Desenvolvimento de Sistemas

Repetições: estruturas de repetição condicional pré-teste, pós-teste e com variável de controle

Introdução

Você, como um ótimo aluno, certamente pensa da seguinte forma: “Enquanto eu não entender este conteúdo, não irei parar de estudar!”. “Tenho 5 horas para estudar. Para cada uma hora de estudo, vou relaxar por 10 minutinhos!”. “Vou enviar minhas atividades para ver se estão corretas. Enquanto eu não for aprovado, enviarei novamente!”. O que esses pensamentos têm em comum?

Além de muita determinação, eles também podem ser vistos como estruturas de repetição! Mas o que são estruturas de repetição, afinal? Você saberá reconhecer quais tipos de estrutura de repetição foram mencionados ao final deste conteúdo.

Códigos repetidos que apresentam uma mesma função podem não afetar códigos pequenos utilizados para estudo, entretanto, imagine uma grande empresa em que a quantidade de informações com as quais você trabalhará é muito maior! Trabalhar individualmente com cada informação pode tornar muito lento seu trabalho e o próprio processamento do sistema, portanto, criar laços de repetição é um meio de realizar as tarefas repetitivas de um código utilizando um número menor de linhas para otimizar sua produção. As estruturas de repetição são basicamente blocos de comandos que são repetidos até atingirem uma condição definida, como *loopings*.

Juntamente com outras estruturas condicionais e conceitos de lógica anteriormente estudados, as estruturas de repetição são muito importantes para você utilizá-las ao longo de sua carreira. Um desempenho lógico bem desenvolvido facilita a introdução e a compreensão em qualquer linguagem de programação, portanto, é imprescindível dominar cada uma das bases que serão apresentadas.

As estruturas de repetição podem ser definidas como pré-teste, pós-teste e com variável. Veja a seguir, de forma mais detalhada, como cada uma delas funciona e como aplicá-las.

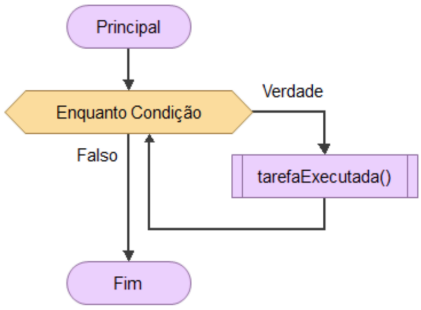
Os trechos de código em Portugol a seguir apresentarão apenas o conteúdo dentro da “funcao inicio()”. Para testar o funcionamento dos códigos, basta copiar o trecho desejado e colar dentro do seu código na “funcao inicio”.

Estrutura de repetição pré-teste: enquanto (while)

A estrutura pré-teste, tal como seu nome aponta, é uma estrutura que verificará se a condição especificada no código é atendida antes mesmo de executar os comandos do laço de repetição, dessa forma, **enquanto** a **condição** estiver sendo atendida (retornando **verdadeiro**), os comandos definidos dentro desse laço serão executados. Se a condição seguir sendo verdadeira, o comando seguirá sendo executado repetidamente até que a condição em algum momento retorne **falso**. Caso a condição não seja atendida já em sua primeira execução, a estrutura de repetição será finalizada e as informações dentro dela não serão percorridas em momento algum, portanto, ao criar essa estrutura de repetição, atente-se para que o contexto inserido permita a entrada nessa estrutura no mínimo uma vez quando necessário.

Esse tipo de laço de repetição pode ser observado no diagrama a seguir, seguido por sua sintaxe em Portugol.

Diagrama	Sintaxe



(objetos/diagrama_1.png)

Símbolo arredondado que demarca o início do sistema com “Principal”, direcionando a sequência para uma estrutura “Enquanto Condição”, em que, caso a condição seja verdadeira, levará para a execução de uma tarefa. Após esse comando, o caminho estará novamente sendo direcionado à estrutura “Enquanto Condição”, possibilitando percorrer novamente o caminho de execução de tarefa caso a condição seja verdadeira. No caminho onde o retorno da condição é falso, a estrutura direciona para o símbolo arredondado que demarca o fim do sistema. Essa representação está sendo utilizada como comparativo para a sintaxe apresentada do lado direito.

```
enquanto(condição)
{
    <Instrução 1>
    <Instrução 2>
    ...
    <Instrução N>
}

<Instrução seguinte>
```

Observa-se que, inicialmente, a **condição** é avaliada. Caso o resultado da condição seja **verdadeiro**, as tarefas definidas dentro dessa estrutura serão executadas, retornando para a **condição** que será novamente avaliada. Caso o resultado da condição seja **falso**, a estrutura de repetição será finalizada, realizando, de forma imediata, as próximas linhas de código programadas.

Aplicar esse trecho em seu código reduzirá muitas linhas de código onde haveria ações repetidas. Veja nos exemplos a seguir.

Exemplo 1

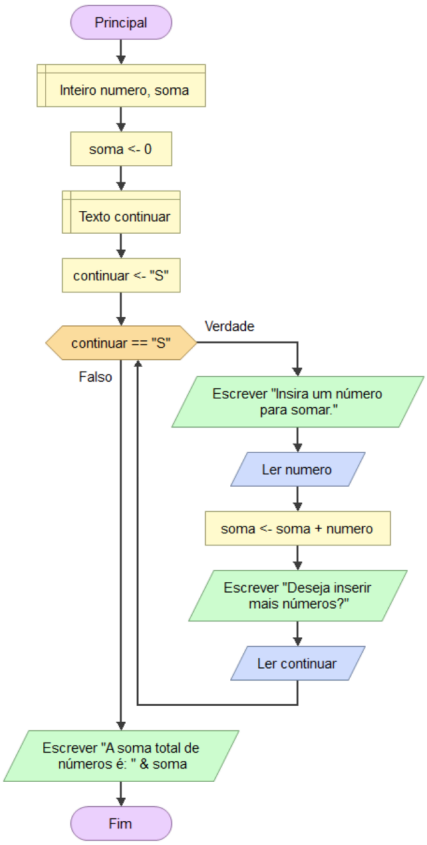
Será necessário desenvolver um código que receba vários valores inteiros e realizar a soma de todos os valores recebidos. Com o conhecimento adquirido anteriormente, seu código ficará similar ao seguinte trecho:

```
inteiro número, soma = 0

escreva("Insira um número para somar. ")
leia(número)
soma = soma + número

escreva("Insira mais um número para somar. ")
leia(número)
soma = soma + número

escreva("Insira mais um número para somar. ")
leia(número)
soma = soma + número

escreva("Insira mais um número para somar. ")
leia(número)
soma = soma + número

escreva("A soma total dos números é: ", soma)
```

Utilizando o laço de repetição **enquanto**, você não precisará escrever vários blocos de código para apresentar ao usuário a solicitação da informação e para posteriormente realizar a leitura da variável informada. Você poderá executar um mesmo bloco de comando várias vezes, utilizando somente o laço de repetição e reduzindo, dessa forma, a quantidade de linhas em seu código.

