Estrutura de repetição enquanto. (pseudocódigo)

Apresentação

A execução do mesmo conjunto de comandos mais de uma vez é chamada de laço. As linguagens de programação possuem algumas estruturas que permitem a definição de laços, que são chamadas de estruturas de repetição.

Nesta Unidade de Aprendizagem, estudaremos a construção de pseudocódigos com a utilização da estrutura de repetição "enquanto" (em inglês, *while*), que executa um conjunto de comandos enquanto uma condição for verdadeira.

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Identificar problemas que precisem de repetição (laços) para construção da solução algorítmica.
- Construir pseudocódigos que utilizem a estrutura de repetição "enquanto".
- Resolver problemas através da especificação com algoritmos.

Desafio

Realizamos muitos processos repetitivos no cotidiano, como caminhar enquanto não chegamos ao nosso destino, contar moedas enquanto temos moedas no cofre, aguardar enquanto o semáforo não fica verde, entre outras situações.

Entender como funciona a repetição é uma competência desejável para os programadores. A estrutura de repetição “enquanto” é iniciada por um teste (<condição>). Se o resultado for verdadeiro (SIM), executa o conjunto de comandos da repetição e retorna o fluxo para antes da condição, mas, se o resultado for falso (NÃO), sai da repetição e continua o fluxo do programa.

Suponha que você deseje construir um programa que escreva os números inteiros de 1 até um número que você informe (limite). Qual pode ser o processo para resolver esse problema? Primeiro, deve ser informado o valor limite; depois, você pode atribuir 1 (um) para uma variável, mostrar seu valor, somar 1 (um) e repetir o processo enquanto o valor da variável for menor ou igual ao limite. O algoritmo em pseudocódigo a seguir apresenta a solução desse problema:

```
algoritmo "repetindo"
var
  numero:numerico
  limite: numerico
  inicio
  leia(limite)
  numero<- 1
  enquanto numero <=limite faca
    escreval(numero)
    numero<- numero + 1
  fimenquanto
fimalgoritmo
```

Padrão de resposta esperado
O algoritmo em pseudocódigo a seguir apresenta a solução do problema de ler a taxa de câmbio de dólar para real. Na sequência, fica lendo um valor em dólar e apresentando sua conversão para real, enquanto o valor digitado for diferente de 0 (zero).

```
algoritmo "taxas"
var
  valor, taxa, brasil:numerico
  inicio
  escreval("Digite a taxa de cambio do dolar:")
  leia(taxa)
  valor<-1
  enquanto valor <> 0 faca
    escreval("Digite o valor em dolar:")
    leia(valor)
    brasil<-valor*taxa
    escreval("em Real: ",brasil)
  fimenquanto
fimalgoritmo
```

Agora é a sua vez!

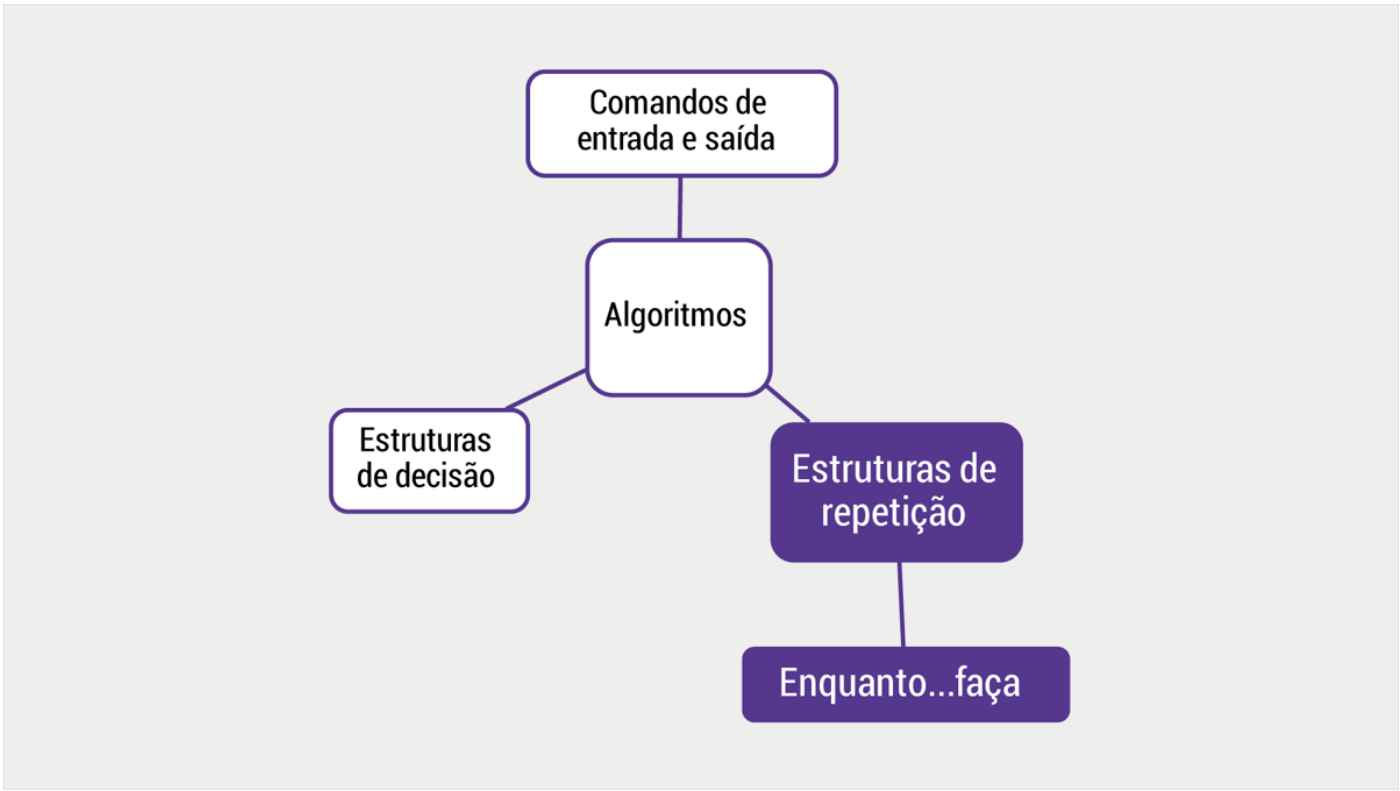
Você viajou para os Estados Unidos nas férias e a moeda utilizada por eles é o dólar, que varia conforme a taxa de câmbio do dia. Você não está acostumado com o valor dos produtos em dólar e

