Estruturas de repetição

1. Fundamentação Teórica:

Considere o seguinte problema: Elaborar um algoritmo que tenha como entrada a nota de **um** aluno e como saída o resultado: "***Aprovado***" se o aluno obtiver nota superior ou igual a 60 e "***Reprovado***" se o aluno obtiver nota inferior a 60. Uma solução para o problema poderia ser a seguinte:

```
programa
{
    funcao inicio()
    {
        real nota
        escreva ("Informe a nota do aluno: ")
        leia (nota)
        se (nota >= 60)
        {
            escreva ("Aluno aprovado.")
        }
        senao
        {
            escreva ("Aluno reprovado")
        }
    }
}
```

Porém, um professor deseja checar o resultado de 30 alunos. Então, o programa acima deverá ser ***chamado ou executado* 30 vezes** para mostrar o resultado de todos os 30 alunos de uma turma, pois, a execução do mesmo permite informar a nota de somente **um** aluno. Podemos alterá-lo para que com somente uma ***chamada ou execução*** o mesmo possa mostrar o resultado de todos os 30 alunos.

Observa-se que na prática, existem vários problemas, que normalmente precisam de programas que necessitam processar repetidas vezes um bloco de instruções. Portanto, uma sequência de instruções se repetirem, copiá-las estas repetidas vezes não é recomendado, por não obedecer às boas práticas de programação. Portanto, é sempre interessante ter um código-fonte limpo e simples.

Para isso é necessário utilizarmos uma ESTRUTURA DE REPETIÇÃO, ou seja, uma estrutura de controle do fluxo lógico que permite executar diversas vezes um mesmo trecho do algoritmo, porém, sempre verificando **antes** de cada execução se é "permitido" repetir o mesmo trecho, ou seja, se uma condição é satisfeita (verdadeira).

1.1. Estrutura de repetição padrão (com teste no início)

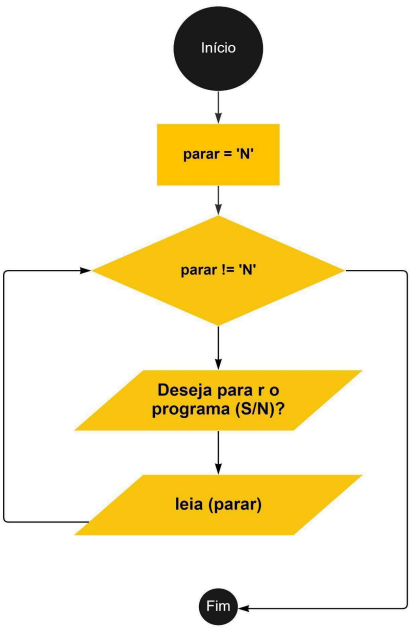
Para realizar repetições com teste no início, podemos utilizar o comando **enquanto**, permitindo que um bloco de comandos ou uma ação primitiva seja repetida **enquanto** uma determinada (condição) for verdadeira.

Observe abaixo a sintaxe do comando **enquanto**.

```
logico condicao = verdadeiro
enquanto (condicao)
{
   //Executa a as instruções dentro do laço enquanto a condicao for verdadeira
}
```

O fluxograma ao lado ilustra um algoritmo que verifica uma variável do tipo caracter. Enquanto a variável for diferente da letra **'S'** o comando **enquanto** será executado, assim como as instruções dentro dele. No momento em que o usuário atribuir **'S'** à variável, a condição do comando **enquanto** será **falsa** e o programa será encerrado.

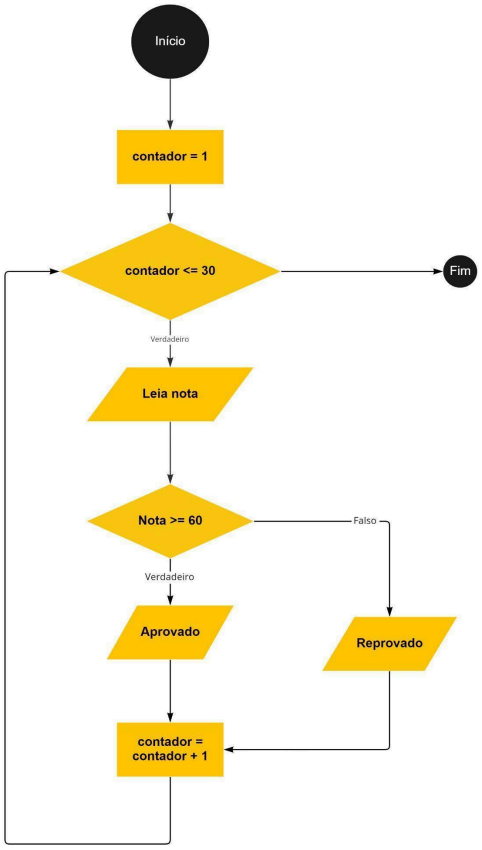
Apresenta-se também o algoritmo em portugol do mesmo fluxograma.




```
programa
{
  funcao inicio()
  {
     caracter parar
     parar = 'N'

     enquanto (parar != 'S')
     {
        escreva ("deseja parar o laço? (S/N)")
        leia (parar)
     }
  }
}
```

