

LÓGICA PARA REDES DE COMPUTADORES

CAPÍTULO 2 - COMO EXECUTAR REPETIDAMENTE UMA INSTRUÇÃO OU UM BLOCO DE INSTRUÇÕES?

Introdução

Quando estudamos programação, algumas questões comuns nos motivam inicialmente, por exemplo: você já se perguntou como os programadores fazem quando precisam ler dez nomes de cliente? Será necessário escrever dez linhas do comando “leia”? E se for um programa para inscrições de um concurso, quantas linhas de código são necessárias para ler os dados de cada candidato? Se, para ler os dados de um candidato, forem necessárias cinco linhas de código, então, para ler os dados de 100 candidatos, serão necessárias 500 linhas? Será que existe uma forma de não ficar repetindo as mesmas linhas de código? Para o bem dos programadores, existe sim!

As linguagens de programação possuem estruturas que possibilitam que linhas de código se repitam quantas vezes forem necessárias. Essas estruturas recebem o nome de estruturas de repetição, laços de repetição, comandos de repetição, ou *loops*.

Três pontos devem ser observados quando trabalhamos com repetições: quando ela começa, quantas vezes ela se repetirá e quando deverá parar. Esses três pontos são muito importantes, pois qualquer falha neles pode levar ao travamento do programa. Por exemplo, se em uma estrutura de repetição não for definida a sua parada, nós dissemos que o programa entrou em *looping* infinito e só sairá se forcarmos o desligamento do programa.

Neste capítulo, vamos compreender a estrutura de repetição e como utilizar os contadores e acumuladores. Na sequência, vamos entender as três estruturas básicas de repetição.

Acompanhe com atenção os estudos sobre comandos de repetição. Vamos começar! Bons estudos!

2.1 Estrutura de repetição

Sabemos que é muito comum e necessário o reaproveitamento de linhas de código no desenvolvimento de um programa. Observe a figura a seguir. Ela mostra um algoritmo para calcular a média aritmética de duas notas para, apenas, um aluno. São necessárias quatro linhas de código para a leitura, cálculo e exibição dos resultados, linhas 12 a 15.

```
1 INICIO
2 /* Algoritmo para calcular a média
3 aritmética de duas notas de um aluno.
4 -dados de entrada são: nome do aluno, duas notas
5 -dado de saída: média aritmética */
6
7     //DECLARAÇÃO DE VARIÁVEIS
8     real: media, nota1,nota2;
9     literal:nome;
10
11     //LINHAS DE COMANDO OU PROCESSAMENTO
12     escreva("Digite seu nome e suas duas notas");
13     leia(nome,nota1, nota2);
14     media=(nota1+nota2)/2;
15     escreva(nome," sua média é ",media);
16 FIM
```

Figura 1 - Algoritmo para calcular a média aritmética de duas notas de um aluno.

Fonte: Elaborada pela autora, 2018.

Pensando nesta mesma situação, a figura a seguir, mostra como é o algoritmo para calcular a média aritmética, agora, para dois alunos. Perceba que foi necessária a cópia das quatro linhas de código responsáveis pelo processamento do cálculo.

```
1 INICIO
2 /* Algoritmo para calcular a média
3 aritmética de duas notas de dois alunos.
4 -dados de entrada são: nome do aluno, duas notas
5 -dado de saída: média aritmética */
6
7     //DECLARAÇÃO DE VARIÁVEIS
8     real: media, nota1,nota2;
9     literal:nome;
10
11     //LINHAS DE COMANDO OU PROCESSAMENTO
12     escreva("Digite seu nome e suas duas notas");
13     leia(nome,nota1, nota2);
14     media=(nota1+nota2)/2;
15     escreva(nome," sua média é ",media);
16     escreva("Digite seu nome e suas duas notas");
17     leia(nome,nota1, nota2);
18     media=(nota1+nota2)/2;
19     escreva(nome," sua média é ",media);
20 FIM
```

Figura 2 - Algoritmo para calcular a média aritmética de duas notas de dois alunos.

Fonte: Elaborada pela autora, 2018.

Vamos fazer um cálculo rápido: quantas linhas terão este algoritmo para calcular a média aritmética das notas de 100 alunos?

Percebeu que é completamente inviável algoritmos construídos desta forma? É por isso que, para simplificar a nossa vida, existem as estruturas de repetição. Para o nosso exemplo de cálculo da média, bastaria fazer com que o programa, depois que executasse a linha 15, retornasse à linha 12 e problema resolvido.

VOCÊ O CONHECE?

Você já ouviu falar de Evgeniy Mikhailovich Bogachev? Também conhecido como Slavic ou lucky12345, é considerado pelo FBI, o *hacker* mais procurado do planeta. Há uma recompensa de U\$3 milhões, o equivalente a quase 12 milhões de reais, para quem der pistas que levem à sua prisão. Ele tem processos nos Estados Unidos por lavagem de dinheiro, fraude, pirataria informática e conspiração (AHRENS, 2017).

Agora que entendemos o básico sobre estrutura de repetição e para que é utilizada, vamos entender como ela funciona.

2.1.1 Introdução à estrutura de repetição

Uma estrutura de repetição possui uma condição ou teste. Se a condição retornar verdadeira/falsa, os comandos dentro da estrutura de repetição, serão executados quantas vezes o programador desejar.

A próxima figura mostra um fluxograma de uma estrutura de repetição genérica. Neste exemplo, o teste está no início, ou seja, nenhuma linha de código será executada dentro do comando de repetição, sem que o teste seja feito primeiro.



Figura 3 - Fluxograma de uma estrutura de repetição.

Fonte: Elaborada pela autora, 2018.

Das três estruturas que estudaremos neste capítulo, o que irá mudar é a posição do teste, se será no início ou no final, e quem irá controlar a parada do laço: o comando ou o desenvolvedor. Não se esqueça que laço, *loop*, comandos de repetição e estruturas de repetição são vários nomes para a mesma coisa.

Basicamente, três comandos de repetição são básicos e existentes em várias linguagens de programação: PARA, ENQUANTO e REPETIR ATÉ. Clique na interação a seguir para conhecê-los.

Algumas observações importantes:

- tudo que você conseguir fazer em um comando de repetição, conseguirá fazer a mesma coisa com os outros dois comandos;
- o usuário não sabe e nem precisa saber da existência de comandos. Na exibição para o usuário não fará diferença se o comando é o PARA ou ENQUANTO ou o REPETIR ATÉ;

O comando de repetição PARA possui a seguinte sintaxe:

```
para(inicializar_variaveis; condição; incremento)  
{  
    //linhas de código  
    //linhas de código  
}
```

A palavra chave PARA indica o nome do comando. Dentro dos parênteses têm-se três campos separados por ; (ponto-vírgula). O primeiro, *inicializar_variaveis* serve para atribuir um valor a uma ou mais variáveis. Caso utilize mais de uma variável, elas deverão estar separadas por vírgula.

O próximo campo é o da condição. Se esta condição retornar verdadeiro fará com que as linhas de código dentro das chaves ({ }) sejam executadas. Caso retorne falso, a primeira linha após a chave de fechamento do comando (}) seja executada, ou seja, ele sai do comando de repetição. Se for a primeira vez, ele nem entra no comando.

O último campo é o do incremento. Ele serve para adicionar valores à uma variável.

O comando ENQUANTO possui apenas um campo e seu teste é no início das linhas de código. A seguir, é mostrado a sintaxe deste comando.

```
enquanto(condição)  
{  
    //linhas de código  
    //linhas de código  
}
```

Este comando funciona assim: enquanto a condição for

verdadeira, repita as linhas de código que estiverem dentro das chaves ({}). Caso a condição retorne falsa, o algoritmo executará a primeira linha após o fechamento do comando, no caso, depois da chave ({}).

O próximo comando, REPETIR ATÉ, dos três é o único que faz o teste no final das linhas de código. O que representa que as linhas de código que estão dentro do comando, ou seja, entre as chaves, serão executadas pelo menos uma vez. Veja a sua sintaxe:

```
repetir
{
    //linhas de código
    //linhas de código
}ate(condição)
```

Observe que o comando REPETIR ATÉ, só possui um campo: condição e se encontra no final do comando. Todas as linhas de código que estão entre as chaves ({}) serão executadas até que a condição retorne falsa. Perceba que, quando a condição retornar verdadeira, o algoritmo retorna para a linha da chave ({}) logo após a palavra REPETIR.

- é possível utilizar comandos de repetição dentro de outro comando de repetição. Por exemplo, podemos colocar um comando PARA dentro de um comando ENQUANTO. Um exemplo clássico seria a de um relógio com horas e minutos. O comando interno marcará os minutos e o externo, as horas.

VOCÊ SABIA?

Precisamos ficar atentos aos pequenos detalhes ou no que mais parece obvio para evitar prejuízos financeiros ou até catástrofes. Um exemplo disto foi o desaparecimento de um satélite da NASA, em 1999, quando ele orbitava Marte. O prejuízo na época foi de \$125 milhões e o motivo, parece brincadeira, mas não é. Um erro na conversão de medidas, a NASA usava o sistema em polegadas (sistema anglo-saxão), enquanto uma empresa contratada utilizava medidas em metros, sistema decimal (BBC NEWS 2014). Leia mais:

```
<https://www.bbc.com/portuguese/noticias/2014/05/140530\_erros\_ciencia\_engenharia\_rb (https://www.bbc.com/portuguese/noticias/2014/05/140530_erros_ciencia_engenharia_rb)>.
```

Antes de estudarmos a primeira estrutura ou comando de repetição, será necessário o entendimento de dois conceitos importantíssimos e muito utilizados em programação: contadores e acumuladores.

2.1.2 Conceitos de contadores e acumuladores

Para entendermos como funciona um contador é necessário relembrarmos alguns conceitos. Uma variável é um espaço reservado na memória do computador. Então, quando escrevemos $x = 4$, leia-se x recebe o valor de 4, x representa uma posição de memória que receberá o valor de 4 (ASCENCIO; CAMPOS, 2002).

VOCÊ SABIA?

Um grupo de pesquisadores chineses conseguiu desenvolver uma memória de computador, chamada ReRAM, com cascas de ovos moídas. Para a pesquisa, o especialista em Física e Engenharia, Guangdong Zhou, da Southwest University in Chongqing, moeu cascas de ovos, transformando-as em um pó, que foi utilizado na construção da memória. Antes disso uma equipe já havia utilizado a clara do ovo na construção de uma memória chamada de Sinapse Artificial (INOVAÇÃO TECNOLÓGICA, 2017).

Vamos entender o que acontece se colocarmos $x = x + 1$, leia-se x recebe o valor de x mais 1, veja a figura a seguir.

```
1 INICIO
2 /*Algoritmo que utiliza um contador*/
3 //DECLARAÇÃO DE VARIÁVEIS
4 inteiro: x;
5 //LINHAS DE COMANDO OU PROCESSAMENTO
6 x = 1;//inicializar o valor de x
7 x = x + 1;
8 x = x + 1;
9 x = x + 1;
10 escreva("O valor de x é ", x);
11 FIM
```

Figura 4 - Algoritmo que utiliza um contador.
Fonte: Elaborada pela autora, 2018.

Na linha seis, o espaço de memória referente a x , recebe o valor de 1. Nesta linha ocorreu o que chamamos de inicializar a variável. Na sequência, linha 7, onde tem-se x

$= x + 1$, o processamento será feito como descrito a seguir. Clique para ver.



Primeiro, será processada a expressão que está à direita do sinal de atribuição ($=$). Ou seja, $x + 1$.

Isto é uma estrutura de um contador, ou seja,
 $\text{variável} = \text{variável} + \text{passo}$

Onde:

- o passo poderá ser qualquer valor. Por exemplo, se você quiser que ele conte de 5 em 5, basta colocar no passo o valor de 5, assim, poderia ser: $x = x + 5$, ou $y = y + 5$.