precisa convertê-los para o real para ter noção do custo dos produtos. Assim, quando foi a uma feira de produtos, precisava consultar constantemente o valor convertido de dólares para reais.

Construa um algoritmo em pseudocódigo que leia a taxa de câmbio de dólar para real e, a seguir, leia o valor de um produto em dólar e apresente a sua conversão para real. Além disso, é preciso que o algoritmo fique executando enquanto o valor digitado em dólar for diferente de 0 (zero), tornando, assim, a consulta dos valores dos produtos muito prática, mostrando o valor convertido de dólar para real. Utilize a estrutura de repetição “enquanto” para implementar a repetição.

Infográfico

Para a construção de algoritmos, as estruturas básicas utilizadas são os comandos de entrada e saída de dados, os comandos de decisão e os comandos para repetição. O comando "enquanto" é uma importante ferramenta para a construção de algoritmos em pseudocódigo.



Conteúdo do Livro

A estrutura de repetição "enquanto...faça" é muito utilizada no desenvolvimento de algoritmos. Sua adequada utilização demonstra domínio da competência de programação.

Conheça um pouco mais sobre esse conteúdo lendo as páginas 134 a 139 do seguinte livro: EDELWEISS, N.; LIVI, M.A.C. *Algoritmos e programação com exemplos em Pascal e C* - Vol. 23. Série Livros Didáticos Informática UFRGS. Porto Alegre: Bookman, 2014.

algoritmos e programação

com exemplos em Pascal e C

nina edelweiss
maria aparecida castro livi

bookman

as autoras

Nina Edelweiss é engenheira eletricista e doutora em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Durante muitos anos, lecionou em cursos de Engenharia e de Ciência da Computação na UFRGS, na UFSC e na PUCRS. Foi, ainda, orientadora do Programa de Pós-Graduação em Ciência da Computação da UFRGS. É coautora de três livros, tendo publicado diversos artigos em periódicos e em anais de congressos nacionais e internacionais. Participou de diversos projetos de pesquisa financiados por agências de fomento como CNPq e FAPERGS, desenvolvendo pesquisas nas áreas de bancos de dados e desenvolvimento de software.

Maria Aparecida Castro Livi é licenciada e bacharel em Letras, e mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Desenvolveu sua carreira profissional na UFRGS, onde foi programadora e analista de sistema, antes de ingressar na carreira docente. Ministrou por vários anos a disciplina de Algoritmos e Programação para alunos dos cursos de Engenharia da Computação e Ciência da Computação. Sua área de interesse prioritário é o ensino de Linguagens de Programação, tanto de forma presencial quanto a distância.

E22a Edelweiss, Nina.
Algoritmos e programação com exemplos em Pascal e C [recurso eletrônico] / Nina Edelweiss, Maria Aparecida Castro Livi. – Dados eletrônicos. – Porto Alegre : Bookman, 2014.

Editado também como livro impresso em 2014.
ISBN 978-85-8260-190-7

1. Informática. 2. Algoritmos – Programação. I. Livi, Maria Aparecida Castro. II. Título.

CDU 004.421

Catalogação na publicação: Ana Paula M. Magnus – CRB 10/2052

5.3 → comando de repetição condicional `enquanto/faça` por avaliação prévia de condição

Existem situações em que as repetições não estão condicionadas a um número definido de vezes. Por exemplo, em uma loja, não é possível saber quantas vendas ocorrerão ao longo de um dia, mas se sabe que essas serão encerradas no horário definido para que a loja feche suas portas. O processamento de cada venda (registro do valor da venda, do vendedor que efetuou a venda, etc.) é semelhante, compondo um conjunto de comandos a serem repetidos. A repetição nesse caso está condicionada à ocorrência de duas condições: venda efetuada e horário adequado.

