DIREÇÃO CONCURSOS

Aula 00 – Lógica de Programação

Conhecimentos específicos para Analista Judiciário - Informática do TRF3

Prof. Hamilton Rodrigues

Sumário

Lógica de Programação

Introdução

Caros amigos, futuros servidores públicos, vamos falar de **Lógica de Programação**? Quase todo mundo é capaz de usar um computador: abrir um editor de texto, digitar um documento, entrar em um browser, navegar na internet, baixar seu game favorito no celular e jogar, não é mesmo?

A planilha eletrônica que você usa no computador, o aplicativo no celular, o aplicativo para assistir séries na SmartTV, o sistema que controla o extrato da sua conta corrente no banco, tudo isso são programas, softwares. Para você poder fazer uso desses programas, alguém do outro lado teve que desenvolvê-los.

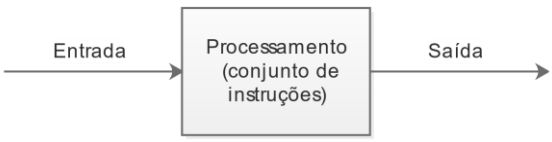
O que possibilita o desenvolvimento de todos esses programas é a técnica da Lógica de Programação.

> ***Lógica de programação**: é uma técnica de resolução de problemas por meio de uma sequência lógica de passos.*

Um dispositivo computacional é como se fosse um robô. Ele não tem vida própria. Para funcionar, necessita que alguém defina para ele um **algoritmo** de funcionamento:

> ***Algoritmo**: conjunto lógico finito bem definido de instruções com o objetivo de cumprir determinada tarefa. Ele recebe as entradas, faz o processamento e retorna uma saída.*

Um algoritmo pode ser representado por meio de um **fluxograma**, isto é, um processo que recebe **entradas** que são processadas por um **conjunto de instruções** e ao final gera uma **saída**. As entradas são dados externos que são fornecidos como ponto de partida para o processamento. Elas não são obrigatórias. Alguns algoritmos funcionam sem nenhum tipo de entrada. O conjunto de instruções tem que ser **finito**. Nosso robô não teria como armazenar em sua memória um número infinito de instruções. Além disso, é importante que as instruções sejam **bem definidas**, não podem ser ambíguas nem inviáveis. Após executar as instruções, é gerada uma **saída**, que é o objetivo para o qual o algoritmo foi feito. Pode ser a solução de um problema ou a execução de uma tarefa.



Resumindo, as características de um **algoritmo** são:

Finito	Conjunto de instruções tem que ser finito.
Bem definido	Não pode haver ambiguidade nem subjetividade nas instruções
Entrada	Parâmetros a partir dos quais é feito o processamento
Saída	Resultado do algoritmo. Solução de um problema ou execução de uma tarefa

OK professor! Mas como eu faço para programar um computador para executar meu próprio algoritmo?

Você deve escrever as instruções em uma linguagem que o computador entenda.

Existe um abismo entre um computador e uma pessoa. O computador entende linguagem de máquina e a pessoa linguagem humana. Quem faz o papel da ponte que transpõe esse abismo e permite que um computador entenda as instruções humanas são as **linguagens de programação**.

Para estudar programação, você não precisa usar uma linguagem específica. Você pode estudar a **lógica de programação**, uma disciplina que foca nos conceitos, estruturas e técnicas de programação de computadores.

Antes de entrarmos no estudo de linguagens de programação específicas nas próximas aulas (Python e R), podemos nos ater por enquanto somente à lógica de programação. Estudaremos o raciocínio por trás do algoritmo. E isso é muito cobrado em concursos públicos!

Professor, mas como eu irei estudar lógica de programação sem usar nenhuma linguagem de programação?!

Por meio de **Pseudocódigo**!

Pseudocódigo é uma forma de escrever algoritmos utilizando palavras intuitivas da língua portuguesa para representar as instruções computacionais. Usando essa linguagem genérica, também chamada de **Portugol**, **Português Estruturado** ou **Metalinguagem**, podemos focar na lógica de solucionar problemas por meio de algoritmos sem nos prendermos a nenhuma linguagem concreta.

Os elementos básicos de um pseudocódigo são: **constantes, variáveis, tipos de dados, operadores, estruturas de decisão, estruturas de repetição e módulos**. Fique tranquilo que iremos estudar cada um desses elementos individualmente.