Agora retornaremos ao primeiro problema, que é ler a nota de um aluno e informar se o mesmo está **aprovado** ou **reprovado.** Considerando o mesmo problema para 30 alunos, teremos que executar o primeiro algoritmo 30 vezes ou elaborar um algoritmo com laço de repetição, conforme exemplificado no fluxograma e no algoritmo em portugol a seguir.




```
programa
{
    funcao inicio()
    {
        inteiro contalunos = 1
        real nota
        enquanto (contalunos <= 30){
            escreva("\nDigite a nota do " + contalunos + "º aluno: ")
            leia(nota)
            se(nota >= 60){
                escreva("Aluno aprovado")
            }senao{
                escreva("Aluno reprovado")
            }//fim do se
            contalunos++ //incremento

        }//fim do enquanto
    }
}
```

RASTREIO

OBS: Diferente dos outros rastreios que estávamos acostumados a fazer, agora representaremos as variáveis em uma tabela para mostrar o que foi acontecendo com as mesmas durante o processo de repetição.

contador	nota
...	...

Na maioria dos loops são necessários:

1) inicializar a(s) variável(eis) de controle antes do loop;
2) incrementar/decrementar ou alterar o valor da(s) variável(eis) de controle dentro do loop;
3) definir bem a condição do loop para evitar repetições infinitas.

Observe que no exemplo anterior foi necessário **incrementar** a variável de controle (variável **contador**). Isso para evitar o loop infinito, que é resultante de erro na lógica.

comando enquanto é útil quando não sabemos o número exato de repetições antecipadamente, mas sabemos a condição que precisa ser atendida.

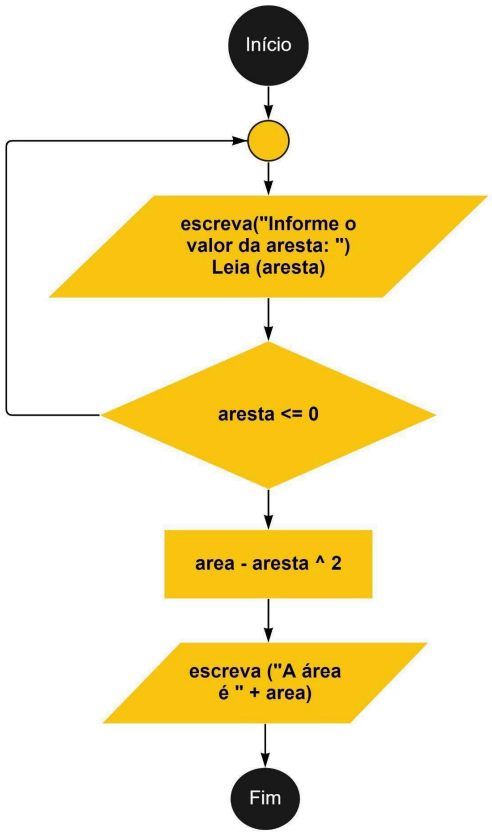
1.2. Estrutura de repetição com teste no final - *faca{ ... }enquanto (condção)*

Em algumas situações, faz-se necessário verificar se uma condição é verdadeira ou não após a entrada de dados do usuário. Para situações como essa, podemos usar o laço de repetição **faca-enquanto**. Este teste é bem parecido com o **enquanto**. A diferença está no fato de que o teste lógico é realizado no final, e com isso as instruções do laço sempre serão realizadas pelo menos uma vez. O teste verifica se elas devem ser repetidas ou não.

A sintaxe é respectivamente a palavra reservada **faca**, entre chaves as instruções a serem executadas, a palavra reservada **enquanto** e entre parênteses a condição a ser testada.

```
logico condicao = verdadeiro
faca
{
   //Executa os comandos pelo menos uma vez, e continua executando enquanto a condição for verdadeira
} enquanto (condicao)
```

A figura abaixo ilustra um algoritmo que calcula a área de um quadrado. Note que para o cálculo da área é necessário que o valor digitado pelo usuário para aresta seja maior que 0. Caso o usuário informe um valor menor ou igual a 0 para a aresta, o programa repete o comando pedindo para que o usuário entre novamente com um valor para a aresta. Caso seja um valor válido, o programa continua sua execução normalmente e ao fim exibe a área do quadrado.




```
programa
{
  funcao inicio()
  {
     real aresta, area

     faca
     {
        escreva ("Informe o valor da aresta: ")
        leia (aresta)
     } enquanto (aresta <= 0)

     area=aresta*aresta
     escreva("A área é: ", area)
  }
}
```

A seguir outro exemplo com laço de repetição com teste no final. O algoritmo recebe um conjunto de valores e ao final, calcula e exibe a média.

```
programa
{
    funcao inicio()
    {
        real valor, media, soma=0.0
        caracter continuar = 's'
        inteiro cont = 0

        faca{
            escreva("Informe um valor: ")
            leia (valor)
            soma = soma + valor
            cont++
            escreva("Deseja continuar (s/n):")
            leia (continuar)
        }enquanto(continuar != 'n')

        media = soma / cont
        escreva("A média é " + media)
    }
}
```

Observe que a variável **continuar** é inicializada com o valor **'s'** e o algoritmo entra pelo menos uma vez no laço de repetição. Em seguida recebe o valor, efetua o cálculo da soma, incrementa a variável **cont** e logo pergunta ao usuário **se ele deseja continuar**. Caso o usuário informe **'s',** o programa continua e pede o próximo valor. Caso contrário, o laço de repetição é encerrado, é realizado então o cálculo da média e a exibição do resultado.