IMPORTANTE: como as variáveis indicam uma posição de memória e, ao serem criadas, elas podem ter qualquer valor, antes de usar um contador, você deve inicializar a variável do contador, no nosso exemplo, é a variável x .

VOCÊ QUER LER?

É muito importante o entendimento sobre uma variável e a sua relação com a memória. Para entender melhor esses conceitos, leia capítulo 2.4 “Conceitos de Memória do livro “C ++ Como Programar” (DEITEL; DEITEL, 2011). Busque na biblioteca virtual: <<https://Animabrasil.blackboard.com/webapps/ga-bibliotecaSSO-BBLEARN/homePearson> (<https://Animabrasil.blackboard.com/webapps/ga-bibliotecaSSO-BBLEARN/homePearson>)>.

Agora que entendemos o que é um contador, ficará mais fácil entendermos o que é um acumulador. A diferença entre eles, é que o passo no contador é determinado pelo programador e no acumulador, é determinado pelo usuário. Vamos ver um exemplo de um algoritmo para somar três números digitados pelo usuário, na figura a seguir.

```
1 INICIO
2 /*Algoritmo para somar três
3 números digitados pelo usuário*/
4     //DECLARAÇÃO DE VARIÁVEIS
5     inteiro: soma, num;
6     //LINHAS DE COMANDO OU PROCESSAMENTO
7     soma = 0; //inicializar o valor da soma
8     escreva("Digite um número");
9     leia(num);
10    soma = soma + num;
11    escreva("Digite um número");
12    leia(num);
13    soma = soma + num;
14    escreva("Digite um número");
15    leia(num);
16    soma = soma + num;
17    escreva("A soma dos 3 números é ", soma);
18 FIM
```

Figura 5 - Algoritmo para somar três números digitados pelo usuário.

Fonte: Elaborada pela autora, 2018.

Na linha 7, a variável *soma* foi inicializada com o valor zero por ser um elemento neutro da soma. Na linha 10, temos o acumulador ou neste caso, um somador. No lado esquerdo, a equação $soma + num$, irá retornar o valor de *num*, ou seja, o primeiro número que o usuário digitou e será armazenado na variável *soma*.

Na linha 13, o acumulador se repete. No lado esquerdo, temos novamente a equação $soma + num$, só que, agora, a variável *soma* tem armazenado o valor do primeiro número digitado pelo usuário e a variável *num* tem o segundo número digitado pelo usuário. A soma destes dois valores será armazenada na variável *soma*, que será adicionada ao terceiro número digitado pelo usuário e armazenado na variável *soma*, na linha 16. Portanto, será exibida a soma dos três números que o usuário digitou.

Você pode estar se perguntando, se para o nosso exemplo da figura anterior, não poderíamos utilizar três variáveis para armazenar os três números do usuário e depois fazer a soma dos três, como no exemplo apresentado a seguir. Clique para ver.

Sim, poderia ser feito, mas não esqueça que estamos nos preparando para usar comandos de repetição e os conceitos de contadores e acumuladores são fundamentais. Vamos ver nosso primeiro comando de repetição a seguir. Acompanhe!



escreva("Digite 3 números");

2.2 Estrutura de repetição com teste no início

Dos três comandos básicos de repetição que iremos estudar, dois fazem o teste no início da estrutura, os comandos PARA e ENQUANTO. Com o teste no início, o programador consegue controlar quando as linhas de código, que estão dentro do comando, serão executadas. O que não ocorre com o comando REPETIR ATÉ, pois seu teste será no final, ocasionando a execução de, pelo menos, uma vez as linhas de código.

O primeiro comando a ser estudado é o PARA. Vamos conhecer este comando.

2.2.1 Introdução à estrutura de repetição com teste no início

A figura a seguir mostra a estrutura ou sintaxe do comando de repetição PARA.

1	PARA (<i>inicializar variável ; condição ; incremento</i>)
2	{
3	<i>linhas de comando;</i>
4	<i>linhas de comando;</i>
5	
6	}

Figura 6 - Sintaxe do comando de repetição PARA.

Fonte: Elaborada pela autora, 2018.

O quadro a seguir explica o significado de cada campo do comando PARA.

Campo	Significado
Inicializar variável	<ul style="list-style-type: none"> • Espaço reservado para atribuir valor inicial à variável que será usado na condição. • É o primeiro campo a ser executado pelo programa. • Pode-se inicializar mais de uma variável.
Condição	<ul style="list-style-type: none"> • Teste de parada do comando. • É o segundo campo executado pelo programa. • Tem que ser uma equação que utiliza somente operadores relacionais (>, <, >=, <=, !=, <=>), ou seja, só pode retornar falso ou verdadeiro. • Se retornar verdadeiro, o programa executa as linhas de comando que estão dentro dos {}, ou seja, o que estiver entre as linhas dois a seis da figura anterior. • Caso retorne falso, o programa irá executar o comando que estiver na linha 7.
Incremento	<ul style="list-style-type: none"> • Neste campo coloca-se um contador, ou seja, no formato variável=variável+ passo. • Atenção!!!! Ele só será executado a partir da segunda iteração, ou seja, segunda execução.

Quadro 1 - Campos do comando PARA.

Fonte: Elaborado pela autora, 2018.

Para entender o funcionamento do comando PARA, vamos analisar o algoritmo para listar os números de 1 a 10 para o usuário. Observe a figura a seguir.

```
1 INICIO
2 /*Algoritmo para imprimir
3 os números de 1 a 10*/
4     //DECLARAÇÃO DE VARIÁVEIS
5     inteiro: i;
6     //LINHAS DE COMANDO OU PROCESSAMENTO
7     para(i=1; i<=10; i=i+1)
8     {
9         escreva(i, " - ");
10    }
11 FIM
```

Figura 7 - Algoritmo para imprimir para o usuário os números de 1 a 10.

Fonte: Elaborada pela autora, 2018.

No comando PARA, linha 7, a primeira coisa que acontece é a variável **i** receber o valor de 1. Depois, é feito o teste **i <= 10**, como **i** tem o valor de 1, então, o teste será **1 <= 10**, o que retornará verdadeiro. Ao retornar verdadeiro, serão executados todos os comandos entre as {}. No nosso caso, apenas o comando escreva, que imprimirá ao usuário "1 - ". Ao chegar na }, linha 10, o programa sabe que deve retornar para a linha 7, onde ele executa a parte do incremento, **i = i + 1**. Como **i** tem o valor de 1, ao somar 1 ele irá pra 2. Depois, executa o teste **i <= 10**, como **i** tem o valor de 2, então, o teste será **2 <= 10**, o que retornará verdadeiro. Como retornou verdadeiro, ele excuta o comando escreva. A saída será "1 - 2 - ". Este processo se repetirá até **i** receber o valor de 10, a saída será "1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - ". Ao chegar na linha 10, ele retorna para a linha 7, onde incrementa a variável **i** que terá seu valo atualizado para 11, executa o teste **i <= 10**, como **i** está valendo 11, o teste retornará falso. Como retornou falso, ele executa a linha 11, que no nosso exemplo é o fim do algoritmo.

O fluxograma deste algoritmo é ilustrado pela figura a seguir. Observe que ele executa apenas uma vez a inicialização da variável e logo em seguida faz o teste. Se o teste retornar verdadeiro, o algoritmo escreve a mensagem e incrementa a variável **i**, retornando para o teste. Se o teste retornar falso, ele finalizará.

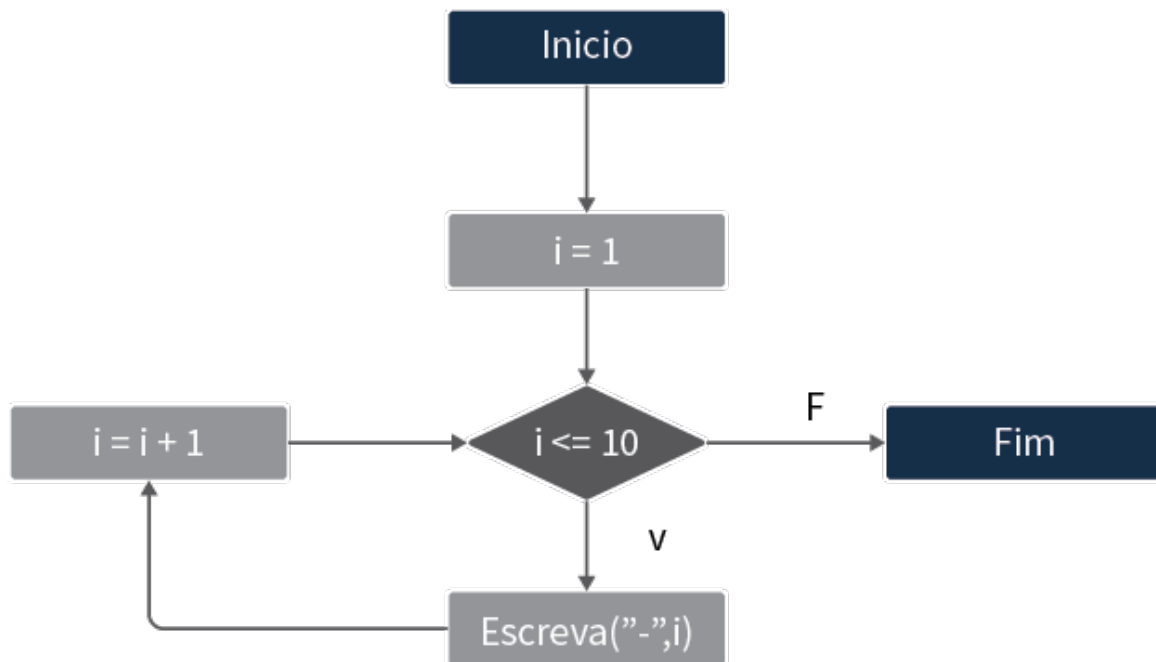


Figura 8 - Fluxograma de um algoritmo para imprimir os números de 1 a 10 para o usuário.

Fonte: Elaborada pela autora, 2018.

Vamos ver um exemplo um pouco mais complicado. A figura a seguir, mostra um algoritmo para ler cinco números digitados pelo usuário e imprimir a soma e a média destes números.

```
1 INICIO
2 /*Algoritmo para imprimir a soma e a média dos 5 números
3 digitados pelo usuário*/
4 //DECLARAÇÃO DE VARIÁVEIS
5 inteiro: x, soma, num;
6 real: media;
7 //LINHAS DE COMANDO OU PROCESSAMENTO
8 soma=0;
9 para(x = 1; x <= 5; x = x + 1)
10 {
11     escreva("Digite o ", x,"° número");
12     leia(num);
13     soma=soma+num;
14 }
15 media=soma/5;
16 escreva("A soma dos números é ", soma);
17 escreva("A média aritmética destes números é ", media);
18 FIM
```

Figura 9 - Algoritmo para imprimir a soma e a média de 5 números digitados pelo usuário.

Fonte: Elaborada pela autora, 2018.

Neste algoritmo da figura acima, temos um acumulador (***soma = soma + num***) dentro do comando de repetição e sua variável foi inicializada (***soma = 0***) fora do comando, linha 8. Se ela fosse inicializada dentro do comando, ou seja, entre as linhas 10 a 14, o que iria acontecer? Todas as cinco vezes que o programa passasse pela linha de código ***soma = 0***, iria zerar a variável soma e ela não guardaria a soma de todos os números.

IMPORTANTE!

- As chaves ({ }) indicam onde começa e termina as linhas de código que se repetirão.
- Este recuo dado nas linhas dentro da estrutura é chamado de indentação. A indentação não interfere na execução do programa. Se você digitar tudo em uma única linha, seu programa irá rodar, se estiver correto. O grande problema será procurar por um erro. Com a indentação, fica visualmente mais fácil de entender o programa.
- As boas práticas de programação recomendam: indentação, comentários e documentação dos programas.

VOCÊ QUER LER?