O **comando de repetição condicional `enquanto/faça`** faz que um comando, simples ou composto, tenha sua execução condicionada ao resultado de uma expressão lógica, isto é, a execução desse comando é repetida enquanto o valor lógico resultante da avaliação da expressão de controle for verdadeiro. A sintaxe de um comando de repetição condicional `enquanto/faça` é:

```
enquanto <expressão lógica> faça
   <comando>
```

A Figura 5.4 representa o fluxograma referente ao comando de repetição condicional `enquanto/faça`.

O laço nunca será executado caso o valor inicial da expressão lógica seja falso logo de início, já que a avaliação da condição de controle ocorre antes da execução do comando a ser repe-

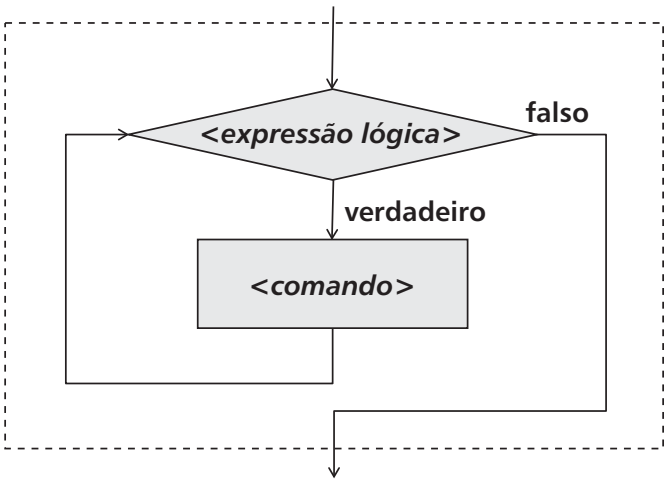


figura 5.4 Fluxograma do comando de repetição `enquanto/faça`.

tido. Se o valor inicial da expressão lógica de controle for verdadeiro, é necessário que algum componente dessa expressão seja alterado dentro do laço para que ela, em algum momento, seja avaliada como falsa, fazendo que a repetição encerre, evitando assim a ocorrência de um laço – *loop* – infinito. Portanto, esse comando somente é indicado para situações em que a expressão de controle inclui algum dado a ser modificado pelo programa dentro do próprio laço, determinando assim o encerramento das repetições.

No trecho de programa a seguir, a variável `a` é inicializada com 1 e, com esse valor, é iniciada a execução do comando `enquanto/faça`. A cada repetição do laço, `a` é incrementada em uma unidade e isso se repete enquanto `a` for inferior a 5. Quando `a` atingir o valor 5, o comando `enquanto/faça` será encerrado, sendo executado o próximo comando, que imprime o valor final de `a`, que é 5.

```
a ← 1
enquanto a < 5 faça
  a ← a + 1
escrever (a)
```

Variáveis lógicas são muito utilizadas na expressão lógica de comandos `enquanto/faça`. Por exemplo, no trecho de programa a seguir, a leitura e a escrita de um valor são repetidas até que seja lido o valor zero. A variável lógica `segue` é inicializada com verdadeiro antes do comando `enquanto/faça` e é testada a cada repetição, tornando-se falsa no momento em que é lido o valor zero:

```
segue ← verdadeiro
enquanto segue faça
  início
  ler(valor)
  se valor ≠ 0
  então escrever(valor)
  senão segue ← falso
  fim
```

5.3.1 sinalização de final de dados

Para exemplificar o uso do comando `enquanto/faça`, considere uma situação em que o número de alunos que terão as notas processadas é desconhecido. Neste caso, como saber quando encerrar o processo de leitura de notas, cálculo de média e informação da média obtida pelo aluno? O encerramento de um comando de repetição condicional que inclui a entrada dos dados a serem processados pode ser implementado de três formas, apresentadas a seguir.

marca de parada no último dado válido. Define-se, dentre os valores a serem lidos, qual o último valor que deve ser processado, valor esse que funciona como marca de parada. Como essa marca de parada é um dado válido, só depois de processá-la e de processar os demais dados a ela associados é que o processamento deve ser encerrado. No exemplo das notas de alunos, esse controle poderia ser o código do último aluno a ser processado, como mostrado no Algoritmo 5.3, que lê as notas e o código de um conjunto de alunos e informa as suas médias:

```
Algoritmo 5.3 - MédiaAlunos_1
{INFORMA MÉDIA DOS ALUNOS DE UMA TURMA}
{CONDIÇÃO DE PARADA: CÓDIGO DO ÚLTIMO ALUNO}
  Entradas: nota1, nota2, nota3 (real)
            código, cod_último (inteiro)
  Saídas: média (real)
início
  ler(cod_último)              {ENTRADA DO CÓDIGO DO ÚLTIMO ALUNO}
  código ← 0 {INICIALIZA CÓDIGO SÓ PARA COMPARAR A PRIMEIRA VEZ}
  enquanto código ≠ cod_último faça
    início
    ler(código)                                {LÊ CÓDIGO DO ALUNO}
    ler(nota1, nota2, nota3)                 {ENTRADA DAS 3 NOTAS}
    média ← (nota1 + nota2 + nota3) / 3              {CALCULA MÉDIA}
    escrever(código, média) {INFORMA CÓDIGO DO ALUNO E SUA MÉDIA}
    fim
fim
```