Constantes e Variáveis

As **constantes** armazenam valores fixos, que não podem ser alterados ao longo de um algoritmo. Já as **variáveis** são dados que podem ser alterados ao longo do código. Tanto as constantes quanto as variáveis podem receber nomes para que sejam identificadas e referenciadas dentro do algoritmo.

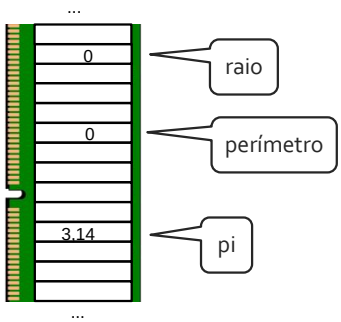
Por exemplo, lembra da fórmula de cálculo do perímetro da circunferência? $perímetro = 2\pi r$. Vamos fazer um pseudocódigo para fazer esse cálculo.

```
01  pi = 3,14
02  raio = 0
03  perímetro = 0
04
05  leia(raio)
06  perímetro = 2 * pi * raio
07  escreva(perímetro)
```

Observe que na 1ª linha é criada uma **constante** com o nome de **pi** que recebe o valor 3,14 e ao longo de todo código ele não muda. Nas linhas 02 e 03 são criadas as variáveis de nome **raio** e **perímetro** respectivamente. Inicialmente é atribuído o valor 0 a cada variável, mas esses valores mudam ao longo do algoritmo. O valor de **raio** muda na linha 05 ao ler o input do usuário. O valor de **perímetro** muda na linha 06 ao receber o resultado da multiplicação 2 * pi * raio. Na linha 07 o valor do perímetro calculado é retornado como **saída** do algoritmo.

Tipos de dados

Vocês repararam no exemplo anterior que tanto a constante quanto as variáveis guardam **dados**? Dados são valores armazenados em determinada região da memória do computador. Os nomes das variáveis (ou constantes) são referências que apontam para uma região específica de memória. Pense na memória do computador como uma longa fita onde cada célula tem um espaço disponível para armazenar um dado.



Os **tipos de dados** podem ser divididos em 2 grupos: **elementares** e **estruturados**.

Dados elementares

Os tipos de dados **elementares** também podem ser chamados de **simples, atômicos** ou **primitivos**. Eles são chamados assim porque não podem ser quebrados em partes menores.

Basicamente há 4 tipos de dados elementares: **lógico, inteiro, real** e **caractere**.

Os dados **lógicos** (ou **booleanos**) são usados quando se quer representar valores como VERDADEIRO/FALSO, TRUE/FALSE, 1/0, SIM/NÃO.

Os **inteiros** (ou **int**) representam dados numéricos sem a parte fracionária como por exemplo 7 (anos de idade), 193 (países no mundo), 200 (vagas para o concurso).

Os **reais** (ou **ponto flutuante**, ou **float**, ou **double**) armazenam dados numéricos com parte fracionária como por exemplo 9,9 (nota na prova do concurso), 1,78 (altura em metros), 1201,50 (saldo na conta corrente) e por aí vai.

Os **caracteres** (ou **char**) são dados que representam letras, dígitos ou símbolos. Exemplos de caracteres: 'a', 'B', 'c', '1', '2', '3', '#', etc.

Resumindo:

Tipos de dados Elementares	Descrição	Exemplos
Lógico	Possui 1 dentre 2 possíveis valores	VERDADEIRO/FALSO TRUE/FALSE 1/0, SIM/NÃO
Inteiro	Número sem parte fracionária	5 12
Real	Número com a parte fracionária	3,1415 10,50
Caractere	Dígito	'a', 'b', 'c', '$', '?', '4'

Dados estruturados

Por outro lado, os tipos de dados **estruturados** são composições feitas em cima dos dados elementares. Por causa disso, os dados estruturados também podem ser chamados de dados **compostos**. Há diversos tipos de dados estruturados. As possibilidades são infinitas, mas podemos destacar alguns tipos principais de dados compostos: **vetores** e **listas**.

Um **vetor** (ou **array** ou **arranjo**) é uma estrutura unidimensional com número fixo de posições que comporta somente dados elementares do mesmo tipo.

Explicando melhor, um exemplo de vetor de inteiros com 7 posições:

4	2	1	3	3	5	2

Um caso particular de vetor muito utilizado na programação é o **string**, que nada mais é que um vetor de caracteres mais conhecido como **cadeia de caracteres**.

Veja 2 exemplos de strings, um com 4 caracteres e outro com 5.