No artigo “Boas Práticas de Programação”, o professor Neri Aldoir Neitzke (2011) descreve seis boas práticas que devem ser utilizadas na programação. A primeira e a segunda devem ser utilizadas, independente da linguagem escolhida. Leia o artigo completo em: <https://www.devmedia.com.br/boas-praticas-de-programacao/21137> (<https://www.devmedia.com.br/boas-praticas-de-programacao/21137>)>.

Antes de prosseguir nos estudos, é altamente recomendado que você pratique um pouco. Forbellone (2005) inicia os estudos do comando PARA e explica o desenvolvimento de vários algoritmos.

O próximo comando de repetição é o ENQUANTO, que é também uma estrutura de repetição com o teste no início. Prepare-se para mais um comando!

2.3 Estrutura de repetição com teste no início

Outro comando de repetição com teste no início é o ENQUANTO. A estrutura e funcionamento são simples. Para entrar no laço de repetição, ele fará o teste, se retornar verdadeiro, ele entra no laço e se retornar falso ele não entra dentro da estrutura de repetição. Só temos que ficar atentos à condição de parada, porque enquanto o teste retornar verdadeiro o programa não sairá do *loop*. Vamos entender melhor a seguir.

2.3.1 Estrutura de repetição com teste no início usando flag de parada

A figura a seguir mostra a sintaxe do comando ENQUANTO. Veja que ele só possui o campo da condição.

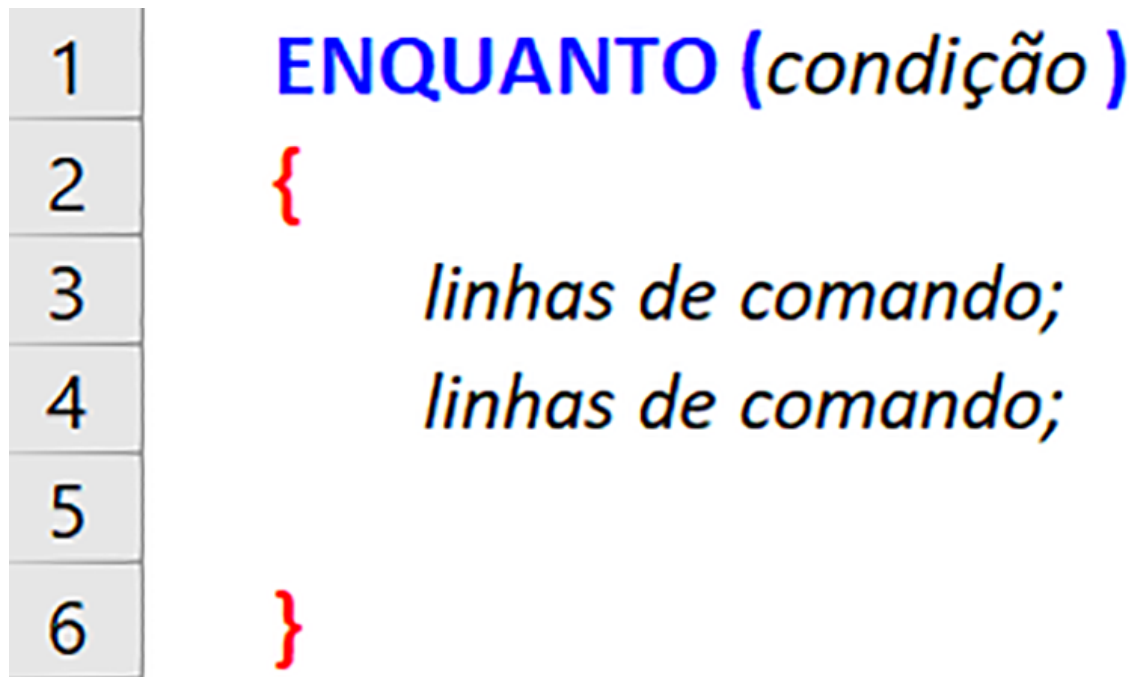


Figura 10 - Estrutura do Comando de Repetição ENQUANTO.

Fonte: Elaborada pela autora, 2018.

Seu funcionamento é simples, enquanto a condição retornar verdadeiro, ele executa as linhas de código que estão entre as { }, que no nosso exemplo, figura anterior, é o que está escrito entre as linhas de 2 a 6. Quando a condição retornar falsa, ou ele nem entra, se for o primeiro teste, ou ele sai do laço.

Um aspecto muito importante é o controle sobre a condição. Em algum momento, você deverá tornar a condição falsa. Se isso não ocorrer, o programa entrará num *looping* infinito, ou seja, ele nunca sairá de dentro do comando. Veja o exemplo a seguir, figura a seguir.

```
1 INICIO
2 /*Algoritmo em looping infinito*/
3 //DECLARAÇÃO DE VARIÁVEIS
4 inteiro:m;
5 //LINHAS DE COMANDO OU PROCESSAMENTO
6 m = 0;
7 enquanto (m == 0)
8 {
9     escreva("Estou preso aqui");
10 }
11
12 FIM
```

Figura 11 - Algoritmo em looping infinito.

Fonte: Elaborada pela autora, 2018.

Na linha 6, a variável ***m*** foi inicializada com zero, o que tornou a condição (***m* == 0**) verdadeira, linha 7. Assim, o programa entrou na estrutura de repetição. Mas, o valor de ***m*** não é alterado dentro da estrutura, linha 8 a 10. Para sair da estrutura, a condição deverá retornar falso, como isso não acontece, o programa fica preso dentro da estrutura para sempre. Aqui, este “sempre” significa até o usuário interferir e abortar a execução, como, por exemplo dando um *boot* na máquina.

Vamos inserir uma parada neste algoritmo? Por exemplo, eu quero que ele imprima apenas três vezes e depois saia do *loop*. Veja o exemplo a seguir.

```
1 INICIO
2 /*Algoritmo para imprimir 3 vezes
3 "Agora vou sair" para o usuário*/
4 //DECLARAÇÃO DE VARIÁVEIS
5 inteiro:m;
6 //LINHAS DE COMANDO OU PROCESSAMENTO
7 m = 1;
8 enquanto (m <= 3)
9 {
10     escreva("Agora vou sair!");
11     m = m + 1;
12 }
13 FIM
```

Figura 12 - Algoritmo com controle de parada.

Fonte: Elaborada pela autora, 2018.

Agora, o algoritmo ficou com a inicialização da variável **m**, linha 7 da figura anterior, condição, linha 8 e incremento, linha 11. Ficou parecido com o comando PARA, né? Se ficou parecido, então, porque a existência de dois comandos parecidos? O que acontece é que cada comando tem uma aplicação mais típica. Quando terminarmos os três comandos, faremos uma comparação mais detalhada. Por agora, vamos aprender o funcionamento de cada comando.

VOCÊ QUER VER?

O filme “Feitiço do tempo” (RAMIS; RUBIN, 1993), lançado no Brasil em 1993, tem como protagonista um meteorologista que chama Phil, interpretado por Bill Murray. Phil fica preso em um lapso de tempo, ele acorda sempre no mesmo dia. Não importa o que ele faça, a cada amanhecer o dia será o mesmo, com as mesmas pessoas e os mesmos acontecimentos. Isto é um bom exemplo de um *looping*. Assista ao filme e tente descobrir qual é o teste de parada desta estrutura de repetição.

Não precisamos utilizar este comando sempre com um contador como variável de saída do *loop*. Podemos utilizar o usuário como elemento de decisão. Considere a seguinte situação: criar um algoritmo que informe ao usuário se o número, que ele digitou, é par ou ímpar. Para sair da brincadeira, basta o usuário digitar o número zero. O algoritmo também informará quantos números o usuário digitou antes do zero.

O fluxograma, o algoritmo e a tabela com as explicações de cada linha de comando é ilustrado pelas figuras a seguir.

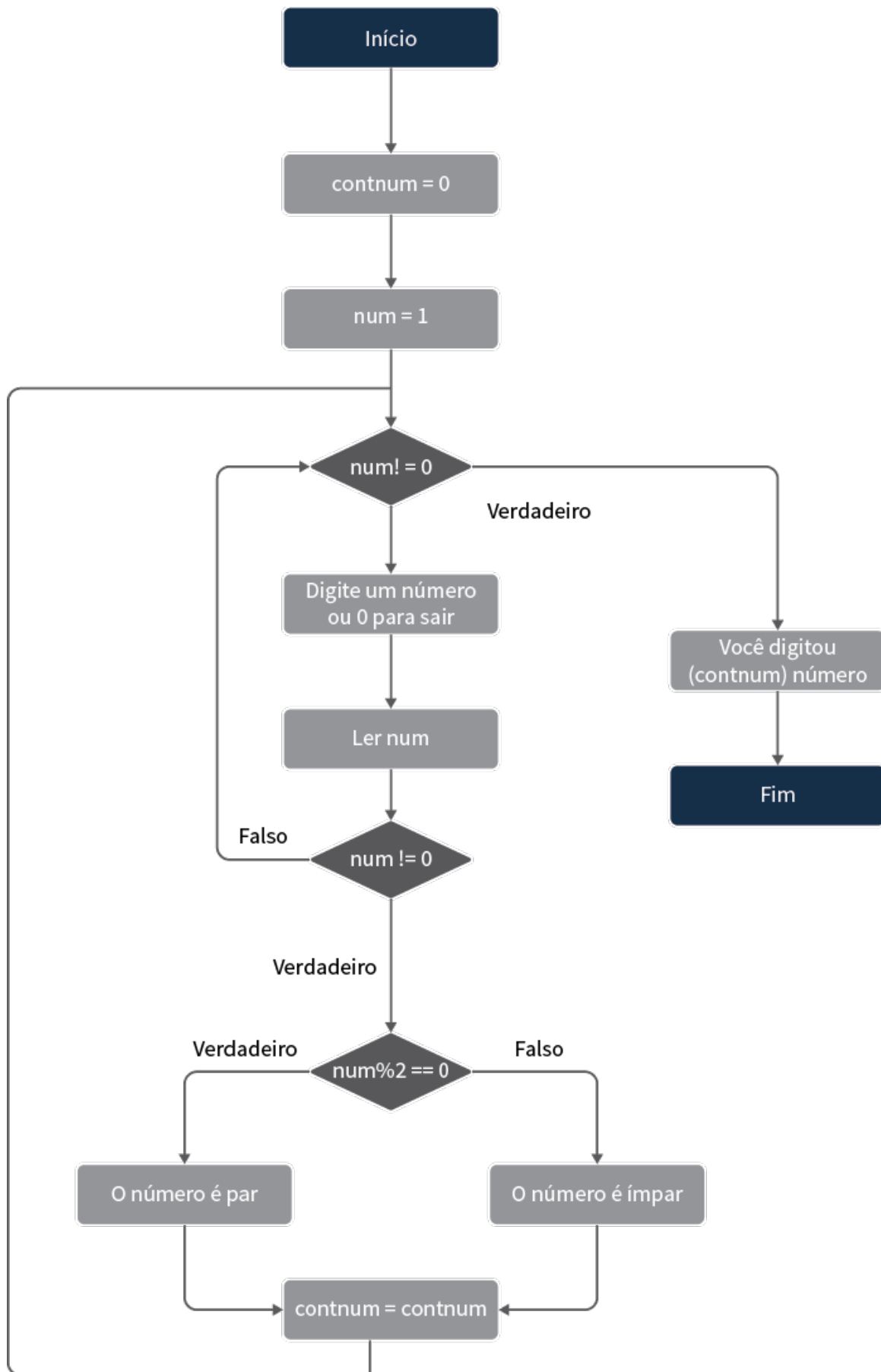


Figura 13 - Fluxograma para informar se os números são pares ou ímpares.

Fonte: Elaborada pela autora, 2018.