marca de parada após os dados válidos. É definido um valor de parada que não constitui um dado válido. Esse valor não deve ser processado, funcionando somente como indicação de parada das repetições. Por exemplo, no caso dos alunos pode ser uma primeira nota negativa:

```
Algoritmo 5.4 - MédiaAlunos_2
{INFORMA MÉDIA DOS ALUNOS DE UMA TURMA}
{CONDIÇÃO DE PARADA: PRIMEIRA NOTA LIDA É NEGATIVA}
  Entradas: nota1, nota2, nota3 (real)
```

```
          código (inteiro)
  Saídas: média (real)
início
  ler (nota1, nota2, nota3)                    {ENTRADA DE 3 NOTAS}
  enquanto nota1 ≥ 0 faça
    início
    ler(código)                                {LÊ CÓDIGO DO ALUNO}
    média ← (nota1 + nota2 + nota3) / 3          {CALCULA MÉDIA}
    escrever(código, média) {INFORMA CÓDIGO DO ALUNO E SUA MÉDIA}
    ler(nota1, nota2, nota3)       {ENTRADA DAS PRÓXIMAS 3 NOTAS}
    fim
fim
```

Observar que foi necessário ler as notas do primeiro aluno antes de entrar no comando de repetição para que o teste do comando `enquanto/faça` pudesse ser realizado adequadamente já na sua primeira execução. Ao final do processamento das notas de um aluno, são lidas as do próximo, devolvendo o controle ao comando `enquanto/faça` para que seja realizado novamente o teste da primeira nota, definindo se vai ser realizada nova repetição ou se o comando deve ser terminado. Assim tem-se a garantia de que não são processadas, como se fossem dados válidos, as notas que contêm a marca de parada.

parada solicitada pelo usuário. Ao final de cada iteração, o usuário decide se deseja continuar ou parar, respondendo a uma pergunta explícita, conforme mostrado no Algoritmo 5.5:

```
Algoritmo 5.5 – MédiaAlunos_3
{INFORMA MÉDIA DOS ALUNOS DE UMA TURMA}
{CONDIÇÃO DE PARADA: INFORMADA PELO USUÁRIO}
  Entradas: nota1, nota2, nota3 (real)
            código (inteiro)
            continuar (caractere)
  Saídas: média (real)
início
  continuar ← 'S'{INICIALIZA CÓDIGO PARA COMPARAR A PRIMEIRA VEZ}
  enquanto continuar = 'S' faça
    início
    ler(código)                                 {LÊ CÓDIGO DO ALUNO}
    ler(nota1, nota2, nota3)                  {ENTRADA DAS 3 NOTAS}
    média ← (nota1 + nota2 + nota3) / 3           {CALCULA MÉDIA}
    escrever(código, média) {INFORMA CÓDIGO DO ALUNO E SUA MÉDIA}
    escrever('Mais alunos? Responda S ou N ')
    ler(continuar)           {USUÁRIO INFORMA SE TEM MAIS ALUNOS}
    fim
  fim
```

5.3.2 contagem de repetições

Caso se necessite saber quantas repetições foram realizadas, uma vez que esse número é desconhecido, é preciso fazer uso de uma variável do tipo `contador`, incrementada dentro do laço a cada iteração. O algoritmo a seguir estende o Algoritmo 5.4, informando também a média da turma. Para o cálculo dessa média é necessário conhecer o número de alunos, informado através do contador `cont_al`:

```
Algoritmo 5.6 - MédiaAlunoETurma_2
{INFORMA MÉDIA DOS ALUNOS DE UMA TURMA E A MÉDIA GERAL DESSA TURMA.
PARA INDICAR FIM DE PROCESSAMENTO, O CONTEÚDO INFORMADO EM NOTA1 SERÁ
NEGATIVO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          soma_médias (real)
          média_turma (real)
  Variável auxiliar:
      cont_al (inteiro) {CONTADOR DE ALUNOS PROCESSADOS}
início
  soma_médias ← 0  {SOMA MÉDIAS INDIVIDUAIS: VALOR INICIAL ZERO}
  cont_al ← 0           {CONTADOR DE ALUNOS: VALOR INICIAL ZERO}
  ler(nota1, nota2, nota3)        {ENTRADA DAS 3 PRIMEIRAS NOTAS}
  enquanto nota1 ≥ 0 faça
    início
    cont_al ← cont_al + 1                 {CONTA ALUNO LIDO}
    média ← (nota1 + nota2 + nota3) / 3      {CALCULA MÉDIA}
    escrever(cont_al, média)                 {INFORMA MÉDIA}
    soma_médias ← soma_médias + média      {SOMA DAS MÉDIAS}
    ler(nota1, nota2, nota3)   {ENTRADA DAS PRÓXIMAS 3 NOTAS}
    fim {DO ENQUANTO}
  média_turma ← soma_médias / cont_al       {MÉDIA DA TURMA}
  escrever(média_turma)
fim
```