Observe a seguir o código anterior “refatorado” utilizando o laço de repetição **enquanto** com base no diagrama e na sintaxe.

Diagrama	Sintaxe




```
inteiro número, soma = 0
caracter continuar = 'S'

enquanto(continuar == 'S')
{
escreva("Insira um número p
ara somar. ")
leia(número)
soma = soma + número
escreva("Deseja inserir mai
s números? S/N")
leia(continuar)
}

escreva("A soma total dos n
úmeros é: ", soma)
```

Símbolo em elipse inicia direcionando a sequência para uma estrutura de declaração de variáveis em que são definidas as variáveis “número” e “soma” do tipo inteiro. Após essa declaração, o diagrama direciona para a estrutura de atribuição de valores, sendo atribuído o valor zero para a variável “soma”. Na sequência, a estrutura de atribuição aparece novamente, sendo criada a variável “continuar” do tipo texto. Logo após a

declaração, o diagrama direciona para a estrutura de atribuição novamente, em que o valor “S” é atribuído para a variável “continuar”. Na sequência, o diagrama entra na estrutura de repetição “enquanto”, testando a condição “continuar é igual a “S”. Se essa condição resultar em verdadeiro, o diagrama direciona para um caminho que é iniciado com a estrutura de escrita, solicitando que o usuário insira um número para somar. Na sequência, a estrutura de leitura de variáveis lê a variável “número”. Após a leitura, o diagrama segue para a estrutura de atribuição, em que, para a variável “soma”, é atribuído “soma” mais “número”, referentes às variáveis criadas anteriormente. Após essa atribuição, por meio da estrutura escrever, o usuário é questionado se deseja inserir mais números. Na sequência, o diagrama segue para a estrutura de leitura, em que lê

a variável “continuar”. Após a leitura da variável, a estrutura retorna novamente ao laço de repetição “enquanto” para testar a condição novamente. Caso ainda seja verdadeira, a sequência de passos anteriores será repetida. Caso a condição seja falsa, o diagrama direciona para a estrutura de escrita, em que o usuário é informado qual é a soma total dos números inseridos.	

No código acima, repetiu-se o bloco de comandos considerando a variável **continuar** do tipo caractere. Cada vez que o usuário digitar **‘S’** no fim do laço de repetição, simbolizando que deseja seguir inserindo números, a condição continuará sendo verdadeira, portanto, o *loop* se manterá até que o usuário informe que não deseja mais cadastrar e a variável **continuar** seja atualizada para o valor **‘N’**. Dessa forma, o cadastro não será limitado ao número de vezes em que as informações foram solicitadas via código, mas sim por meio da interação com o usuário.

São classificados como **interativos** os códigos que são similares a esse, em que a intervenção direta do usuário é necessária para que o laço de repetição continue a ser executado.

Veja também:

- Se a variável **continuar** não tivesse sido inicializada em sua declaração com o valor **‘S’** desde o início, o código não entraria nenhuma vez no laço de repetição.
- Se a linha onde foi atualizada a variável **continuar** for removida, o bloco de instruções será sempre repetido, pois o código entrará em um *loop* sem fim. A condição nunca será atualizada, portanto, essa estrutura não terá fim.
- Sem utilizar o laço de repetição, seria necessário perguntar várias vezes ao usuário as informações desejadas, aumentando o tamanho do código, como no primeiro exemplo.

Acompanhe mais um exemplo para reforçar este aprendizado.

Exemplo 2

Considere, para o próximo exemplo, que o usuário deverá inserir cinco números e realizar a soma total de cada um dos números inseridos. Posteriormente o sistema deverá apresentar ao usuário a média desses valores.

```
inteiro valor, cont = 0, soma = 0

enquanto(cont < 5)
{
escreva("Insira um valor para realizar a soma:")
leia(valor)
soma = soma + valor
cont = cont + 1
}

escreva("Você terminou de cadastrar os valores!")
escreva("A média dos valores inseridos é:" + soma/cont )
```

Novamente, nesse código, é realizada a soma de vários números inteiros. O cadastro de um novo número para a soma está sendo solicitado, entretanto, dessa vez, a quantidade de vezes que esse laço de repetição será executado está sendo limitada por um número predefinido (5). Perceba que, para finalizar esse laço de repetição, será utilizado um contador denominado “cont”.

Mas como funciona um contador, afinal? Um **contador** nada mais é do que uma variável criada para realizar a contagem de algum acontecimento dentro do seu código, seja contar quantas vezes entrou em um *loop*, quantas vezes o usuário respondeu a uma pergunta ou quantos números pares têm em um intervalo de números que você estiver testando. Se você desejar registrar a quantidade de vezes que um acontecimento específico acontece, você deverá criar uma variável para isso.

Conforme o que você desenvolver, pode incrementar (aumentar) ou decrementar (diminuir) o valor do contador toda vez que essa ação específica acontecer. Essa variável normalmente é criada para realizar o controle dos laços de repetição, sendo incrementada ao final de cada execução do *loop* e comparada para definir o ponto de finalização do bloco de código.

Nesse exemplo, a variável “cont” foi iniciada com o valor **0**. Para que seja realizada uma comparação entre os valores inteiros, é necessário que seja atribuído um valor inicial ao contador, caso contrário, o sistema não terá todos os dados necessários para testar a condição e comparar um número com o outro, portanto, o código não funcionará. Se você não inicializar a variável antes de realizar a comparação, será como comparar um número inteiro com um valor vazio.

Outro aspecto que deve receber atenção é a **condição** estabelecida para o teste no laço de repetição. Embora tenha sido definido que no código fossem cadastrados cinco números, a condição do **enquanto** é **cont < 5**. Se a opção foi de que cinco números fossem cadastrados, por que está sendo testado se o contador é menor que cinco? Isso ocorre devido ao contador ter sido inicializado com o valor 0. A primeira inserção executada será vinculada à posição 0, afinal, o contador foi

iniciado nesse valor e nenhuma inserção havia sido realizada ainda, portanto, é a inserção número 0. Com base nessa observação, pode-se notar que as inserções ocorrerão na sequência de posições 0, 1, 2, 3 e 4, somando assim as 5 posições e os 5 registros objetivados.

Neste conteúdo, a análise é voltada apenas para a percepção da lógica da contagem. Você aprenderá mais detalhes sobre a perspectiva de posicionamentos no conteúdo de estrutura de dados, em vetores e matrizes, em que as informações são registradas conforme o posicionamento e são acessadas também dessa forma.