Compare este fluxograma da figura anterior, com o algoritmo correspondente ilustrado pela figura abaixo.

```
1 INICIO
2 /*Algoritmo para informar se os números digitados pelos usuário
3 são pares ou ímpares. Para sair do algoritmo, o usuário digitará zero*/
4 //DECLARAÇÃO DE VARIÁVEIS
5 inteiro:contnum, num;
6 //LINHAS DE COMANDO OU PROCESSAMENTO
7 contnum = 0;
8 num = 1;
9 enquanto(num != 0)
10 {
11     escreva("Digite um número ou digite 0 (zero) para sair");
12     ler(num);
13     Se (num!=0)
14         então
15             {
16                 se ((num%2)==0)
17                     então
18                         escreva("O número ", num, " é par");
19                     senão
20                         escreva("O número ", num, " é ímpar");
21                 FimSe
22                 contnum = contnum + 1;
23             }
24         FimSe
25     }
26     escreva("Você digitou ", contnum, " números");
27 FIM
```

Figura 14 - Algoritmo para informar se os números são pares ou ímpares.

Fonte: Elaborada pela autora, 2018.

O quadro a seguir, explica o funcionamento deste algoritmo, figura anterior, linha por linha. Para cada linha deste quadro, tente localizar no algoritmo e no fluxograma, as linhas equivalentes. Por exemplo, a linha 1 que representa o início do algoritmo equivale ao símbolo de início no fluxograma. Só prossiga se ficar muito bem claro pra você o funcionamento deste algoritmo.

Linha	Ação
1	Início do algoritmo.
2	Comentários de parágrafo (/ * /).
3	Comentários de linha (/ /).
4	Declaração das variáveis <i>contnum</i> (para contar os números digitados) e <i>num</i> (para guardar os números que o usuário digitar).
5	Comentários de linha (/ /).
6	Inicialização da variável <i>contnum</i> com o valor 0 (zero). Foi escolhido o valor de zero por ser um elemento neutro da soma, ou seja, este valor não interferirá no resultado final da quantidade de números digitados.
7	Inicialização da variável <i>num</i> com o valor 1 (um). Aqui poderia ser escolhido qualquer valor diferente de zero, pois o objetivo é tornar verdadeira o teste (<i>num!=0</i>), leia-se num diferente de zero, na linha 9. Assim, o algoritmo entrará na primeira execução dentro da estrutura de repetição, que começa na linha 10 e termina na linha 25.
8	Comando de repetição ENQUANTO com a condição (<i>num!=0</i>). Neste momento, <i>num</i> possui o valor de 1, então, a condição (<i>1!=0</i>) retornará verdadeiro. O algoritmo entra no loop e executa a linha 10. Caso a condição (<i>num!=0</i>) retorne falso, o algoritmo pularia para a linha 26 e imprimiria para o usuário a mensagem “Você digitou 0 números”.
9	Indica o início do comando ENQUANTO. Todos os comandos entre as linhas de 10 a 25 estão dentro da estrutura de repetição.
10	Manda mensagem para o usuário e pula automaticamente para a linha 12, pois o comando de saída de dados, escreva, não provoca pausa na execução.
11	Aguarde até que o usuário digite um número qualquer e depois digite a tecla <i>enter</i> . O número digitado será armazenado na variável <i>num</i> .
12	No comando condicional, tem-se o teste <i>num!=0</i> . Se o usuário digitar o número 0 (zero), a condição <i>0!=0</i> será falsa, o que significa que o usuário quer sair do sistema. O algoritmo pulará para a linha 24. Se o usuário digitar qualquer valor diferente de 0 (zero), a condição <i>num!=0</i> retornará verdadeira significando que o usuário deseja saber se o número é par ou ímpar. Seguirá para a linha 14.
13	A palavra reservada “então” significa que começará a execução de códigos quando a condição do comando SE retornou verdadeiro. Segue para a linha 15.
14	Aqui cabe uma observação: foi adicionado um par de chaves, abrindo na linha 15 e fechando na linha 23. Esta adição serve para indicar onde começam e terminam as linhas de código do caminho do “então”. Quando for uma única linha de comando, as chaves podem ser omitidas. Observe as linhas 18 e 20. Segue para a linha 16.
15	Comando condicional com o teste ((<i>num%2</i>)==0). O operador % pega o resto de uma divisão de inteiro. Por exemplo, se você fizer a divisão de inteiros de 11 por 5, o resultado será 2 e o resto será 1. O operador % seleciona o resto, no caso do exemplo, o número 1. Todo número par é divisível por 2, o que significa que se o número for par, o resto da divisão por 2 será zero. É isto que este teste faz. Ele verifica se o resto da divisão do número digitado pelo usuário é zero. Se for, ou seja, se retornar verdadeiro, o número é par. Segue para a linha 17. Se o teste retornar falso, então o número é ímpar, segue para a linha 19.
16	A palavra reservada “então” significa que começará a execução de códigos, quando a condição do comando SE retornou verdadeiro. Vai para a linha 18.
17	Imprime a mensagem para o usuário que o número é par. Segue para a linha 21.
18	A palavra reservada “senão” significa que começará a execução de códigos quando a condição do comando SE retornou falso. Vai para a linha 20.
19	Imprime a mensagem para o usuário que o número é ímpar. Segue para a linha 21.

20	Indica o final do comando condicional SE que começou na linha 16. Vai para a linha 22.
21	Execução de <i>contnum=contnum+1</i> . Incrementa de 1 a variável <i>contnum</i> . O que representa que o usuário digitou um número diferente de zero. Segue para a linha 23.
22	Fechamento do bloco de comandos referente ao “então” do comando SE iniciado na linha 13. Ou seja, se o teste do comando SE, linha 13, retornar verdadeiro, os comandos entre as linhas 15 a 23 serão executados. Próxima linha será a 24.
23	Finalização do comando condicional SE, linha 13. Segue para a linha 25.
24	Fechamento do bloco de comando que se repetirão, caso a condição no comando ENQUANTO, linha 9, retornar verdadeiro. Volta para a linha 9.
25	Só parará por esta linha se a condição do comando ENQUANTO, linha 9, retornar falso. Ou seja, quando o usuário digitar o número zero.

Quadro 2 - Explicação do algoritmo da figura anterior.

Fonte: Elaborado pela autora, 2018.

Agora, considere que além de informar se o número é par ou ímpar e a quantidade de números digitados, o algoritmo deverá informar quantos números pares e quantos ímpares o usuário digitou. Bem, para isto, basta inserir uma linha com um contador de números pares, ***contpar = contpar + 1***, na linha 18. Não podemos esquecer de inicializar a variável ***contpar*** fora do comando de repetição, antes da linha 9. Faça o mesmo para contar os números ímpares. Analise o algoritmo da figura a seguir.

```

1  INICIO
2  /*Algoritmo para informar se os números digitados pelos usuário
3  são pares ou ímpares. Para sair do algoritmo, o usuário digitará zero*/
4  //DECLARAÇÃO DE VARIÁVEIS
5  inteiro:contnum, num, contpar, contimpar;
6  //LINHAS DE COMANDO OU PROCESSAMENTO
7  contnum = 0;
8  contimpar = 0;
9  contpar = 0;
10 num = 1;
11 enquanto(num != 0)
12 {
13     escreva("Digite um número ou digite 0 (zero) para sair");
14     ler(num);
15     Se (num!=0)
16     então
17     {
18         se((num%2)==0)
19         então
20         {
21             escreva("O número ",num, " é par");
22             contpar = contpar + 1;
23         }
24         senão
25         {
26             escreva("O número ",num, " é ímpar");
27             contimpar = contimpar + 1;
28         }
29         FimSe
30         contnum = contnum + 1;
31     }
32     FimSe
33 }
34 escreva("Você digitou ", contnum, " números");
35 escreva("Sendo ", contpar, " pares e ", contimpar, " ímpares");
36 FIM

```

Figura 15 - Algoritmo para informar e contar a quantidade de números pares e ímpares.

Fonte: Elaborada pela autora, 2018.

Observe o que mudou. Inicializar as variáveis que contarão a quantidade de números ímpares e pares, linhas 8 e 9, figura anterior. Quando o algoritmo passar pela parte do então, linha 19, ou do senão na linha 24, tem-se duas linhas de comando para cada opção. Por este motivo, foi inserido as chaves, linhas 20 e 23 e linhas 25 e 28.

A próxima figura mostra programa equivalente ao algoritmo anterior, na linguagem de programação C++.

Clique na interação a seguir e analise as diferenças.

O comando de saída de dados COUT faz o mesmo papel que o ESCRIVA e o CIN, o mesmo que o LEIA.

São poucas, né? O comando ENQUANTO é o WHILE em linguagem C++.

O comando condicional SE é o comando IF, aqui temos uma pequena diferença: em linguagem C, não se utiliza a palavra equivalente ao ENTÃO, que seria o THEN.

Somente a equivalente ao SENÃO, que é ELSE. Já percebeu que a maioria dos comandos é a tradução para o inglês?

figura15.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  int main ()
5  {
6      /*Programa para informar e contar a quantidade de números pares e ímpares .
7      Para sair do algoritmo, o usuário digitará zero*/
8      //DECLARAÇÃO DE VARIÁVEIS
9      int contnum, num, contpar, contimpar;
10     //LINHAS DE COMANDO OU PROCESSAMENTO
11     contnum = 0;
12     contimpar = 0;
13     contpar = 0;
14     num = 1;
15     while(num != 0)
16     {
17         cout<< "Digite um numero ou digite 0 (zero) para sair \n";
18         cin>>num;
19         if (num!=0)
20         {
21             if((num%2)==0)
22             {
23                 cout<< "O numero "<<num<< " e par \n";
24                 contpar = contpar + 1;
25             }
26             else
27             {
28                 cout<< "O numero "<<num<< " e impar\n";
29                 contimpar = contimpar + 1;
30             }
31         }
32     }
```