5.3.3 comandos de repetição aninhados

Assim como para o comando `para/faça`, o comando incluído dentro de um laço de repetições do comando `enquanto/faça` pode ser um comando qualquer, inclusive outro comando de repetição `para/faça` ou `enquanto/faça`. O exemplo a seguir mostra o aninhamento de um comando `para/faça` dentro de um `enquanto/faça`. A variável `mais_um` é do tipo `caractere`. Os comandos do laço do `enquanto/faça` serão repetidos enquanto não for lido um caractere "N". A cada repetição, todo o comando `para/faça` é executado:

```
ler(mais_um)
enquanto mais_um ≠ 'N' faça
  início
  ler(lim_sup)
  para i de 1 incr 1 até lim_sup faça
    escrever(i)
  ler(mais_um)
  fim
```

5.4 → comando de repetição condicional `repita/até` por avaliação posterior de condição

O **comando de repetição condicional** por avaliação posterior **`repita/até`** também vincula a execução de um conjunto de comandos ao resultado da avaliação de uma expressão lógica. O comando inicia pela execução do laço e, quando essa execução é concluída, a expressão é avaliada: se o valor lógico obtido for falso, o laço é executado novamente; se for verdadeiro, o comando é encerrado. Isso significa que o laço é sempre executado pelo menos uma vez, independentemente do valor lógico inicial resultante da avaliação da expressão de controle. Observar que, normalmente, o valor inicial da expressão lógica será falso, pois se deseja repetir o laço mais de uma vez. Portanto, é necessário que, em algum momento, o conteúdo de alguma variável utilizada nesta expressão lógica tenha o valor alterado dentro do laço, de forma a modificar o valor resultante de sua avaliação para verdadeiro, evitando assim a ocorrência de um laço – *loop* – infinito.

A sintaxe de um comando de repetição condicional `repita/até` é a seguinte:

```
repita
   <comandos>
até <expressão lógica>
```

Observar que, diferentemente dos comandos anteriores, aqui não é necessário um comando composto, pois a sintaxe aceita múltiplos comandos, delimitados pela cláusula `até`.

O fluxograma representado na Figura 5.5 mostra o funcionamento desse comando, em que o laço de repetição é sempre executado pelo menos uma vez.

O aninhamento de comandos de repetição também se aplica ao comando `repita/até`, incluindo os outros comandos de repetição já vistos.

O Algoritmo 5.7, a seguir, adapta o Algoritmo 5.6, utilizando no laço de repetição um comando `repita/até` em lugar do `enquanto/faça`. Observar que, como o laço desse comando é sempre executado pelo menos uma vez, se tornou necessária a inclusão de um comando condicional logo no início para condicionar a execução do laço ao valor inicial de `nota1`.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Dica do Professor

Vamos aprender como construir pseudocódigos utilizando a estrutura de repetição "enquanto...faça"? Confira no vídeo!

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

1) **Considere o seguinte algoritmo em pseudocódigo que lê um numeral em decimal e converte para a sua representação em binário:**

```
Algoritmo "converte"
Var numero, decimal, resto, binario, posicao : inteiro
Inicio
 Escreval("Digite um numero em decimal: ")
 Leia(numero)
 enquanto (numero < 0) faca
  Escreval("Digite um numero em decimal: ")
  Leia(numero)
 fimenquanto
 decimal <- numero
 binario <- 0
 posicao <- 1
 enquanto (decimal > 0) faca
  resto <- decimal mod 2
  binario <- binario + (resto * posicao)
  posicao <- posicao * 10
  decimal <- decimal div 2
 fimenquanto
 Escreval("Numero em decimal: ",numero," | Numero em binario: ",binario)
FimAlgoritmo
```

Analise as alternativas a seguir e assinale a falsa.

A) Se for digitado o valor 7 para "numero", na execução do comando leia(numero), a variável "posicao" receberá, durante a execução do algoritmo, o seguinte conjunto de valores: conjunto de valores {1,10,100, 1000}.

B) Se for digitado o valor 63 para "numero", na execução do comando leia(numero), ao final do algoritmo será informado:
Numero em decimal: 63 | Numero em binario: 111111.

C) Durante a execução desse algoritmo, a variável "resto" poderá receber qualquer valor no intervalo [0,9]. Ou seja, conforme os valores de entrada, "resto" poderá receber 0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9.