É possível alterar a condição para cont <= 5 e inicializar a variável cont com 1 para que o contador finalize somente quando chegar ao número 5, entretanto, quando o sistema apresentar o contador final, será apresentado como se tivessem sido realizados 6 cadastros, afinal o trecho cont = cont + 1 será executado 5 vezes até que a condição do teste seja falsa e o laço de repetição acabe. Dessa forma, é necessário alterar também a apresentação dos dados ao usuário, colocando “escreva(“Você cadastrou ”, cont-1, “ usuários.”)” para que o valor apresentado esteja correto.

Após a execução do laço de repetição dentro do limite preestabelecido, o sistema escreve ao usuário que todos os cadastros foram realizados, seguido pela média dos valores inseridos por meio do cálculo de soma dividido pelo contador, afinal, a soma conterá a soma de todos os valores inseridos pelo usuário, e o contador terá o número total de vezes em que o usuário inseriu um número. Essa média está sendo realizada diretamente no comando **escreva**, entretanto, se o valor dessa média fosse utilizado em outras partes do código, o ideal seria criar uma variável para inserir o valor da média para que não fosse necessário realizar o cálculo várias vezes.

Por meio desse exemplo, observa-se que também é possível desenvolver um código de repetição que não dependa de confirmação do usuário, executando somente ações com base em informações predefinidas. Esse tipo de código é classificado como iterativo.

Embora você possa utilizar esse laço de repetição tendo um valor predefinido, o **enquanto** é aplicado com maior frequência quando não foram definidas quantas vezes o bloco de comandos deverá ser repetido, como no primeiro exemplo desse laço.

Observe mais um exemplo contendo o que você aprendeu nos conteúdos anteriores: laço de repetição **enquanto** e contadores.

Exemplo 3

O próximo exemplo é um código em que o usuário deverá inserir suas notas de provas. O sistema deverá parar de efetuar registros quando a nota inserida for negativa. Considerando-se que a média para aprovação é 7, ao final do código deverá ser apresentado ao usuário em quantas provas o aluno foi aprovado e em quantas foi reprovado.

```
real nota = 0
inteiro contAprovado = 0, contReprovado = 0

escreva("-- ATENÇÃO: Inserir um número negativo finaliza o programa -- \n")

enquanto(nota >= 0)
{
escreva("Insira a nota de sua avaliação!")
leia(nota)

se (nota >= 7)
{
  contAprovado++
}
senao se (nota >= 0 e nota < 7)
{
  contReprovado++
}
}

escreva("Você cadastrou ",contAprovado+contReprovado," provas. \n")
escreva ("Você aprovou em ", contAprovado, " provas e reprovou em ", contReprovad
o,".")
```

Nesse exemplo, é reforçado um pouco mais de todo o conteúdo visto. O código foi iniciado criando as variáveis de **nota** do tipo real para registrar até mesmo valores com casas decimais. Também foram criados **contadores** do tipo inteiro para apresentar ao usuário a quantidade de provas que o aluno foi aprovado e a quantidade de provas que o aluno foi reprovado. Logo no início do código, o usuário é informado que, ao inserir um número negativo, o programa será finalizado.

É muito importante que as informações sejam sempre apresentadas ao usuário de forma clara para que ele esteja ciente do processo.

O laço de repetição é iniciado e a partir dele seguirá a regra mencionada anteriormente. Nesse caso, o código executará o bloco de comandos pelo menos uma vez, pois a variável “nota” já foi inicializada, sendo possível testar a condição.

Como não se sabe quantas notas serão inseridas, foi definido que o laço encerrará conforme a interação do usuário.

No bloco de comandos executados dentro dele, inicia a solicitação e a leitura de dados. Com as informações obtidas, utiliza-se a condição **se** para validar se a **nota** é **maior ou igual** a 7, para que seja acrescentado **+1** ao contador de aprovação (“contAprovado”). Em contrapartida, caso o número não atenda à primeira condição, também será testado **se** o **número** é **maior ou igual** a 0 (afinal, se o número for negativo, menor que 0, deverá finalizar a execução do código) e se o número é menor do que 7.

Observe que o operador lógico **E** foi utilizado. Conforme você estudou anteriormente, isso significa que para que a **condição** seja **verdadeira**, ambas as validações devem retornar **verdadeiro**. Dessa forma, o **número** precisa ser **maior ou igual** a 0 **e menor** do que 7 para o sistema incrementar com **+1** o contador de reprovação (“contReprovado”).

Ao sair do laço de repetição, é apresentada a informação de quantas notas foram registradas (soma dos dois contadores, afinal, cada aprovação ou reprovação é equivalente a uma prova). Por fim, é apresentado o número de provas em que houve aprovação e o número de provas em que houve reprovação com base nos contadores utilizados.

Chegou a hora de praticar!

Agora que você aprendeu um pouco mais sobre essa estrutura, desenvolva os seguintes algoritmos utilizando o laço de repetição **enquanto**.

1Desenvolva um sistema que questione o nome e a idade do usuário. Não poderá ser permitido que o usuário prossiga caso insira um número negativo ou maior do que 100 para a idade. Na sequência, essas informações deverão ser apresentadas ao usuário.

2Desenvolva um sistema que apresente a soma dos 100 primeiros números naturais (1+2+3+ ... +98+99+100).

3Elabore um programa que efetue o cálculo fatorial do valor inteiro 5 e apre-sente o resultado dessa operação. Para entender esta atividade proposta, considere que, do ponto de vista matemático, fatorial é o produto dos núme-ros naturais desde 1 até o limite informado, neste caso, 5. Assim sendo, a fa-torial do valor 5, representada matematicamente como 5!, é a multiplicação de 1 x 2 x 3 x 4 x 5, que resulta no valor 120.

Para desenvolver mais sua habilidade analítica, observe o enunciado e o código a seguir. O código foi elaborado para realizar o cadastro de três equipes e suas respectivas pontuações. Entre as equipes, deverá ser testado qual foi a maior pontuação para definir um grupo vencedor. Posteriormente, deverá ser exibido ao usuário o nome da equipe vencedora, ou das equipes, em caso de empate, seguido da pontuação correspondente.

```
programa {
    função início() {

        cadeia equipeVencedora = "Empate!", equipe1, equipe2, equipe3
        inteiro pontuação, pontuaçãoVencedora = 0

        escreva("--- REGISTRO DE EQUIPES: CAMPEONATO: ---")

        escreva("Insira o nome da equipe 1:")
        leia(equipe1)

        escreva("Insira a pontuação da equipe 1:")
        leia(pontuacao)

        se(pontuação > pontuaçãoVencedora)
        {
            equipeVencedora = equipe1
            pontuaçãoVencedora = pontuação
        }
        senao se(pontuação == pontuaçãoVencedora)
        {
            equipeVencedora = "Empate!"
        }

        escreva("Insira o nome da equipe 2:")
        leia(equipe2)

        escreva("Insira a pontuação da equipe 2:")
        leia(pontuação)

        se(pontuação > pontuaçãoVencedora)
        {
            equipeVencedora = equipe2
            pontuaçãoVencedora = pontuação
        }
        senao se(pontuação == pontuaçãoVencedora)
        {
            equipeVencedora = "Empate!"
        }

        escreva("Insira o nome da equipe 3:")
        leia(equipe3)

        escreva("Insira a pontuação da equipe 3:")
        leia(pontuação)

        se(pontuação > pontuaçãoVencedora)
        {
```

```
            equipeVencedora = equipe3
            pontuaçãoVencedora = pontuação
        }
        senao se(pontuação == pontuaçãoVencedora)
        {
            equipeVencedora = "Empate!"
        }

        escreva("A equipe vencedora é: ", equipeVencedora, " | Pontos: ", pontuaç
ãoVencedora)
    }
}
```

