 em 1)

4. COMANDO INTERROMPA

Este comando só deve ser usado dentro de uma estrutura de repetição; em nosso caso, os blocos **Enquanto** e **Para**. A lógica por trás deste tipo de comando é bem simples: **sair "mais cedo" do laço**. Isto é, quando o comando INTERROMPA é encontrado, a execução do laço é interrompida e o fluxo tomará o caminho para fora do laço de repetição (mesmo que a condição de parada do laço ainda não tenha sido atendida).

Veja um exemplo a seguir:

Exemplo 01: Tente imprimir os números de 1 a 10. Porém, quando encontrar o valor 5, saia do laço!

```
Algoritmo "INTERROMPA no Cinco"
// O programa vai tentar imprimir os números de 1 a 10.
// Quando encontrar o valor 5, saia do laço.

var numero : INTEIRO

inicio
    PARA numero de 1 ATE 10 faca // note que passo 1 foi omitido (opcional)
          Escreva (numero)
          SE (numero = 5) ENTAO
                INTERROMPA
          FIMSE
    FIMPARA
fimalgoritmo
```

A saída do programa: 1 2 3 4 5

4. ESTRUTURA DE MÚLTIPLA ESCOLHA

As palavras reservadas "**ESCOLHA**" e "**FIM-ESCOLHA**" marcam esta estrutura. Cada possível opção de escolha deverá ser precedida da palavra **CASO**. Esta estrutura permite fazer uma seleção de opções de maneira mais simples do que se fossem usados vários SE-ENTÃO-FIMSE aninhados.

Sintaxe:

ESCOLHA <expressão-de-seleção>
CASO <exp11>, <exp12>, ..., <exp1n>
<seqüência-de-comandos-1>
CASO <exp21>, <exp22>, ..., <exp2n>
<seqüência-de-comandos-2>
...
OUTROCASO
<seqüência-de-comandos-extra>
FIMESCOLHA

Observe que:

- Para simplificar, a **opção** só poderá ser um número inteiro em nossa pseudolinguagem.
- Apenas um **Caso** será executado. Ao executar o bloco de comandos associado ao Caso selecionado, o algoritmo sairá da estrutura de Escolha.
- O comando **Senão** é opcional. Se for colocado dentro da estrutura de escolha, será executado se a condição não atender a nenhum dos testes do Caso em questão.

Exemplo 01: Menu de escolhas

```
Algoritmo "MENU"
// implementar um menu de escolhas simples a ser apresentado continuamente
// num dispositivo de saída até que o usuário decida sair."
var item : inteiro
    estar_no_laco: logico
inicio
    estar_no_laco <- VERDADEIRO
    ENQUANTO ( estar_no_laco ) FACA
          Escreval("-----------------------------------------")
          Escreval("Digite a operação desejada ou 5 para sair")
          Escreval("-----------------------------------------")
          Escreval("1- Multiplicar dois números")
          Escreval("2- Somar dois números")
          Escreval("3- Dividir dois números")
          Escreval("4- Subtrair dois números")
          Escreval("5- Sair do sistema")
          Escreva("Digite a opção:")
          Leia(item)

          ESCOLHA ( item )
                Caso 1
                      Escreval("Você escolheu a opção Multiplicar!")
                Caso 2
                      Escreval("Você escolheu a opção Somar!")
                Caso 3
                      Escreval("Você escolheu a opção Dividir!")
                Caso 4
                      Escreval("Você escolheu a opção Subtrair!")
                Caso 5
                       estar_no_laco <- FALSO
                OutroCaso
                       Escreval("Esta opção não existe no menu!")
          FIMESCOLHA
    FIMENQUANTO
fimalgoritmo
```

5. CONSIDERAÇÕES FINAIS DA NOTA DE AULA

- As estruturas de decisão podem ser classificadas em simples e compostas.
- Existem vários tipos de laços. Contudo, podemos separá-los, basicamente, em duas categorias: laços condicionais e laços contados.
- A escolha do tipo de laço a usar depende da natureza da aplicação. O uso de um laço inadequado pode tornar o algoritmo difícil de ler.
- O programador deve certificar-se de que o laço vai parar em algum momento. Laços que nunca acabam são conhecidos como "loops" infinitos. A chance de se cometer um loop infinito é maior nos laços condicionais, já que o laço contado é mais legível.
- É possível sair do laço mais cedo do que o desejado com o comando INTERROMPA.
- Se quisermos selecionar uma opção dentre várias em um algoritmo, podemos utilizar o bloco Escolha/Caso/FimEscolha

6. CRÉDITOS E BIBLIOGRAFIA UTILIZADA PARA A GERAÇÃO DESTA NOTA DE AULA:

[1] EGYPTO, Cândido. **Lógica e Algoritmos**. CEFET-PB, 2003. (Livro-texto)
[2] PEREIRA, Frederico C. G. **Algoritmos**. CEFET-PB, 2004.