'J'	'o'	's'	'é'

'M'	'a'	'r'	'i'	'a'

No pseudocódigo, o string é representado como uma palavra entre aspas. Veja um exemplo de declaração de uma constante para armazenar um string.

```
nome = "José Maria"
```

Outro tipo de dado bastante comum em programação é a **lista**. Ela é muito parecida com um vetor, isto é, armazena diversas ocorrências de dados elementares do mesmo tipo, só que, ao contrário do vetor que tem **tamanho fixo**, a lista pode ter **tamanho variável**. Ela pode ser criada vazia, ir aumentando o número de posições para receber mais dados. Ou diminuir, se não houver mais necessidade de espaço.

Recapitulando:

Tipos de dados Estruturados	Descrição	Exemplos
Vetor	Sequência de tamanho fixo de dados elementares do mesmo tipo	[1, 7, 4, 8, 23] [25.50, 46.30, 99.99]
String	Vetor de caracteres	"Polícia Federal"
Lista	Sequência de tamanho variável de dados elementares do mesmo tipo	(igual aos vetores, só que podendo aumentar ou diminuir de tamanho)

Vamos fazer uma questão para fixar esses conhecimentos sobre tipos de dados?

(COPS-UEL - 2015 - UEL - Agente Universitário - Técnico em Informática)

Em relação a tipos de dados, atribua V (verdadeiro) ou F (falso) às afirmativas a seguir.

( ) Char é um tipo simples.

( ) Double é um tipo estruturado.

( ) Float é um tipo simples.

( ) Int é um tipo simples.

( ) Void é um tipo estruturado.

Assinale a alternativa que contém, de cima para baixo, a sequência correta.

(a) V, V, F, F, F.

(b) V, F, V, V, F.

(c) F, V, V, F, V.

(d) F, F, V, F, V.

(e) F, F, F, V, V.

RESOLUÇÃO:

() Char é um tipo simples? Verdadeiro. Char representa uma letra, dígito ou número. É um tipo de dado que não pode ser subdividido.

() Double é um tipo estruturado? O Double é um tipo de ponto flutuante (assim como o float) só que com o dobro do espaço de memória do float. Assim, ele pode armazenar números reais com maior precisão em memória, mais casas decimais. Alternativa Falsa porque tanto o Double quanto o Float são tipos simples.

( ) Float é um tipo simples? Verdadeiro

( ) Int é um tipo simples? Verdadeiro. Int refere-se a um tipo de dado elementar numérico inteiro.

( ) Void é um tipo estruturado? Falso. Void é a mesma coisa que nada. É considerado um tipo elementar.

Resposta: B – V,F,V,V,F

Entrada/saída de dados

Como vimos anteriormente, um algoritmo recebe inicialmente dados de entrada que são processados e ao fim do processamento são gerados os dados de saída. Quando você entra no seu computador e abre o aplicativo da calculadora, a entrada são os números e as operações que você quer calcular. A saída é o resultado do cálculo.

Em computação, o termo **entrada/saída** de dados também é muito utilizado em inglês: **input/output**. Os dados de entrada podem ser números, textos, áudio, vídeo, etc. Idem para os dados de saída.

Seguem alguns exemplos de dispositivos de entrada e saída de dados.

Operadores

Até agora estudamos constantes e variáveis que por sua vez armazenam dados. E que há vários tipos de dados: inteiros, reais, lógicos, caracteres, vetores, listas, etc.

E como fazer os processamentos em cima dos dados? Por meio de **operadores**!

É como numa calculadora. Você entra com os dados e sobre eles aplica as operações.

Operadores são utilizados para criar expressões matemáticas e podem ser de 3 tipos: **aritméticos**, **relacionais** e **lógicos**.

Operadores aritméticos

São utilizados para operações numéricas entre os operandos. Aquelas operações bem básicas como **somar**, **subtrair**, **multiplicar**, **dividir**, **exponenciar**, por exemplo.

Além desses, programação também é muito utilizado o operador **módulo** para obter o resto da divisão inteira entre dois números. Usamos o símbolo **%** (ou **mod**) para representar esse operador módulo. Por exemplo, para calcular o resto da divisão inteira entre 13 e 3 escrevemos **13 % 3**. O resultado dessa operação é 1.

Como assim professor?

Vamos lá, com calma. Dividindo 13 por 3 chegamos a 4 (quociente inteiro) e 1 (resto). Esse resto é o módulo.

13	3
1	4

Pessoal, muita atenção com a ordem de prioridade dos operadores!