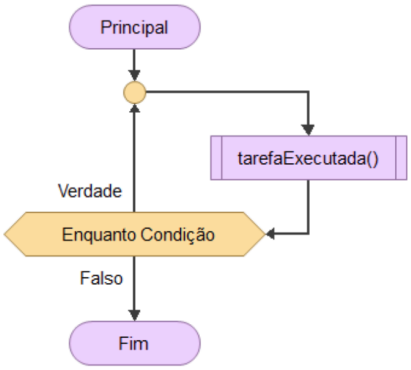
Reescreva este código utilizando o laço de repetição **enquanto** para reduzir a quantidade de linhas necessárias em seu desenvolvimento. Observe o código como um todo e deixe-o da forma mais simples possível, porém, ainda assim, atingindo o objetivo do algoritmo. Tente também acrescentar a possibilidade de o usuário escolher quantas equipes serão registradas. Faça todos os aperfeiçoamentos de código que achar interessante e depois compare seu novo código com o código apresentado anteriormente para que possa observar melhor os aprimoramentos realizados.

Estrutura de repetição pós-teste: faça enquanto (*do-while*)

A estrutura pós-teste tem seu nome autoexplicativo. Esse laço de repetição executará primeiramente a sequência de instruções definidas mesmo que nenhuma condição tenha sido verificada ainda. Após essa primeira execução automática, o código chegará à expressão condicional que será avaliada antes que o bloco de comandos possa ser executado novamente. Se a condição for verdadeira, então o fluxo de execução volta a rodar os comandos definidos para repetição, caso contrário, o comando executará os códigos seguintes à estrutura de repetição.

Essa execução pode ser melhor observada a seguir no diagrama e na sintaxe correspondente:

Diagrama	Sintaxe



(objetos/diagrama_3.png)

Forma de elipse com “início” direcionando na sequência para o começo da estrutura de um laço de repetição, em que não há nenhuma condição para ser verificada. Dessa forma, o diagrama já é direcionado para execução de tarefa. Após a execução ocorrer, o diagrama é direcionado na sequência para um teste de “enquanto condição”. Caso a condição seja verdadeira, direcionará para a execução da tarefa novamente. Caso o retorno seja falso, a estrutura será direcionada para a elipse “fim”. Essa representação está sendo utilizada como comparativo para a sintaxe apresentada à direita.

```
faça
{
    <Instrução 1>
    <Instrução 2>
    ...
    <Instrução N>
} enquanto (condição)
```

Por meio dessa representação, observa-se que a tarefa será executada primeiramente e no mínimo uma vez antes da expressão condicional ser verificada.

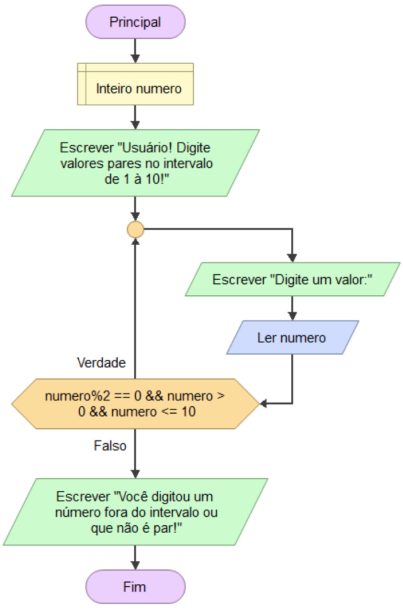
O laço **faça enquanto** também pode ser utilizado quando não se tem certeza de quantas vezes o código deverá se repetir, entretanto, é necessário que o bloco de comandos seja executado no mínimo uma vez.

Veja exemplos práticos.

Exemplo 1

Neste exemplo, o código a seguir solicitará que o usuário digite somente valores pares dentro do intervalo de 1 a 10. Caso o usuário digite um número que não atenda a essas condições, ele deverá ser informado que o número inserido é inválido.

Diagrama	Sintaxe



Forma elíptica com “início” direcionando para a estrutura de declaração de variável na sequência, em que a variável “número” do tipo inteiro foi declarada. Seguindo, o diagrama é direcionado para uma estrutura de escrita em que o usuário é informado com a seguinte mensagem: “Usuário! Digite valores pares no intervalo de 1 a 10!”. A sequência segue para o início da estrutura de um laço de repetição, em que não há nenhuma condição para ser verificada. Dessa forma, o diagrama já é direcionado para uma estrutura de escrita,

```
inteiro número

escreva("-- Usuário! Digite
valores pares no intervalo
de 1 a 10! -- \n")

faca
{
escreva("Digite um valor:
\n")
leia(número)
}enquanto (número%2 == 0 e
número > 0 e número <= 10)

escreva("Você digitou um nú
mero fora do intervalo ou q
ue não é par!")
```

apresentando ao usuário a solicitação “Digite um valor:”. Após essa solicitação, é realizada a leitura da variável “número” por meio da estrutura de leitura. O diagrama é direcionado na sequência para um teste de “enquanto condição”. Caso a condição seja verdadeira, direcionará novamente para a execução das tarefas mencionadas anteriormente. Caso o retorno seja falso, a estrutura será direcionada para uma estrutura de escrita, em que é apresentado ao usuário a seguinte mensagem: "Você digitou um número fora do intervalo ou que não é par!”. Na sequência, o diagrama direciona para o símbolo elipse com “fim”. Essa representação está sendo utilizada como comparativo para a sintaxe apresentada à direita.	

O operador % é utilizado para retornar o resto de uma divisão. Embora possa ser utilizada conforme a necessidade, é frequentemente aplicada para verificar se um número é par ou não.

O primeiro código é bastante simples. Como descrito anteriormente, é possível observar que o bloco de comandos (mensagem ao usuário e leitura de dados) está sendo executado antes de a condição ser validada.

Perceba que, neste caso, não foi necessário atribuir previamente um valor para a variável “número” logo ao declará-la, uma vez que, devido ao laço de repetição **faça enquanto** executar os comandos pelo menos uma vez, pode-se atribuir um valor para a variável dentro dos próprios comandos iniciais do laço de repetição. Quando a execução chegar à linha da validação, já existirá um valor para ser comparado. Caso esse código tenha sido feito com o comando **enquanto**, seria necessário já inicializá-la durante a criação da variável, caso contrário, o código não funcionaria, pois não seria possível utilizar um valor vazio para testar a condição.