D) Se for digitado o valor 0 (zero) para "numero", na execução do comando leia(numero), o resultado da variável binário será 0 e a segunda repetição (enquanto) não executará seu bloco de comandos, fará apenas o teste.

E) Se for digitado um valor menor que 0 (zero) para "numero", na execução do comando leia(numero), o programa solicitará que o usuário digite um número até que seja digitado um número maior ou igual a zero.

2) **Uma professora da 2ª série do ensino fundamental encomendou um programa que auxilie a gerar tabelas de tabuadas de multiplicação para seus alunos. O programa deve ler o número de base da tabuada e gerar a tabela de multiplicação para o intervalo [1,10], conforme o exemplo abaixo (considere que foi lido o número de base 5):**

Tabuada do 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Analise as alternativas a seguir, que apresentam algoritmos em pseudocódigo que pretendem resolver o problema da professora e selecione a mais correta em relação à solicitação e ao exemplo de exibição da tabuada.

A)

```
Algoritmo “tabuada”
        Var base, valor, resultado : inteiro
Inicio
        repita
                Escreval(“Digite a base da tabuada: ”)
                Leia(base)
        ate (base > = 1)
        Escreval(“Tabuada do ”,base)
        enquanto (valor < 10) faca
                resultado <- base * valor
                Escreval(base,”x ”,valor,” = “,resultado)
                valor <- valor + 1
        fimenquanto
FimAlgoritmo
```

B)

```
Algoritmo “tabuada”
        Var base, valor, resultado : inteiro
Inicio
        repita
                Escreval(“Digite a base da tabuada: ”)
                Leia(base)
        ate (base > = 1)
        Escreval(“Tabuada do ”,base)
        valor <- 1
        enquanto (valor < 10) faca
                resultado <- base * valor
                Escreval(base,”x ”,valor,” = “,resultado)
                valor <- valor + 1
        fimenquanto
FimAlgoritmo
```

C)

```
Algoritmo “tabuada”
        Var base, valor, resultado : inteiro
Inicio
        repita
                Escreval(“Digite a base da tabuada: ”)
                Leia(base)
        ate (base > = 1)
        Escreval(“Tabuada do ”,base)
        valor <- 1
        enquanto (valor < 10) faca
                resultado <- base + valor
                Escreval(base,”x ”,valor,” = “,resultado)
                valor <- valor + 1
        fimenquanto
FimAlgoritmo
```

D)

```
Algoritmo “tabuada”
        Var base, valor, resultado : inteiro
Inicio
        repita
                Escreval(“Digite a base da tabuada: ”)
                Leia(base)
        ate (base > = 1)
        Escreval(“Tabuada do ”,base)
        valor <- 1
        enquanto (valor < 10) faca
                resultado <- base + valor
                Escreval(valor,”x ”,base,” = “,resultado)
                valor <- valor + 1
        fimenquanto
FimAlgoritmo
```

E)

```
Algoritmo “tabuada”
        Var base, valor, resultado : inteiro
Inicio
        repita
                Escreval(“Digite a base da tabuada: ”)
                Leia(base)
        ate (base > = 1)
        Escreval(“Tabuada do ”,base)
        valor <- 1
        enquanto (valor <= 10) faca
                resultado <- base * valor
                Escreval(base,”x ”,valor,” = “,resultado)
                valor <- valor + 1
        fimenquanto
FimAlgoritmo
```

3) **As estruturas de repetição das linguagens de programação foram concebidas para que seja possível repetir determinados conjuntos de comandos. Essas estruturas são um importante recurso para o desenvolvimento de algoritmos/programas.**

Analise as alternativas a seguir relativas à estrutura de repetição "enquanto...faça" e assinale a opção correta.

A) O bloco de comandos da repetição é sempre executado pelo menos uma vez.

B) Não é necessário que a variável que será testada na condição receba um valor (seja inicializada) pois o comando "enquanto" faz isso automaticamente.

C) Se o resultado da condição for falso, o bloco de repetição é executado novamente; caso contrário, é finalizada a repetição.

D) Essa estrutura de repetição pode ser utilizada quando não se sabe exatamente quantas vezes o bloco de repetição deve ser repetido.

E) O bloco de repetição não pode ser executado infinitas vezes.

4) **Considere o seguinte algoritmo em pseudocódigo:**

```
Algoritmo “faz”
Var
 X, Y, Z : inteiro
Inicio
 repita
  Escreval(“Digite um numero: ”)
  Leia(X)
 ate (X > = 0)
 Y <- 1
 Z <- 1
 enquanto (Z <= X) faca
  Y <- Y * Z
  Z <- Z + 1
 fimenquanto
 Escreva(“Resultado: ”,Y)
FimAlgoritmo
```

Analise as alternativas a seguir, relativas a esse algoritmo, e assinale a verdadeira.

A) O algoritmo calcula a potência de um número.

B) O algoritmo calcula o fatorial de um número.

C) O algoritmo calcula o seno de um número.

D) O algoritmo calcula a raiz quadrada de um número.

E) O algoritmo calcula a média dos números.