Enquanto a condição for verdadeira, executará as instruções que estão dentro do laço.

1.3. Laço indeterminado

Há situações em que o número de repetições do laço é desconhecido e é necessário criar uma condição de parada, em tempo de execução do algoritmo, conforme já foi apresentado no exemplo anterior.

Imagine o seguinte problema, é necessário criar um algoritmo que leia a nota de um número indefinido de alunos. Para cada aluno exibir se o mesmo é **aprovado, em recuperação** ou **reprovado**. Como não se sabe o número de alunos da turma, para cada nota lida, é necessário perguntar ao usuário se ele deseja ou não continuar. Caso a resposta seja sim, pediremos a próxima nota, caso contrário o programa será encerrado.

```
programa{
    funcao inicio()
    {
        caracter continuar = 's'
        real nota

        enquanto (continuar == 's'){
            escreva("\nDigite a nota de um aluno: ")
            leia(nota)

            se(nota>=60){
                escreva("Aluno aprovado com a nota " + nota + " pontos.")
            }senao se(nota>=40){//nota<60
                escreva("Aluno em recuperação com a nota " + nota + " pontos.")
            }senao{
                escreva("Aluno reprovado com a nota " + nota + " pontos.")
            }//fim do se

            escreva("\nDeseja continuar(s/n): ")
            leia(continuar)
        }//fim do enquanto
    }
}
```

VAMOS PRATICAR?

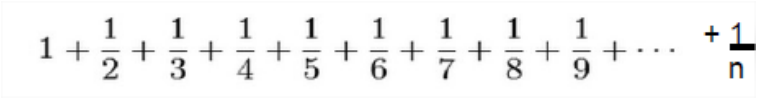
2. Exercícios:

1) Elabore um algoritmo que leia as notas das provas dos alunos de uma classe, calcule e exiba a média das notas da turma. Valide as notas, aceitando apenas notas entre 0 e 100, inclusive. E não se sabe a quantidade de alunos.

2) Elabore um programa que conte quantos números são divisíveis por 3, entre 1 e N, sendo N um número qualquer fornecido pelo usuário.

3) Escreva um algoritmo que leia um número desconhecido de valores, um de cada vez, e conta quantos deles estão em cada um dos intervalos: [0, 25],]25,50],]50,75],]75,100].

4) Observe o somatório abaixo.



Leia do usuário o valor de n, calcule e exiba o resultado da série acima. Não aceitar valor negativo ou nulo para N.

5) Faça um algoritmo que leia um número inteiro (num) e imprima os num primeiros números pares.
Exemplo:
Valor lido: 6
Saída do algoritmo: 2 4 6 8 10 12

6) Faça um algoritmo que calcule o produto de 2 números inteiros lidos (num1 e num2) por meio do método de somas sucessivas. Suponha que num1 e num2 são positivos.
Exemplo:
Valores lidos: 3 4
Saída do algoritmo: 12
Dica: 3 x 4 = 3 + 3 + 3 + 3 = 12

7) Crie um algoritmo que valide a senha informada pelo usuário, comparando-a com uma senha pré-definida. Peça ao usuário para digitar uma senha e exiba na tela:
   - Senha correta, caso o usuário digite a senha certa.
   - Pedir para informar a senha novamente, caso o usuário erre a senha.

8) Crie um algoritmo que peça ao usuário para digitar uma sequência de números e, em seguida, imprima o maior e o menor número digitado. Para cada número lido, perguntar ao usuário se ele deseja informar outro número. Quando ele informar uma resposta negativa, dê o resultado e encerre o algoritmo.

9) Elabore um algoritmo que leia nome e salário de N funcionários.
Ao final da leitura, deve ser informado na tela o nome e o salário do funcionário com maior salário da empresa e depois, o nome e o salário do funcionário com menor salário.

10) Elabore um algoritmo para cadastrar os produtos de um supermercado com os seguintes dados:
   - código
   - descrição
   - estoque mínimo
   - estoque atual
   - preço de custo
   - preço de venda

   Calcular e exibir:
   - A quantidade de produtos com **estoque mínimo menor que 20**.
   - O **preço médio de custo** dos produtos em que o **estoque atual é maior que 50**.
   - Quantidade de produtos que o **estoque atual está abaixo do estoque mínimo**.
   - A porcentagem de produtos em relação ao **total**, cujo lucro seja abaixo de R$500,00.

   Dica: **Lucro** é obtido **subtraindo** do **preço de venda pelo preço de custo**.
   Para cada produto lido, o usuário deverá informar se ele deseja continuar a informar dados para mais produtos.