Exemplo 2

Veja mais um exemplo!

Considere um sistema de pedidos de produto em que o usuário precise escolher entre as opções “Adicionar novo produto à sacola”, “Remover um produto da sacola” e “Finalizar pedido”. Toda vez que o usuário adicionar um produto na sacola, o sistema deverá incrementar o contador de produtos. Toda vez que o usuário remover um produto da sacola, o contador deverá ser decrementado. Ao finalizar o pedido, deverá ser informado ao usuário quantos ao usuário itens foram inseridos em sua sacola.

```
caracter opção
inteiro contProduto = 0

faca
{
escreva("Escolha uma opção: \n A) Adicionar um novo produto à sacola \n B) Remove
r um produto da sacola \n C) Confirmar Pedido \n")
leia(opcao)

se(opcao == 'A')
{
  contProduto++
  escreva("Produto adicionado com sucesso. Sua sacola possui ", contProduto, " it
ens. \n")
}
senao se(opção == 'B')
{
  se(contProduto != 0)
  {
      contProduto--
      escreva("Produto removido com sucesso. Sua sacola possui ", contProduto, "
itens. \n")
  }
  senao
  {
    escreva("Sua sacola já está vazia! \n")
  }
}

} enquanto (opcao == 'A' ou opcao == 'B')

escreva("Pedido finalizado! Sua sacola possui ", contProduto, " itens.")
```

Da mesma forma que o código anterior, não houve a necessidade de predefinir o valor da opção para que o usuário pudesse ter acesso às funções do sistema pelo menos uma vez. Sair do laço de repetição diretamente sem realizar nenhuma das outras opções disponíveis novamente é uma escolha do usuário. Dentro da estrutura está ocorrendo o **incremento** e o **decremento** da variável “contProduto”. Embora a variável “opção” tenha sido inicializada dentro do laço de repetição, o mesmo **não** pode ocorrer com o **contador**, caso contrário, sempre que o laço de repetição

iniciasse novamente, a variável seria atualizada para seu valor inicial de 0, não realizando corretamente sua função de acumulador, portanto, é muito importante que você faça uma análise de quais variáveis podem ou não ser inicializadas dentro da sua estrutura.

Dentro do laço de repetição **faça enquanto** foram utilizadas as estruturas condicionais **se** e **senão se** para redirecionar ao bloco de tarefas correspondente à opção selecionada pelo usuário (incremento e decremento do contador de itens na sacola). É possível combinar todas as estruturas lógicas conforme a necessidade. Por exemplo, acrescentar também um **enquanto** dentro da estrutura **faça enquanto** para validar a variável **opção**, conforme você pode ver no exemplo a seguir:

```
faça
{
escreva("Escolha uma opção: \n A) Adicionar um novo produto à sacola \n B) Remove
r um produto da sacola \n C) Confirmar Pedido \n")
leia(opção)

enquanto(opção != 'A' e opção != 'B' e opção != 'C')
{
   escreva("Opção inválida. Selecione uma das opções disponíveis: \n A) Adicionar
um novo produto à sacola \n B) Remover um produto da sacola \n C) Confirmar Pedid
o \n")
   leia(opção)
}

…
…
…
<Comandos seguintes>
```

Por meio dessa validação, o usuário só conseguiria prosseguir caso escolhesse corretamente uma das opções apresentadas na tela.

Não existe restrição sobre combinação de estruturas condicionais e laços de repetição. Você pode combiná-las e utilizá-las conforme se encaixe em sua necessidade, sendo necessário somente analisar se a lógica de programação está sendo aplicada de forma correta dentro do contexto do seu problema e se as estruturas utilizadas são o meio mais adequado de atingir a solução.

Exemplo 3

Observe mais um exemplo para reforçar seus conhecimentos! No código a seguir, o objetivo é realizar o cadastro de cinco *e-mails* de contato para ganhar um desconto!

```
inteiro cont = 0
cadeia email

escreva("Insira o email de 5 amigos para ganhar desconto máximo!\n")

faca
{
escreva("Insira um novo email para indicação: ")
leia(email)
escreva("Obrigada! Entraremos em contato com seu amigo(a) pelo email ", email, "
\n")
cont++
escreva("Você já cadastrou ", cont,"/5 indicações.\n\n")
}enquanto(cont < 5)

escreva("Você ganhou 50% de desconto para utilizar em sua próxima compra! Parabén
s!")
```

Desta vez o código não está utilizando a interação do usuário como forma de limitar o número de execuções do bloco de comandos. O trecho de solicitação irá se repetir obrigatoriamente mesmo que o usuário não queira cadastrar mais usuários, pois a condição é que sejam cadastrados cinco emails. Observe também que este

exemplo poderia ser facilmente executado com um laço de repetição **enquanto**. Grande parte dos exemplos podem ser elaborados com outros laços de repetição. Avalie os exemplos dos outros laços de repetição e tente refazê-los com laços de repetição diferentes!

Chegou a hora de praticar!

Agora que você aprendeu um pouco mais sobre essa estrutura, desenvolva os seguintes algoritmos:

1Desenvolva um sistema que solicite ao usuário informar um número de 1 a 10. O sistema deverá multiplicar o valor total (que será inicializado com 1) pe-lo valor inserido pelo usuário. Quando o valor total passar de 500, o sistema deverá informar ao usuário o resultado final e finalizar.

2Elabore um sistema que apresente um menu ao usuário com as opções “A - Verificar desconto do produto” e “B - Sair“. Na primeira opção, o usuário deve-rá informar o valor inicial do produto e a porcentagem do desconto. Com base nessas informações, você deverá calcular qual será o valor do desconto. Ca-so o usuário selecione a opção B, o sistema deverá encerrar.

3Elabore um programa que leia valores positivos inteiros até que um valor ne-gativo seja informado. Ao final, devem ser apresentados o maior valor e o menor valor informados pelo usuário sem considerar o número negativo in-formado para a saída do sistema.

Para desenvolver mais sua habilidade analítica, observe o enunciado e o código a seguir. O código foi elaborado para realizar o primeiro cadastro de um usuário e senha. O sistema não pode permitir que seja feito um cadastro incorreto de usuário e senha inicial. Quando o cadastro for realizado com sucesso, seja este o primeiro cadastro ou somente uma atualização, o sistema deverá apresentar qual foi o usuário e a senha cadastrados e questionar se o usuário deseja atualizar novamente as informações.