5) **Um método simples para realizar o cálculo da raiz quadrada é encontrar a parte inteira, simplesmente subtraindo inteiros ímpares. Por exemplo, para calcular a parte inteira da raiz quadrada de 19, calcula-se a sequência:**

1. 19 – 1 = 18
2. 18 – 3 = 15
3. 15 – 5 = 10
4. 10 – 7 = 3

Como 3 é menor que 9, o processo termina aqui. Como quatro subtrações foram efetuadas, a resposta é 4, ou seja, a raiz quadrada de 19 é 4.

Analise as alternativas a seguir e selecione a que apresenta a implementação correta em pseudocódigo desse método de cálculo.

A)

```
Algoritmo "raizquadrada"
Var
 m,n,i : inteiro
Inicio
 repita
  Escreval("Digite um numero: ")
  Leia(m)
 ate (m > = 0)
 n <- 0
 i <- 1
 enquanto (m > = i) faca
  m <- m - i
  i <- i + 2
  n <- n + 1
  Escreval("m= ",m," i= ",i," n= ",n)
 fimenquanto
 Escreval("Parte inteira da raiz quadrada: ",n)
FimAlgoritmo
```

B)

```
Algoritmo "raizquadrada"
Var
 m,n,i : inteiro
Inicio
 repita
  Escreval("Digite um numero: ")
  Leia(m)
 ate (m > = 0)
 n <- 0
 i <- 1
 enquanto (m > = i) faca
  m <- m - i
  i <- i + 2
  n <- n + 1
 fimenquanto
```

```
 Escreval("Parte inteira da raiz quadrada: ",i)
FimAlgoritmo
```

C)

```
Algoritmo "raizquadrada"
Var
 m,n,i : inteiro
Inicio
 repita
  Escreval("Digite um numero: ")
  Leia(m)
 ate (m > = 0)
 n <- 0
 i <- 1
 enquanto (m > = i) faca
  m <- m - i
  i <- i + 1
  n <- n + 1
 fimenquanto
 Escreval("Parte inteira da raiz quadrada: ",n)
FimAlgoritmo
```

D)

```
Algoritmo "raizquadrada"
Var
 m,n,i : inteiro
Inicio
 repita
  Escreval("Digite um numero: ")
  Leia(m)
 ate (m > = 0)
 n <- 1
 i <- 1
 enquanto (m > = i) faca
  m <- m - i
  i <- i + 2
  n <- n + 1
 fimenquanto
 Escreval("Parte inteira da raiz quadrada: ",n)
FimAlgoritmo
```

E)

```
Algoritmo "raizquadrada"
Var
 m,n,i : inteiro
Inicio
 repita
  Escreval("Digite um numero: ")
  Leia(m)
 ate (m > = 0)
 n <- 0
 i <- 1
 enquanto (m <= i) faca
  m <- m - i
  i <- i + 2
  n <- n + 1
 fimenquanto
 Escreval("Parte inteira da raiz quadrada: ",n)
FimAlgoritmo
```

Na prática

A realização de processos repetitivos é mais comum do que podemos imaginar. Por exemplo, para nos deslocarmos, caminhando, de um ponto a outro, poderíamos expressar essa atividade da seguinte maneira: enquanto não chegarmos ao destino, dar um passo. Assim, o processo "dar um passo" será repetido enquanto a condição "não chegarmos ao destino" for verdadeira.

Se pensarmos na decoração de um salão para uma festa infantil, sempre são utilizados diversos balões. A tarefa de encher os balões poderia ser assim especificada: enquanto houver balões vazios, encher um balão. A atividade "encher um balão" será repetida diversas vezes, enquanto houver balões a encher.

Agora, vamos analisar um desafio da decoradora desse salão para a festa: ela tem quatro cores de balões (vermelho, amarelo, azul e verde) e deseja fazer conjuntos com dois balões cada (de cores diferentes). Quantos grupos diferentes de balões ela pode fazer?

Esse é um problema estudado pela matemática em análise combinatória, que possui um conjunto de procedimentos para construção de grupos com elementos finitos e seguindo determinadas regras. No caso, para auxiliar a decoradora, será necessário utilizar a combinação simples. Considere "m" a quantidade de elementos do conjunto e "p" o tamanho dos agrupamentos a serem construídos no exemplo m = 4 (4 cores) e p = 2 (agrupamentos de 2 balões cada). Para definir a quantidade de grupos diferentes que podem ser feitos, utiliza-se a seguinte expressão da combinação simples:

$C(m,p) = m!/[(m-p)!\ p!]$

O operador ! define o cálculo do fatorial de um número. O fatorial de um número n (n pertence ao conjunto dos números naturais – inteiros positivos) é o produto de todos os seus antecessores, incluindo a si próprio e excluindo o zero. A representação é feita pelo número fatorial seguido do sinal de exclamação, n! (lê-se fatorial de n ou n fatorial).