```
31 |         contnum = contnum + 1;  
32 |     }  
33 | }  
34 | cout<< "Voce digitou "<< contnum<< " numeros\n";  
35 | cout<< "Sendo "<< contpar<< " pares e "<< contimpar<< " impares";  
36 | }
```

Figura 16 - Programa para informar e contar a quantidade de números pares e ímpares.

Fonte: Elaborada pela autora, 2018.

Vamos executar este programa?

Ao executar o programa, a primeira tela que o usuário vê é exibida pela próxima figura. Observe que o cursor pulou para a linha abaixo e o programa fica parado esperando o usuário digitar um número. Isso acontece porque foi inserido o caractere “\n” no comando COUT na linha 17 da figura anterior.

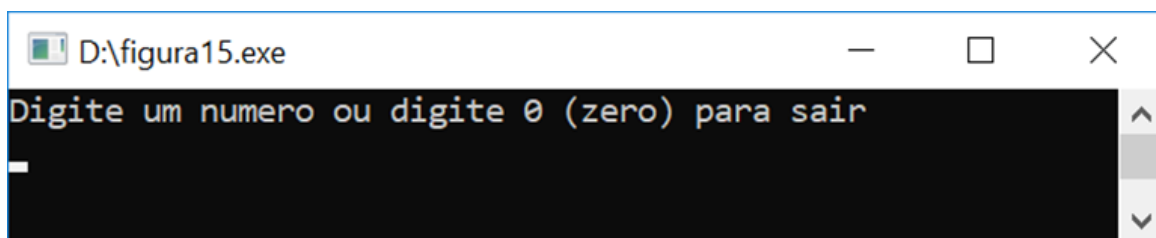
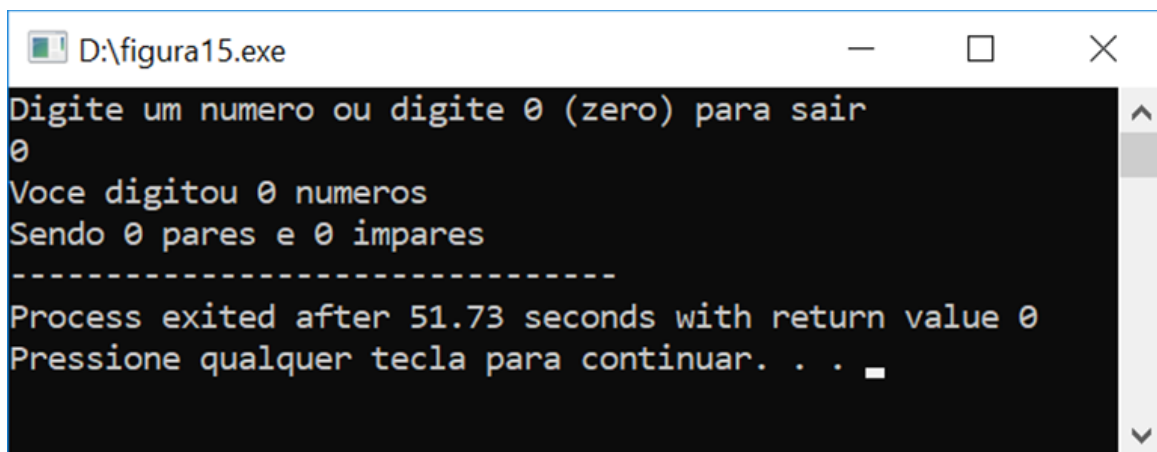


Figura 17 - Tela de execução do programa.

Fonte: Elaborada pela autora, 2018.

A figura a seguir é um *print* da tela de execução para uma situação onde o usuário entrou no sistema e digitou, primeiramente, o número zero. Observe que o programa mandou a mensagem para o usuário, informando que ele não digitou números.

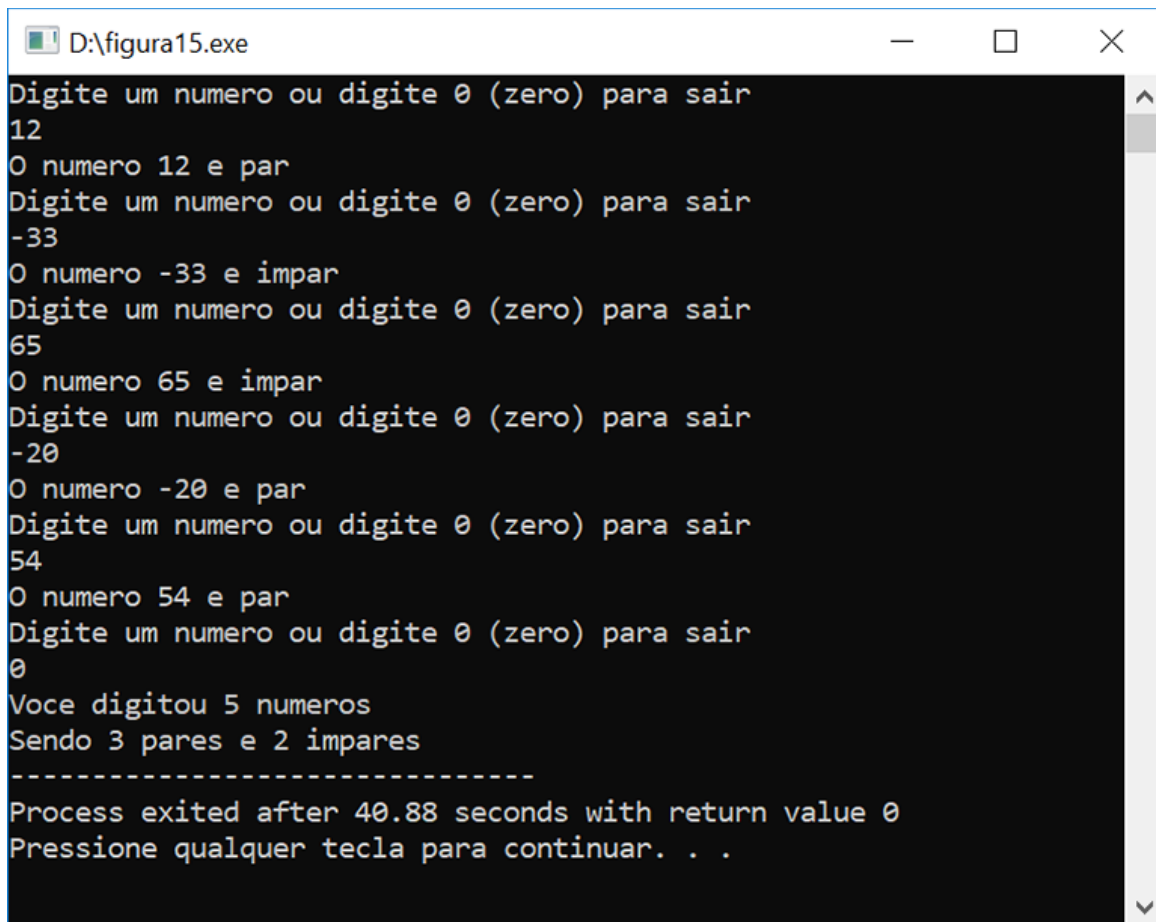


```
D:\figura15.exe
Digite um numero ou digite 0 (zero) para sair
0
Voce digitou 0 numeros
Sendo 0 pares e 0 impares
-----
Process exited after 51.73 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 18 - Print da tela da execução do programa para informar e contar a quantidade de números pares e ímpares.

Fonte: Elaborada pela autora, 2018.

A próxima figura mostra a execução do programa com o usuário digitando seis números. Para cada número digitado, o programa informa se ele é par ou ímpar. Acompanhe a execução do programa comparando com o algoritmo *para informar e contar a quantidade de números pares e ímpares*.



```
D:\figura15.exe
Digite um numero ou digite 0 (zero) para sair
12
O numero 12 e par
Digite um numero ou digite 0 (zero) para sair
-33
O numero -33 e impar
Digite um numero ou digite 0 (zero) para sair
65
O numero 65 e impar
Digite um numero ou digite 0 (zero) para sair
-20
O numero -20 e par
Digite um numero ou digite 0 (zero) para sair
54
O numero 54 e par
Digite um numero ou digite 0 (zero) para sair
0
Voce digitou 5 numeros
Sendo 3 pares e 2 impares
-----
Process exited after 40.88 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 19 - Execução do Programa equivalente ao algoritmo para informar e contar a quantidade de números pares e ímpares.

Fonte: Elaborada pela autora, 2018.

Compreendemos o funcionamento do comando de repetição ENQUANTO. Possui um campo, no final do comando que funciona como um teste. Agora, o nosso próximo comando é o REPETIR ATÉ.

2.4 Estrutura de repetição com teste no final e variável de controle

O terceiro comando que iremos estudar é o comando REPETIR ATÉ. Ele possui duas características que o difere dos demais. A primeira é a localização da condição e se encontra no final do comando. E a segunda é que, pelo fato de o algoritmo não fazer teste no início, significa que ele não fará teste para entrar na estrutura de repetição. É garantido que as linhas de código dentro do comando serão executadas pelo menos uma vez. Neste comando, se a condição retornar falsa, o algoritmo sairá do comando.

Não se preocupe se estas duas características não ficaram ainda claras pra você. Vamos estudar o funcionamento e algumas aplicações deste comando a seguir.

2.4.1 Estrutura de repetição com teste no final e variável de controle

A sintaxe da estrutura de repetição REPETIR ATÉ é exibida pela próxima figura.

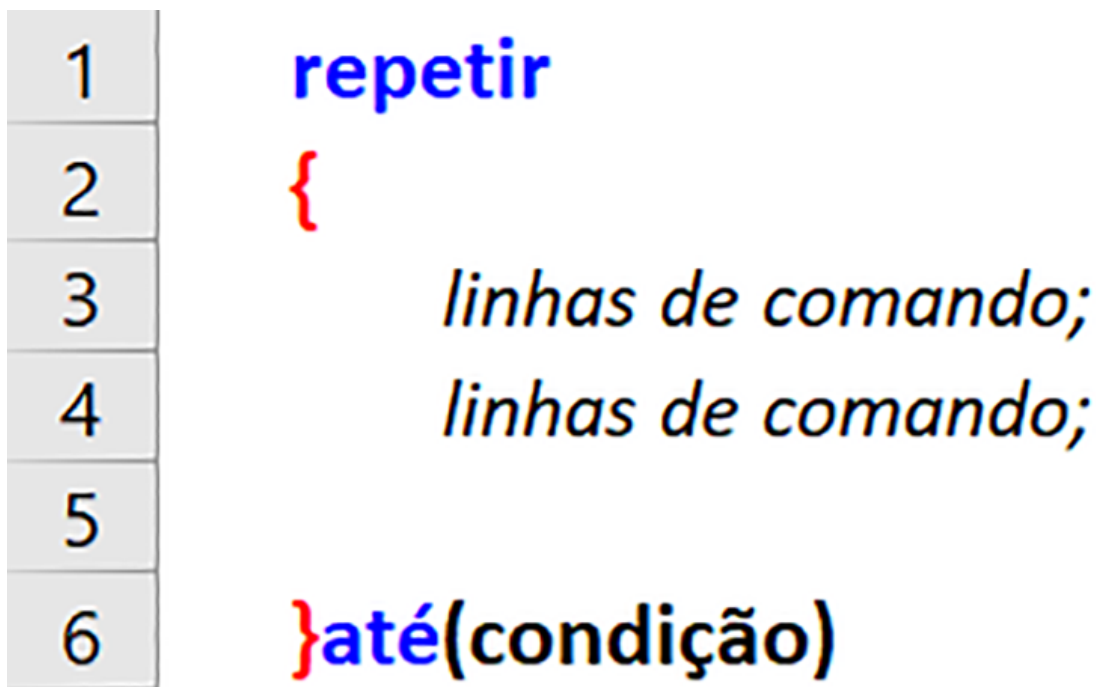


Figura 20 - Sintaxe do comando REPETIR ATÉ.

Fonte: Elaborada pela autora, 2018.

Na linha 1, da figura anterior, temos a palavra chave REPETIR, indicando que é o comando de repetição. O algoritmo não fará nada nesta linha, ele simplesmente irá pular para a próxima linha, no caso, linha 2. As linhas 2 e 6 contêm as chaves ({}) que representam onde começam e terminam as linhas de código que se repetirão.

Na linha 6, também temos a condição, que é uma equação booleana, ou seja, só pode retornar verdadeiro ou falso. Se retornar falso, o algoritmo sairá do fluxo de repetição.

O fluxograma deste comando é apresentado pela próxima figura. Observe a condição no final do comando.

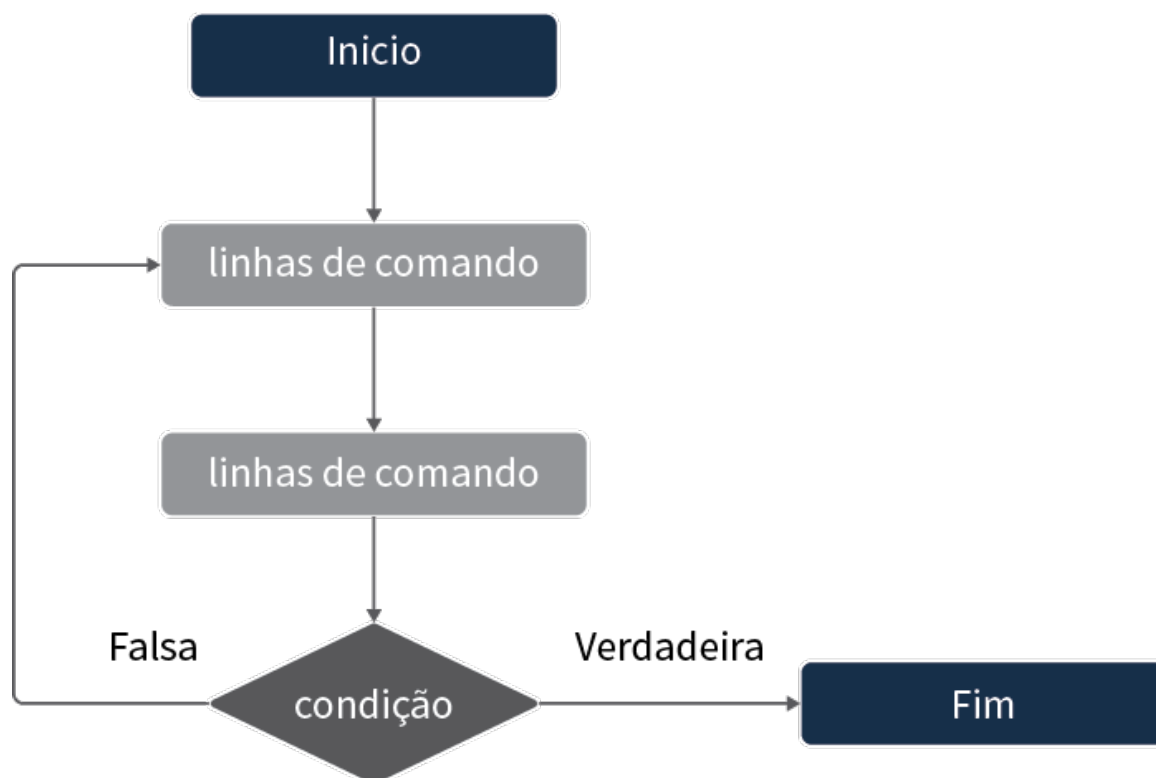


Figura 21 - Fluxograma do comando de repetição REPETIR ATÉ.

Fonte: Elaborada pela autora, 2018.

Vamos ver outro exemplo. Uma professora de matemática precisa de um sistema para auxiliá-la em suas aulas. Os assuntos que ela está abordando em sala de aula são a tabuada de multiplicação e fatorial de um número. Ela gostaria que o sistema pudesse corrigir ou mostrar a resposta ao aluno.

Por exemplo, o aluno está fazendo exercícios a respeito de tabuada do número cinco e quer verificar se o valor está correto. Então, ele executa o programa e obtém a resposta correta. O mesmo esquema deverá ocorrer para o fatorial de um número. Depois de pesquisar a respeito e conversar com vários profissionais da área, ela descobriu que este sistema é viável de ser construído, bastando pra isso, utilizar os comandos de repetição. Ela gostaria que o sistema tivesse uma tela inicial, dando a oportunidade ao aluno de escolher qual opção irá usar.

Que tal construirmos este sistema para calcular a tabuada e o fatorial de um número?

A tabuada de multiplicação, nada mais é do que a multiplicação de um número pelos números de 1 a 10. Veja no exemplo a seguir, a tabuada do número 3.

$$3 \times 1 = 3$$

$$3 \times 2 = 6$$

$$3 \times 3 = 9$$

$$3 \times 4 = 12$$

$$3 \times 5 = 15$$

$$3 \times 6 = 18$$

$$3 \times 7 = 21$$

$$3 \times 8 = 24$$

$$3 \times 9 = 27$$

$$3 \times 10 = 30$$

Vamos analisar com carinho. Observe que o primeiro número, no nosso exemplo o número 3, é o número que desejamos calcular a tabuada. Em seguida, temos o caractere **x** que representa a multiplicação matemática. Logo após o sinal de multiplicação, aparece os números de 1 a 10, acompanhados do sinal de igualdade (=). Por último, o resultado da multiplicação do primeiro número pelo segundo, como você pode ver clicando na interação a seguir.

Montando uma equação temos:

$$\text{numero}_{\text{constante}} \times \text{numero}_{\text{variavel}} = \text{resultado}$$

Onde:

numero_{constante} é o número que desejo calcular a tabuada, ele não muda em todas as linhas;

numero_{variavel} vai de 1 até 10, com passo de 1, ou seja, de 1 em 1;

resultado é a multiplicação do **numero_{constante}** pelo **numero_{variavel}**.

Então, para montarmos um algoritmo para calcular uma tabuada basta repetirmos a equação acima, 10 vezes. Qual comando eu consigo repetir uma linha 10 vezes, gerando números de 1 a 10? Comando de repetição, e nós temos 3. Qual eu devo usar? Qualquer um.

É sério, você pode escolher qualquer um dos três comandos de repetição. Mas, qual é o melhor? O melhor é aquele que você souber usar corretamente. Vou escolher o comando PARA, porque os controles do comando estão todos no mesmo lugar. Este trecho de algoritmo ficará assim:

```
escreva("Digite um número");
leia(num);
para(i = 1; i <= 10 ; i = i + 1)
{
    escreva(num," x ", i, " = ", num*i);
}
```

Depois que você entendeu como fazer a tabuada de multiplicação, vamos para o fatorial.

Fatorial de um número é o produto deste número pelos números menores, em sequência, até o número 1. Por exemplo, o fatorial do número 5 é **$5 \times 4 \times 3 \times 2 \times 1 = 120$** . Agora ficou claro? A representação do fatorial é dada pelo sinal de ! (exclamação). Para o nosso exemplo, seria 5!, leia-se cinco fatorial. Para se trabalhar com fatorial, é importante conhecer algumas definições. Clique nos itens a seguir.

-

Fatorial só pode ser calculado para números inteiros positivos.

-

Fatorial de zero é um ($0! = 1$).

-

fatorial de um é um ($1! = 1$).

Agora, vamos montar a equação do fatorial. Para o 5! temos **$5 \times 4 \times 3 \times 2 \times 1$** , que podemos escrever como **$1 \times 2 \times 3 \times 4 \times 5$** , correto? Pois, para a multiplicação a ordem dos fatores não altera o produto. Observe que temos números de vão de 1 até 5, com passo de 1. Com um comando de repetição eu consigo fazer isto. Este produto pode ser feito em partes.

Primeiro, faremos **1×2** e armazenamos em uma variável, por exemplo ***fatorial***, depois pegamos este valor, no caso 2, multiplicamos por 3 e guardamos na mesma variável ***fatorial***. Neste momento, a variável fatorial está com o valor de 6. Repetimos o processo, pegamos o valor que está em ***fatorial*** e multiplicamos por 4 e guardamos em ***fatorial***. Portanto, a variável ***fatorial*** armazena o valor de 12. Para finalizar, pegamos este valor e multiplicamos por 5 e guardamos em ***fatorial***. Analise o que temos:

fatorial = *fatorial* x *i*

Onde *i* irá variar de 1 até o número

Isso mesmo! É o mesmo formato de um acumulador, só que no acumulador a operação é de soma e aqui vamos utilizar a multiplicação. Na soma, o elemento neutro é o zero, porque o zero não altera o valor da soma. Mas, na multiplicação não podemos usar o zero. O elemento neutro da multiplicação é o número 1. Qualquer número multiplicado por 1, dará o mesmo número. Então, devemos inicializar a variável fatorial com 1. A seguir, é mostrado o algoritmo para calcular o fatorial de um número.

```
escreva("Digite um número");
```

```
leia(num);
```

```
fatorial=1;
```

```
para(i = 1; i <= num ; i = i + 1)
```

```
{
```

```
    fatorial=fatorial * i;
```

```
}
```

```
escreva("O fatorial de ", num, " e ", fatorial);
```

Depois de analisarmos como calcular uma tabuada e o fatorial de um número, podemos montar o algoritmo completo. Como são dois programas separados, será interessante se dermos para o usuário a opção de escolha, bastando, pra isso, uma tela inicial. Então, o primeiro passo será a construção da tela inicial. Analise a figura a seguir.