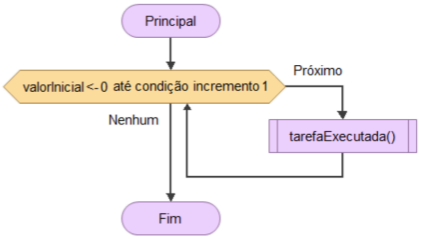
```
cadeia userAdm = "", senhaAdm = ""
caracter alterarDados = 'S'

enquanto(alterarDados == 'S')
{
    escreva ("Você deseja definir um novo nome de usuário administrador e senha?
\n")
    leia(alterarDados)

    se(alterarDados == 'N' e userAdm == "" ou senhaAdm == "")
    {
        escreva("Você precisa realizar este cadastro no mínimo uma vez. \n")

        enquanto(userAdm == "" ou senhaAdm == "")
        {

            escreva("Cadastre seu nome de usuário administrador: \n")
            leia (userAdm)
            escreva("Cadastre sua senha de acesso: \n")
            leia (senhaAdm)
        }
    escreva("Dados salvos com sucesso! | Usuário: ", userAdm, " | Senha: ", senha
Adm, " | \n")
        alterarDados = 'S'
    }
    senão se(alterarDados == 'S')
    {
        enquanto(userAdm == "" ou senhaAdm == "")
        {

            escreva("Cadastre seu nome de usuário administrador: \n")
            leia (userAdm)
            escreva("Cadastre sua senha de acesso:\n")
            leia (senhaAdm)
        }
        escreva("Dados salvos com sucesso! | Usuário: ", userAdm, " | Senha: ", s
enhaAdm, " | \n")
        escreva ("Você deseja definir um novo nome de usuário administrador e sen
ha? \n")
        leia(alterarDados)
    }
}
```

Reescreva esse código utilizando o laço de repetição **faça enquanto** para reduzir a quantidade de linhas necessárias em seu desenvolvimento. Observe o código como um todo e deixe-o da forma mais simples possível, porém, ainda assim, atingindo o objetivo do algoritmo conforme o enunciado. Não deixe de realizar a validação de campo vazio, para que o usuário cadastre os dados novamente, se necessário. Faça todos os aperfeiçoamentos de código que achar interessante e depois compare seu novo código com o código apresentado para que possa observar melhor os aprimoramentos que você fez.

Estrutura de repetição com variável de controle: *for* (para)

A estrutura de repetição com variável de controle é um pouco diferente dos laços de repetição estudados anteriormente. Na declaração dessa estrutura, já é definido um valor inicial para a comparação de condição e a respectiva condição de finalização do *loop* e, para finalizar, o valor inicial é incrementado. Você pode analisar de uma forma mais simplificada o funcionamento dessa estrutura por meio do seguinte diagrama e da seguinte sintaxe:

Diagrama	Sintaxe



(objetos/diagrama_5.png)

Símbolo elíptico com “Principal” direcionando para o início da estrutura de um laço de repetição em que a variável “valorInicial” é declarada e recebe o valor zero, a condição de execução já é definida e, por fim, a variável valorInicial é incrementada em um. Esse laço de repetição apresenta dois caminhos: o primeiro caminho é o “Próximo”, seguindo uma ordem. Nesse percurso, a tarefa definida é executada, retornando ao laço de repetição e validando a condição nela definida na sequência. Caso não haja um “Próximo”, o caminho da estrutura será “Nenhum”. Esse caminho direciona para o símbolo de elipse “Fim”. Essa

```
para (valorInicial; condiçã
o; incremento valorInicial)
{
    <Instrução 1>
    <Instrução 2>
    ...
    <Instrução N>
}
```

representação está sendo utilizada como comparativo para a sintaxe apresentada à direita.	

Nesse laço de repetição, o valor inicial serve como um contador utilizado para validar a condição. A declaração do **valor inicial**, o teste da **condição** e o **incremento do valor inicial** ocorrem na mesma linha. No momento que a **variável inicial** (variável de controle) atingir o valor definido para o fim da contagem, serão executados os comandos fora desse laço e o código seguirá normalmente.

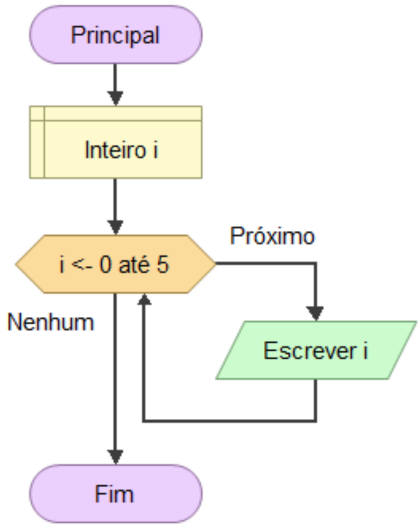
Para os laços de repetição que são iterativos, existe o hábito de utilizar a variável de controle denominada “i”. O nome “i” para variável provém do termo iteração (ou *iteration*, em inglês). Caso você veja em algum código, agora saberá a origem!

Veja os exemplos para que você possa compreender melhor.

Exemplo 1

A ideia deste código é simples: escrever para o usuário números de 0 até 5.

Diagrama	Sintaxe




```
para (inteiro i = 0; i <=
5; i++)
{
  escreva(i, "\n")
}
```

Símbolo em elipse com “Principal” direcionando para a estrutura de declaração de variável, em que a variável “i” do tipo inteiro é criada na sequência. O diagrama segue para a estrutura de repetição “para”, atribuindo o valor zero para a variável “i” e, na sequência, comparando com a condição “até 5”. Dessa forma, de 0 até 5, caso a variável “i” estiver nesse intervalo, deverá entrar na sequência “próximo” dessa estrutura de repetição e incrementar mais um à variável “i”. Nessa sequência, o diagrama é direcionado à estrutura de escrita em que a variável “i” é apresentada ao usuário. Após essa execução (única definida dentro desse

caminho), o diagrama é direcionado novamente a testar a condição definida. Caso a condição ainda seja verdadeira, repetirá o conteúdo do caminho “Próximo”. Caso o valor retorne falso, o caminho “Nenhum” será executado e o diagrama é direcionado para o símbolo de elipse “Fim”. Essa representação está sendo utilizada como comparativo para a sintaxe apresentada à direita.	

A estrutura desse laço de repetição pode ser observada nesse pequeno trecho.