Exemplo de cálculo do fatorial de um número: $4! = 4 * 3 * 2 * 1 = 24$ (fatorial de 4)

Voltando ao desafio do agrupamento de balões, substituindo os valores de m e p na expressão da combinação simples, temos:

$C(4,2) = 4! / [(4-2]!\ 2!]$

C(4,2) = 4! / [2!2!]

Substituindo os fatoriais por seus cálculos:

C(4,2) = (4 * 3 * 2 * 1) / [(2 * 1) * (2 * 1)]

C(4,2) = 24 / 4 = 6

Portanto, é possível fazer seis agrupamentos distintos com dois balões cada, com as quatro cores de balões que temos.

{ (vermelho, amarelo), (vermelho, azul), (vermelho, verde), (amarelo, azul), (amarelo, verde), (azul, verde) }

Pode-se construir um algoritmo que realize esse processo de calcular a quantidade de combinações simples para diferentes valores de “m” e “p”. A seguir, temos a estrutura básica de passos em linguagem narrativa para resolver esse problema:

- Ler os valores de m (quantidade de elementos) e p (tamanho do agrupamento)
- Calcular o fatorial de m (F1)
- Calcular o fatorial de m-p (F2)
- Calcular o fatorial de p (F3)
- Calcular a combinação C = F1 / (F2 * F3)
- Mostrar o resultado da combinação C

Agora, para calcular o fatorial, precisamos realizar um processo de repetição que fará as diversas multiplicações consecutivas do número por cada um dos seus antecessores, conforme o seguinte conjunto de passos em linguagem narrativa:

- Inicializar fatorial = 1 (elemento neutro da multiplicação)
- Inicializar multiplicador = numero (número será o valor do qual se deseja calcular o fatorial)
- Enquanto o multiplicador for maior que zero, faça (essa é a repetição)

o fatorial = fatorial * multiplicador (faz as multiplicações consecutivas)

o multiplicador = multiplicador – 1 (pega o próximo antecessor do número)

- Mostrar ou usar o fatorial do número

O algoritmo a seguir, em pseudocódigo, lê um número, calcula e mostra o fatorial desse número.

```
Algoritmo "fatorial"

var

numero, Fatorial, multiplicador: inteiro

Inicio

Escreval("Digite um numero")

Leia(numero)

Fatorial <- 1

multiplicador<- numero

enquanto multiplicador > 0 faca

Fatorial <- fatorial * multiplicador

multiplicador<- multiplicador -1

fimenquanto

Escreval("Fatorial do numero: ", Fatorial)

FimAlgoritmo
```

Vamos fazer o teste de mesa desse algoritmo “fatorial”? Devemos lembrar que há várias formas de representar o teste de mesa; aqui, optaremos por representar através de uma tabela a evolução dos valores das variáveis. Considere que foi lido 5 para o número.

	Início	Enquanto (5 > 0)	Enquanto (4 > 0)	Enquanto (3 > 0)	Enquanto (2 > 0)	Enquanto (1 > 0)
Número	5					
Fatorial	1	5	20	60	120	120
Multiplicador	5	4	3	2	1	0

A repetição termina quando a condição (multiplicador > 0) for falsa, ou seja, veja pelo teste de mesa que o valor do multiplicador é decrementado a cada iteração; assim, quando o valor do multiplicador for zero, o teste multiplicador > 0 (0 > 0) será falso e, portanto, a repetição será finalizada. Assim, o resultado final da variável Fatorial é 120, indicando que o fatorial de 5! é 120, o que está correto. Faça o teste de mesa para outros valores para variável número, para compreender bem a lógica de funcionamento desse algoritmo.

Depois de compreendermos bem o processo para cálculo do fatorial de um número, podemos construir o algoritmo em pseudocódigo para calcular a quantidade de combinações simples. No código abaixo estão destacados os trechos que realizam o processo de cálculo de cada fatorial.

```
Algoritmo "combinacao"
var
m, p, F1, F2, F3, multiplicador: inteiro
c : real
Inicio
//ler os valores de m e p
Escreval("Digite a quantidade de elementos do conjunto: ")
Leia(m)
Escreval("Digite o tamanho do agrupamento: ")
Leia(p)
// calculo do fatorial de m!
 F1 <- 1
 multiplicador<- m
 enquanto multiplicador > 0 faca
 F1 <- f1 * multiplicador
 multiplicador<- multiplicador -1
 fimenquanto
 // calculo do fatorial de (m-p)!
 F2 <- 1
 multiplicador<- m-p
 enquanto multiplicador > 0 faca
 F2 <- f2 * multiplicador
multiplicador<- multiplicador -1
fimenquanto
//calculo do fatorial de p!
F3 <- 1
multiplicador<- p
enquanto multiplicador > 0 faca
F3 <- f3 * multiplicador
multiplicador<- multiplicador -1
fimenquanto
//calculo das combinações simples
c <- f1 (f2 * f3)
Escreval("Combinacoespossiveis: ", c)
FimAlgoritmo
```

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Lógica de Programação com VisualG Estrutura de Repetição - Enquanto.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Estruturas de Repetição 1 - Curso de Algoritmos.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.