```
1 INICIO
2 /*Algoritmo para calcular a tabuada e o fatorial de um numero.
3 Para sair do algoritmo, o usuário digitará zero*/
4 //DECLARAÇÃO DE VARIÁVEIS
5 inteiro:op, num, i, fatorial;
6 //LINHAS DE COMANDO OU PROCESSAMENTO
7 repetir
8 {
9     escreva("TELA PRINCIPAL");
10    escreva("Digite 1 para Tabuada");
11    escreva("Digite 2 para Fatorial");
12    escreva("Digite 0 para Sair");
13    leia(op);
14    se(op==1)
15    então
16    {
17        escreva("TABUADA");
18        escreva("Digite um número");
19        leia(num);
20        para(i = 1; i <= 10 ; i = i + 1)
21        {
22            escreva(num," x ", i, " = ", num*i);
23        }
24    }
25    se(op==2)
26    então
27    {
28        escreva("FATORIAL");
29        escreva("Digite um número");
30        leia(num);
31        fatorial=1;
32        para(i = 1; i <= num ; i = i + 1)
33        {
34            fatorial=fatorial*i;
35        }
36        escreva("O fatorial de ", num, " e ", fatorial);
37    }
38 }ate(op != 0);
39 escreva("Voce saiu do programa");
40 FIM
```

Figura 22 - Algoritmo para Calcular a tabuada e o fatorial de um número.

Fonte: Elaborada pela autora, 2018.

O comando REPETIR ATÉ começa na linha 7 e termina na linha 38 da figura anterior. Como estamos fazendo uma tela principal com opções para o usuário, não há a necessidade de um teste antes de exibir as opções, pois a primeira coisa que o usuário precisa ver são as opções. A tela exibida para o usuário é mostrada na próxima figura. Para a exibição desta tela, foram executadas as linhas de 1 até 13. A parada na linha 13 ocorreu por causa do comando de entrada de dados, leia.

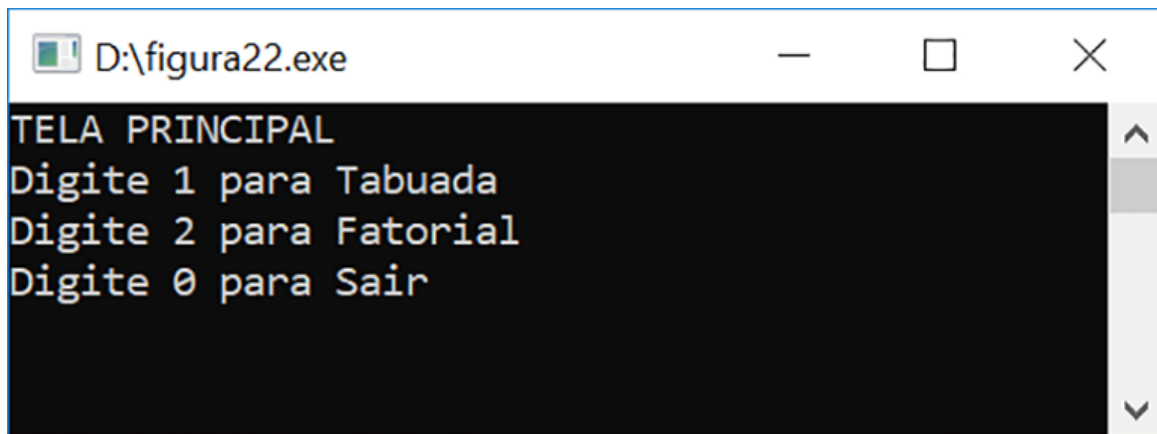


Figura 23 - Tela inicial do programa para calcular a tabuada e o fatorial de um número.

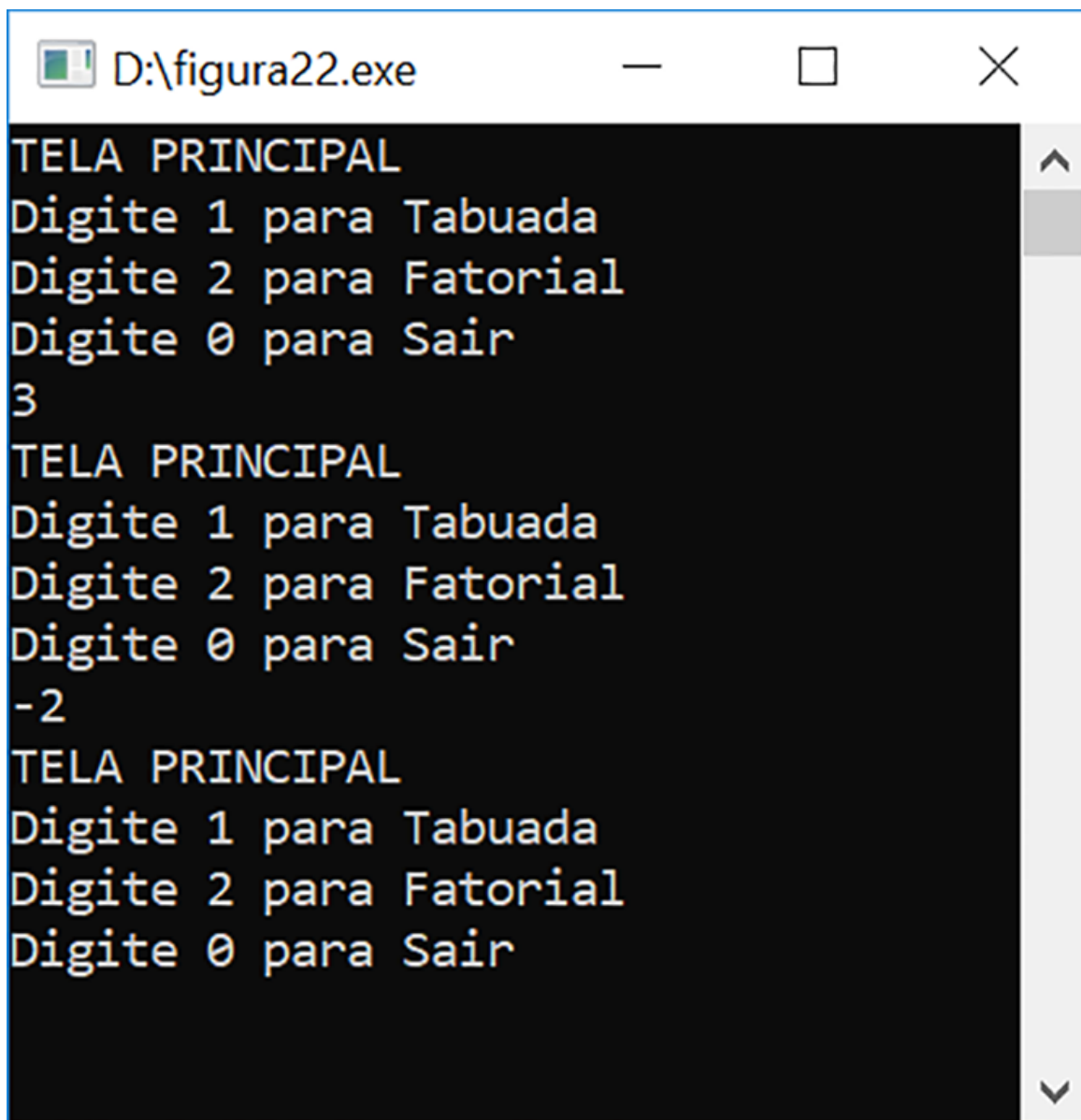
Fonte: Elaborada pela autora, 2018.

Estando na tela principal, o usuário terá as opções de calcular a tabuada de um número se ele digitar 1 ou calcular o fatorial de um número se ele digitar 2 e se quiser sair, basta digitar o 0.

O que acontece se o usuário digitar um número diferente de 0, 1 ou 2? Vamos ver, passo a passo, considerando que o usuário digitou o número 3. Não se esqueça que o programa está parado na linha 13.

- Saindo da linha 13, irá executar o comando condicional SE na linha 14. A condição $op == 1$ retornará falso, pois a variável op tem o valor de 3. Então, o programa irá para a linha 25 da figura que apresenta *Algoritmo para Calcular a tabuada e o fatorial de um número*.
- Na linha 25, existe outro comando condicional e seu teste é $op == 2$, que também retornará falso. Daqui ele pula para a linha 38.
- A condição ou teste do comando de repetição está na linha 38. A condição ($op \neq 0$) retornará verdadeira o que faz o programa retornar para a linha 7, reiniciando o programa.

A figura a seguir mostra a tela para a situação onde o usuário, primeiro, digitou o número 3 e depois o número -2. Observe que a cada número, a tela principal é reimpressa.



```
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
3
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
-2
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
```

Figura 24 - Saída quando o usuário digitou números diferentes de 0, 1 ou 2.

Fonte: Elaborada pela autora, 2018.

Antes de prosseguirmos, analise o algoritmo com o programa em linguagem C, como vemos na próxima figura. O comando REPETIR ATÉ em algoritmo é equivalente ao comando *DO WHILE* em C. Compare linha por linha do algoritmo com o programa equivalente, na figura a seguir.

figura22.cpp

```

1  #include<iostream>
2  using namespace std;
3
4  int main ()
5  {
6      /*Algoritmo para calcular a tabuada e o fatorial de um numero.
7      Para sair do algoritmo, o usuário digitará zero*/
8      //DECLARAÇÃO DE VARIÁVEIS
9      int op, num, fatorial, i;
10     //LINHAS DE COMANDO OU PROCESSAMENTO
11     do
12     {
13         cout<<"TELA PRINCIPAL\n";
14         cout<<"Digite 1 para Tabuada\n";
15         cout<<"Digite 2 para Fatorial\n";
16         cout<<"Digite 0 para Sair\n";
17         cin>>op;
18         if (op==1)
19         {
20             cout<<"TABUADA\n";
21             cout<<"Digite um numero\n";
22             cin>>num;
23             for(i = 1; i <= 10 ; i = i + 1)
24             {
25                 cout<<num<<" x "<< i<<" = "<< num*i<<"\n";
26             }
27         }
28         if (op==2)
29         {
30             cout<<"FATORIAL\n";
31             cout<<"Digite um numero\n";
32             fatorial=1;
33             cin>>num;
34             for(i = 1; i <= num ; i = i + 1)
35             {
36                 fatorial=fatorial*i;
37             }
38             cout<<"O fatorial de "<< num<<" e "<< fatorial<<"\n";
39         }
40     }while(op != 0);
41     cout<<"Voce saiu do programa";
42 }

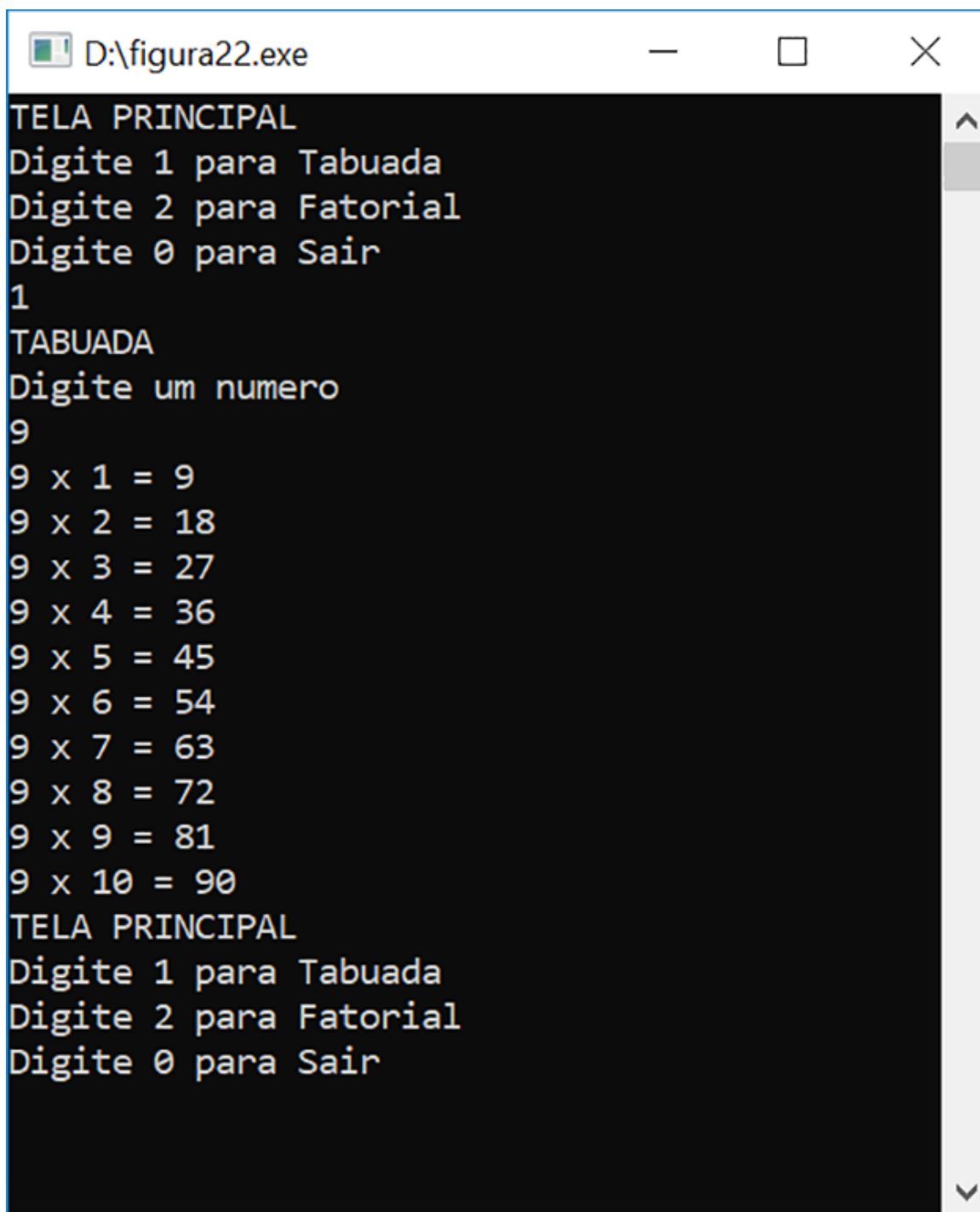
```

Figura 25 - Programa para Calcular a tabuada e o fatorial de um número.

Fonte: Elaborada pela autora, 2018.

Agora, vamos ver o que acontece quando o usuário escolhe as opções corretas. A primeira opção é para o cálculo da tabuada. Acompanhe com o algoritmo, o programa e

a execução do sistema, na figura a seguir.