Embora esse código possa ser alterado conforme a necessidade, a forma padrão em que costuma ser utilizado é estruturada da seguinte maneira: como mencionado, primeiramente serão necessárias a criação e a inicialização de uma variável dentro da declaração do laço de repetição. No exemplo anterior, a variável “i” é declarada como valor inicial, sendo atribuído o valor 0 na sequência. Essa variável também funcionará como contador em nosso laço de repetição.

```
para (inteiro i = 0; i <= 5; i++)
```

Após isso, é também necessário definir um valor final por meio da condição, ou seja, definir um valor que limite quantas vezes essa estrutura poderá ser executada, para que o laço de repetição seja finalizado em algum momento. Conforme o exemplo, o código será finalizado quando apresentar o número 5 ao usuário, que é o valor final. Com o valor inicial e o valor final definidos, o laço de repetição poderá testar se o valor inicial é menor ou igual ao valor final. Se a condição for **verdadeira**, os comandos dentro do laço de repetição serão executados.

```
para (inteiro i = 0; i <= 5; i++)
```

Ao final dessa execução, a variável inicial é incrementada recebendo +1 em seu valor. Caso o valor inicial seja maior do que o valor final, a estrutura encerra, executando os comandos programados após o laço de repetição. Diferentemente dos outros laços de repetição, não será necessário adicionar um contador dentro do bloco de código para controlar a quantidade de repetições, afinal, a própria estrutura trata de criar a variável, incrementá-la e encerrar quando for o momento certo.

```
para (inteiro i = 0; i <= 5; i++)
```

É possível obter o valor final da condição por meio de interação com o usuário no código a seguir. O objetivo aqui segue sendo apresentar os números ao usuário de forma ordenada.

```
inteiro numero

escreva("Até qual número você deseja contar?")
leia(numero)

para (inteiro i = 0; i <= numero; i++)
{
escreva(i, "\n")
}
```

A única alteração feita nesse código em relação ao exemplo anterior foi a inserção da variável “numero” na condição. Ela é a variável que limitará a quantidade de vezes que o laço será repetido. O processo desse laço será repetido tal como o exemplo anterior: para “i” iniciando com o valor 0, **se** a variável “i” for **menor ou igual** ao **valor final** (“numero”), o bloco de comandos será executado. Após a execução, deve-se **incrementar** a variável “i”. Observe que, com base no valor inicial, na condição e no incremento, os números serão apresentados aos usuários de forma crescente.

Neste trecho de código a seguir, o objetivo foi um pouco diferente: apresentar os números para o usuário em ordem decrescente.

```
inteiro numero

escreva("Qual número você deseja contar de forma decrescente até 0?")
leia(numero)

para (inteiro i = numero; i >= 0; i--)
{
escreva(i, "\n")
}
```

Embora o laço de repetição iniciado em 0 e o incremento do contador possam representar um dos formatos mais comuns para a estrutura do laço de repetição **para**, é possível fazer alterações. Esse último trecho é um claro exemplo disso. Se o número que o usuário escolheu foi 10 e os dados forem apresentados de forma decrescente, o código deverá decrementar o valor do contador até que chegue a 0 ao mesmo tempo que apresenta os números um por um ao usuário. Também é possível incrementar ou decrementar mais do que 1, fazendo com que o trecho pule alguns valores, por exemplo.

Embora seja possível explorar muitos aspectos dessa estrutura, é necessário ter cuidado, afinal você pode perder dados essenciais para a execução do seu projeto com uma alteração desnecessária, uma vez que o laço de repetição não estará funcionando dentro de sua forma padrão.

O laço de repetição **para** deve ser utilizado sempre que você souber a quantidade exata de execuções que deverão ser feitas, uma vez que não dependerá da interação com o usuário para finalizar.

Exemplo 2

Acompanhe o exemplo a seguir. O código informa quais números no intervalo de 0 até 25 são ímpares. Após isso, é solicitado o nome de usuários com idade correspondente ao tipo de número testado.

```
inteiro cont = 0

para(inteiro i = 0; i <= 25; i++)
{
se(i%2 == 0)
{
  escreva("O número ", i, " é par.\n")
  escreva("Escreva o nome de um usuário com idade par! \n")
}
senao
{
  escreva("O número ", i, " é impar.\n")
  escreva("Escreva o nome de um usuário com idade par! \n")
}
}
escreva(“Cadastros finalizados com sucesso!")
```

Observa-se, no exemplo acima, que é possível utilizar as posições para realizar outros testes dentro do código ou até mesmo realizar comandos condicionais relacionados à posição atual (i). Da mesma forma que os outros laços de repetição, é possível aplicar qualquer conteúdo dentro do bloco de comandos do laço de repetição. O laço repetirá desde que a estrutura padrão valorInicial/condição/incremento esteja definida corretamente.

Esse tipo de lógica pode ser utilizado em diversos contextos diferentes. Considere, por exemplo, que você deseja apresentar uma lista zebrada ao usuário. Nos cadastros pares, poderia ser definido que o campo da informação tem uma cor diferente da padrão. Dessa forma, destacaria uma linha sim e outra não de sua lista, facilitando a visualização.

Exemplo 3

Veja um código um pouco mais complexo para analisar seu funcionamento.

No exemplo a seguir, é apresentada ao usuário a tabuada de 1 a 10. Diferentemente do exercício no início deste conteúdo, não será solicitada nenhuma informação ao usuário e será apresentada cada uma das tabuadas do valor preestabelecido.

```
para (inteiro i = 1; i < 10; i++)
{
escreva(" ------- Tabuada do ", i, ": ------- \n")

para (inteiro j = 1; j <= 10; j++)
{
  escreva(i ," x ", j ," = ", i*j, "\n")
}
}
```

Na resolução desse exemplo, são utilizados dois laços de repetição **para**, um dentro do outro! No primeiro laço de repetição, escreva ao usuário qual tabuada será apresentada, utilizando o valor inicial (**i**) como referência da informação. Na sequência, um novo laço de repetição é iniciado. Note que, para que o primeiro laço de repetição siga sua execução novamente, é necessário que o laço de repetição interno seja concluído primeiro.

Dentro da estrutura de repetição interna **para**, são utilizadas tanto a variável do contador **i** quanto a variável do contador **j** para realizar o cálculo da tabuada. Como a tabuada inicia-se no valor 1, a variável inicial de ambos os laços de repetição foram iniciadas em 1.

A variável **i** só mudará seu valor quando o segundo laço de repetição e a variável **j** percorrerem os valores de 1 a 10, finalizando sua execução. Dessa forma, você pode utilizar a variável **i** como base para a multiplicação da variável **j**, uma vez que ela não está sendo alterada durante a execução do segundo laço de repetição, e o segundo laço está incrementando o valor da variável **j** a cada execução de 1 até 10.

O resultado apresentado, conforme a formatação inserida no “escreva”, ficará similar ao seguinte trecho, sendo repetido para todos os outros números dentro do intervalo de 1 a 10.

------- Tabuada do 1: -------
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
…

O conteúdo sobre utilização do laço de repetição para dentro de outro laço de repetição, para auxiliar na realização do acesso a informações em matrizes, será aprofundado posteriormente.

Chegou a hora de praticar!

Agora que você aprendeu um pouco mais sobre essa estrutura, desenvolva os seguintes algoritmos:

1Escreva um algoritmo que apresente de forma decrescente somente valores ímpares entre 1 a 10.

2Elabore um programa que apresente o somatório dos valores pares existentes no intervalo de 1 até 500.

3Elabore um programa que apresente os resultados da soma e da média arit-mética dos valores pares situados na faixa numérica de 50 até 70 e apresente quantos números pares foram encontrados neste intervalo.

Para desenvolver mais sua habilidade analítica, observe os enunciados e os códigos a seguir.

O código elaborado deve realizar o cadastro de 5 músicas e seus respectivos estilos, obrigatoriamente. Cada estilo cadastrado deverá ser contabilizado para apresentar posteriormente um relatório de quantas vezes um estilo foi inserido.

```
cadeia nomeMusica
caracter resposta = 'S', estilo
inteiro cont = 0, contRock = 0, contSertanejo = 0, contPop = 0

escreva("--- Vamos criar uma playlist de 5 músicas? --- \n")

enquanto(resposta == 'S')
{
    escreva("Insira uma música para sua playlist: \n")
    leia(nomeMusica)
    escreva("Qual o estilo desta música? \n")
    leia(estilo)

    se(estilo == 'R')
    {
        contRock++
        cont++
    }

    se(estilo == 'S')
    {
        contSertanejo++
        cont++
    }

    se(estilo == 'P')
    {
        contPop++
        cont++
    }

    escreva("A música adicionada foi: ",nomeMusica, " do estilo ",estilo,". \n")
    escreva("Você já cadastrou ",cont ," músicas. Deseja cadastrar mais? S/N \n")
    leia(resposta)
}

escreva("Você adicionou ", contRock, "músicas do estilo Rock, ", contSertanejo,"
do estilo Sertanejo e ", contPop, " músicas estilo Pop.")
```

Reescreva este código utilizando o laço de repetição **para** como forma de reduzir a quantidade de linhas necessárias em seu desenvolvimento. Observe o código como um todo e deixe-o da forma mais simples possível, porém, ainda assim, atingindo o objetivo do algoritmo conforme o enunciado. Faça todos os aperfeiçoamentos de código que achar interessante, incluindo aspectos que aprendeu anteriormente. Depois compare seu novo código com o código apresentado acima para que possa observar melhor os aprimoramentos que você fez.

O código abaixo foi elaborado para realizar uma votação em um concurso a fim de verificar quais dos três participantes passarão para a próxima fase do teste. Cada participante deve receber três votos. Para ser aprovado para a próxima etapa, o participante deve ter pelo menos 2 votos de aprovação. Após a leitura dos votos, deverá ser apresentado ao participante se ele foi aprovado para a próxima etapa ou não. Somente após essa informação ser apresentada, a contagem de votos deverá ser realizada novamente. Após todos os participantes obterem seus resultados, deverá ser questionado ao usuário como foi sua experiência com o *software*, em uma escala de 1 a 5. Essa questão deve ser obrigatoriamente respondida ao menos uma vez, e o usuário deverá ter no mínimo uma possibilidade de alterar sua resposta, podendo alterar mais de uma vez.

```
caracter avaliacao, atualizacao
inteiro aprovado = 0, nota, numAvaliacao = 1

escreva("---- Votação de aprovados para próxima etapa: ---- \n")
escreva(" 1- João \n 2- Maria \n 3- Ana \n\n")


escreva("Insira a avaliação de 3 usuários para verificar se o participante 1 será
aprovado para a próxima etapa.\n\n")

enquanto(numAvaliacao < 3)
{
    escreva("Respostas disponíveis: A - Aprovado(a) | R - Reprovado(a)\n\n")
    escreva("Insira a avaliação número ", numAvaliacao," do participante 1:")
    leia(avaliacao)
    se(avaliacao == 'A')
    {
    aprovado++
    }
}

se(aprovado < 2)
{
    escreva("Participante número 1 você não foi aprovado(a) para a próxima etap
a.\n")
    aprovado = 0
}
senao
{
    escreva("Participante número 1 você foi aprovado(a) para a próxima etapa, par
abéns!\n")
    aprovado = 0
}

numAvaliacao = 0

enquanto(numAvaliacao < 3)
{
    escreva("Respostas disponíveis: A - Aprovado(a) | R - Reprovado(a)\n\n")
    escreva("Insira a avaliação número ", numAvaliacao," do participante 2:")
    leia(avaliacao)
    se(avaliacao == 'A')
    {
    aprovado++
    }
}

se(aprovado < 2)
```

```
{
    escreva("Participante número 2 você não foi aprovado(a) para a próxima etap
a.\n")
    aprovado = 0
}
senao
{
    escreva("Participante número 2 você foi aprovado(a) para a próxima etapa, par
abéns!\n")
    aprovado = 0

}

numAvaliacao = 0

enquanto(numAvaliacao < 3)
{
    escreva("Respostas disponíveis: A - Aprovado(a) | R - Reprovado(a)\n\n")
    escreva("Insira a avaliação número 2 do participante 3:")
    leia(avaliacao)
    se(avaliacao == 'A')
    {
    aprovado++
    }
}

se(aprovado < 2)
{
    escreva("Participante número 3 você não foi aprovado(a) para a próxima etap
a.\n")
    aprovado = 0
}
senao
{
    escreva("Participante número 3 você foi aprovado(a) para a próxima etapa, par
abéns!\n")
    aprovado = 0
}

numAvaliacao = 0

escreva("Em uma escala de 1 a 5, o quanto você gostou da sua experiência conosco?
")
leia(nota)
escreva("Sua nota foi: ",nota,". Deseja alterar sua nota de experiência? Responda
com S/N: ")
leia(atualizacao)
se(atualizacao == 'S')
{
    escreva("Em uma escala de 1 a 5, o quanto você gostou da sua experiência cono
```

```
sco? ")
    leia(nota)
}
```

Reescreva este código utilizando como principal laço de repetição a estrutura **para**, reduzindo a quantidade de linhas necessárias em seu desenvolvimento. Utilize o laço de repetição **enquanto** somente para validar os campos em que o usuário realizará a interação. Para realizar a parte de escala, utilize o laço de repetição **faça enquanto**. Observe o código como um todo e deixe-o da forma mais simples possível, porém, ainda assim, atingindo o objetivo do algoritmo conforme o enunciado. Existem trechos com código desnecessário, portanto preste atenção nos detalhes enquanto aplica os laços de repetição. Faça todos os aperfeiçoamentos de código que achar interessante utilizando também os conhecimentos que aprendeu anteriormente. Após concluir o desafio, compare seu novo código com o código apresentado acima para que possa observar melhor os aprimoramentos que você fez.

Apesar de haver situações específicas em que é mais adequado utilizar determinado laço de repetição, muitas vezes é obtido o mesmo resultado com um laço de repetição diferente. É importante analisar o caso e ver qual tipo de estrutura se encaixa melhor nas necessidades e adaptar as possibilidades conforme necessário. Estude os exemplos e os exercícios propostos, afinal, o conteúdo desenvolvido aqui será fundamental para sua futura carreira!

Assista ao vídeo com exemplos do conteúdo estudado até aqui.


CH presencial:
Recurso