```
 D:\figura22.exe
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
1
TABUADA
Digite um numero
9
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
```

Figura 26 - Calculando a tabuada.
Fonte: Elaborada pela autora, 2018.

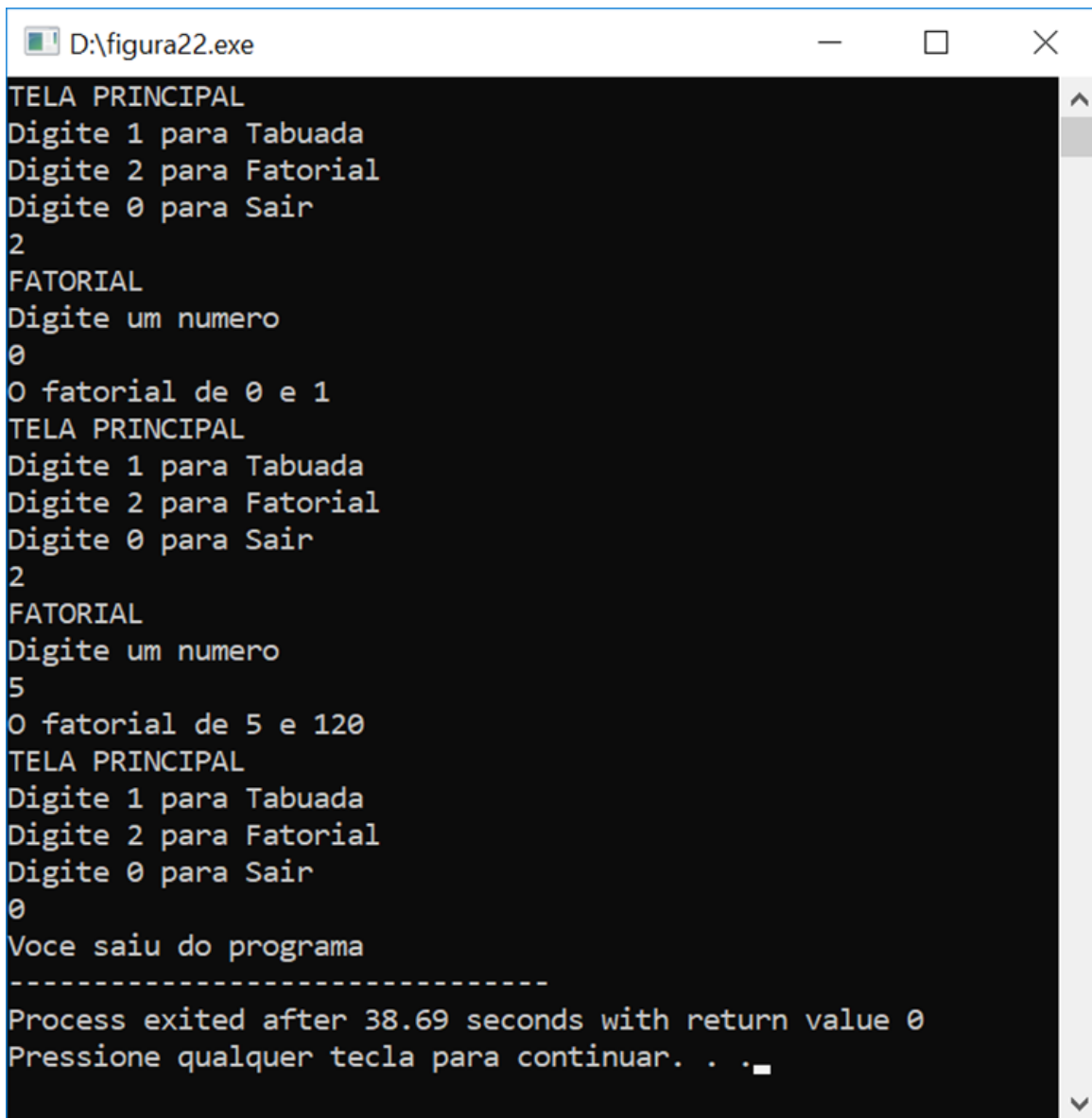
O quadro a seguir descreve as ações do cálculo da tabuada.

Figura Algoritmo n. da Linha	Figura com o programa n. da Linha	Ação
13	17	O usuário digita o número 1 e a tecla <i>enter</i> . Segue para a próxima linha.
14	18	Comando condicional. A condição ($op==1$) retorna verdadeira, irá executar as linhas de código do bloco do então.
17	20	Escreve na tela: TABUADA.
18	21	Escreve na tela: Digite um numero.
19	22	Aguarda o usuário digitar um número. Este número será armazenado na variável <i>num</i> . O usuário digita o número 9, portanto, <i>num</i> recebe o valor de 9. Segue para a linha abaixo.
20	23	Comando de repetição PARA/FOR. Inicializa a variável <i>i</i> com o valor 1 e testa a condição ($i \leq 10$). A condição retorna verdadeiro, então entra dentro do comando de repetição.
22	25	Escreve na tela: $9 \times 1 = 9$.
23	26	Incrementa a variável <i>i</i> com 1. Retorna para o comando de repetição.
20	23	Testa a condição ($i \leq 10$). A condição retorna verdadeiro, então entra dentro do comando de repetição.
22	25	Escreva na tela: $9 \times 2 = 9$
Ele irá repetir este processo até a variável <i>i</i> atingir o valor de 11, assim a condição ($i \leq 10$) será falsa.		
25	28	Comando condicional. A condição ($op==2$) retorna falsa pois o valor de <i>op</i> continua sendo 1.
38	40	Condição do comando REPETIR ATÉ/DO WHILE. A condição ($op!=0$) retorna verdadeira, volta para o início.
7	11	Reinicia o processo.

Quadro 3 - Ações para o cálculo da tabuada.

Fonte: Elaborado pela autora, 2018.

Repita o procedimento para o cálculo do fatorial. A próxima figura irá te ajudar a visualizar a saída na tela.



```
D:\figura22.exe
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
2
FATORIAL
Digite um numero
0
O fatorial de 0 e 1
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
2
FATORIAL
Digite um numero
5
O fatorial de 5 e 120
TELA PRINCIPAL
Digite 1 para Tabuada
Digite 2 para Fatorial
Digite 0 para Sair
0
Voce saiu do programa
-----
Process exited after 38.69 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 27 - Tela de execução do cálculo do fatorial dos números 0 e 5.

Fonte: Elaborada pela autora, 2018.

Estudamos três tipos de comandos de repetição, o PARA, ENQUANTO e o REPETIR ATÉ. Estes três tipos são considerados básicos na programação, praticamente todas as linguagens possuem estes comandos.

Uma dúvida comum para os iniciantes é qual eu devo utilizar? Em um primeiro momento, utilize aquele que você sabe como funciona. Ao resolver um exercício com um comando, repita o mesmo exercício, mas tente outro comando. Na medida em que você for evoluindo no aprendizado perceberá que para cada situação um comando é mais fácil de ser utilizado. Por exemplo, uma situação onde eu sei quando começa, quando

deve terminar e quantas iterações (repetições) deve-se ter, o melhor comando é o PARA. Quando eu não souber quantas iterações o usuário fará, nem quando ele irá parar, o melhor comando é o REPETIR ATE.

CASO

Um shopping resolveu criar uma mascote para as férias do meio do ano. Para isso, organizou uma campanha para incentivar a criação da mascote. Na primeira etapa, um total de 170 crianças inscritas desenharam as mascotes. Destes 170 desenhos, 50 passaram para a segunda etapa e destes 50, apenas dois passaram para a final. Agora, a organizadora da campanha decidiu que o público irá escolher entre as duas mascotes finalistas. Para isso, solicitou que você construísse um algoritmo para permitir a eleição da mascote vencedora. O algoritmo deverá ter uma tela principal com a opção de escolha do voto para a mascote 1 ou a mascote 2. O computador ficará na praça de alimentação e quando um cliente passar poderá digitar seu voto. Para cada voto feito, o algoritmo deverá enviar uma mensagem de voto computador. O resultado só será divulgado no final da votação. Bom trabalho!

Para este exemplo, você utilizará um comando de repetição que não controle a quantidade de iterações e dois comandos condicionais para separar os votos. No final, verifique qual mascote obteve maior quantidade de votos. Observe como fica:

```
1  INICIO
2      inteiro num,contA, contB;
3      contA = 0;
4      contB = 0;
5      repetir
6      {
7          escreva("VOTAÇÃO");
8          escreva("Digite 1 Mascote A");
9          escreva("Digite 2 Mascote B");
10         escreva("Digite 3 Sair");
11         leia(num);
12         se(num == 1)
13             então
14                 contA = contA + 1;
15
16         se(num == 2)
17             então
18                 contB = contB + 1;
19
20     }ate(op != 3);
21     se(contA > contB)
22         então
23             escreva("Mascote A é a vencedora");
24         senão
25             se (contA == contB)
26                 então
27                     escreva("Houve um empate!!");
28                 senão
29                     escreva("Mascote B é a vencedora");
30 FIM
```

Mas, quando eu precisar controlar a execução de um bloco de comandos e não sei quantas execuções serão feitas, o melhor será o comando ENQUANTO.

Em relação a isso, damos algumas dicas a seguir. Clique para ler.

- Não siga para o próximo tópico sem entender o atual.
- Programação não se aprende só olhando, por mais que aparentemente se fácil, tem que praticar muito e um pouquinho de cada vez. Se você deixar para praticar na véspera das provas, não vai dar certo.
- Repita um exercício até compreendê-lo bem.

Finalizamos um capítulo sobre estruturas de repetição. Compreendemos o funcionamento de três estruturas. A primeira, o comando PARA, possui três campos de controle: inicializar variáveis, condição e incremento. Um comando muito útil para situações onde sabemos a quantidade de iterações que serão dadas. O próximo foi o ENQUANTO, que possui apenas um campo, o de condição no início do comando. E por último, o REPETIR ATÉ, que também possui apenas um campo, o de condição, mas no final do comando. Em todos os comandos, quando a condição retornar falsa, o algoritmo sairá do laço.

Síntese

Chegamos ao final deste capítulo. Estudamos diferentes formas de inserir comandos de repetição na programação. Vimos que os comandos de repetição são importantíssimos em programação e são muito utilizados, entre outras funções, eles evitam a repetição de escrita de linhas código. Entendemos o que é e como funciona a estrutura de repetição, os conceitos de contadores e acumuladores, o uso de testes e variáveis de controle.

Neste capítulo, você teve a oportunidade de:

- conhecer o conceito de comandos de repetição, somadores e acumuladores;
- entender a utilização dos conceitos para qualquer tipo de operação e não apenas a soma, como foi o caso do cálculo do fatorial;
- compreender a estrutura do comando de repetição PARA, ideal em situações onde estas três informações são claras;
- entender a utilidade de cada comando;

- empregar o comando ENQUANTO, ideal para situações onde é necessário controlar a entrada no loop e não sabemos quantas iterações serão feitas.



◀ **Clique para baixar o conteúdo deste tema.**

Bibliografia

ASCENCIO, A. F. G.; CAMPOS, E. A. V. **Fundamentos da Programação de Computadores**. São Paulo: Pearson Prentice Hall, 2002.

AHRENS, J. M. **Slavic, o hacker mais procurado (e protegido) do mundo**. Portal El País, publicado em 23 abr 2017. Disponível em: <https://brasil.elpais.com/brasil/2017/04/22/internacional/1492890431_479902.html (https://brasil.elpais.com/brasil/2017/04/22/internacional/1492890431_479902.html)>. Acesso em: 10/01/2019.

BBC NEWS. **Os dez maiores erros de cálculo da ciência e da engenharia**. Portal BBC News. Brasil, publicado em 31 mai. 2014. Disponível em: <https://www.bbc.com/portuguese/noticias/2014/05/140530_erros_ciencia_engenharia_rb (https://www.bbc.com/portuguese/noticias/2014/05/140530_erros_ciencia_engenharia_rb)>. Acesso em: 10/01/2019.

DEITEL, P.; DEITEL, H. C. **C ++ Como Programar**. 6. ed. São Paulo: Pearson Prentice Hall, 2011.

FORBELLONE, A. L. V. **Lógica de programação - A construção de algoritmos e estruturas de dados**. 3. ed. São Paulo: Prentice Hall, 2005.

INOVAÇÃO TECNOLÓGICA. **Memórias de próxima geração são feitas com casca de ovo**. Portal Inovação Tecnológica, publicado em 30/01/2017. Disponível em: <<https://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=memorias-proxima-geracao-feitas-casca-ovo&id=010110170130#.XB5-aFxKg2xn> (<https://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=memorias-proxima-geracao-feitas-casca-ovo&id=010110170130>)>. Acesso em: 10/01/2019.

NEITZKE, N. A. **Boas Práticas de Programação**. Portal Devmedia, Guia Completo de Java, 2011. Disponível em: <<https://www.devmedia.com.br/boas-praticas-de-programacao/21137> (<https://www.devmedia.com.br/boas-praticas-de-programacao/21137>)>. Acesso em: 10/01/2019.

RAMIS, H; RUBIN, D. **Feitiço do tempo**. Direção: Harold Ramis. Produção: Trevor Albert Harold Ramis. Produção executiva: C. O. Erickson. Cor, 103 min. Estados Unidos. 1993.