Se no meio de um algoritmo você encontrar a seguinte expressão, qual valor será armazenado na variável x?

```
x = 2 + 3 * 5
```

O resultado nesse caso é 17! O operador multiplicação (*) tem prioridade sobre a adição (+). Portanto, o computador faz primeiro a multiplicação (3 * 5 = 15) e depois pega o resultado (15) e soma com 2 resultando no valor 17.

Se, ao escrever o seu pseudocódigo, você quisesse aplicar primeiro a soma e só depois a multiplicação, poderia fazê-lo utilizando parênteses da seguinte forma.

```
x = (2 + 3) * 5
```

Neste caso, o resultado seria 25. Os parênteses "forçam a barra" para que a adição seja feita antes da multiplicação. Se você não colocar parênteses, vale a ordem de prioridade normal entre os operadores, da esquerda para a direita.

Veja na tabela a seguir um resumo dos operadores aritméticos com a respectiva prioridade de cada um. Se a prioridade empatar, vale o operador que aparecer primeiro.

Operador	Símbolo	Prioridade	Exemplo
Exponenciação	^	1ª	2 ^ 3 = 8
Multiplicação	*	2ª	3 * 4 = 12
Divisão	/	2ª	8 / 4 = 2
Módulo	%	2ª	10 % 3 = 1
Adição	+	3ª	2 + 7 = 9
Subtração	-	3ª	5 – 1 = 4

É necessário se habituar à notação dos operadores para entender e escrever pseudocódigos corretamente. A notação algébrica, que a gente aprende na escola, é um pouco diferente da notação da programação. Independente disso, o resultado é o mesmo.

Expressão algébrica	Equivalente em pseudocódigo
$y = 3\frac{x}{2}$	y = 3 * x / 2
$z = 3bc + 4$	z = 3 * b * c +4
$a = \frac{x+2}{a-1}$	a = (x + 2) / (a – 1)
$x = \frac{a^2 + b^3}{5}$	x = (a ^ 2 + b ^ 3) / 5

Operadores relacionais

São utilizados para fazer **comparações** entre operandos. Seu retorno é sempre um valor do tipo booleano, isto é, VERDADEIRO ou FALSO.

Operador	Símbolo	Uso
Igual	=	x = y
Diferente	<> ou !=	x <> y ou x != y
Maior	>	x > y
Menor	<	x < y
Maior ou igual	>=	x >= y
Menor ou igual	<=	x <= y

Operadores lógicos

Servem para combinar operandos booleanos (VERDADEIRO ou FALSO) e retornam valores também booleanos.

Operador	Significado
E / And	Combina 2 operandos booleanos retornando VERDADEIRO somente quando os 2 operandos são VERDADEIROS.
Ou / Or	Combina 2 operandos booleanos retornando FALSO somente quando os 2 operandos são FALSOS.
Não / Not	Operador unário (é aplicado sobre somente 1 operando) que inverte um valor booleano. Se o operando for VERDADEIRO o retorno é FALSO e vice versa.

Para ficar mais claro, apresentamos as tabelas-verdade dos 3 operadores lógicos **E, OU** e **NÃO**.

Expressão	Resultado
Verdadeiro **E** Verdadeiro	Verdadeiro
Verdadeiro **E** Falso	Falso
Falso **E** Verdadeiro	Falso
Falso **E** Falso	Falso

Expressão	Resultado
Verdadeiro **Ou** Verdadeiro	Verdadeiro
Verdadeiro **Ou** Falso	Verdadeiro
Falso **Ou** Verdadeiro	Verdadeiro
Falso **Ou** Falso	Falso

Expressão	Resultado
Não Verdadeiro	Falso
Não Falso	Verdadeiro

(CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o próximo item, concernente ao conceito relacionado a algoritmos e linguagens de programação.

A resposta da expressão a seguir é verdadeiro.

```
se ((-(-2-6*12/3-1)) > (3+3-3*3-3^3+3)) então
    escreva "verdadeiro";
senão
    escreva "falso";
```

RESOLUÇÃO:

Esta questão resume boa parte do que acabamos de ver em operadores. A chave para acertá-la é saber a precedência entre os operadores.

Vamos começar pela expressão do lado esquerdo.

(-(-2-6*12/3-1)) =

(-(-2-72/3-1))=

(-(-2-24-1))=

(-(-26-1))=

(-(-27))=

27

E agora o lado direito.

(3+3-3*3-3^3+3)=

(3+3-3*3-27+3)=

(3+3-9-27+3)=

(6-9-27+3)=

(-3-27+3)=

(-30+3)=

-27

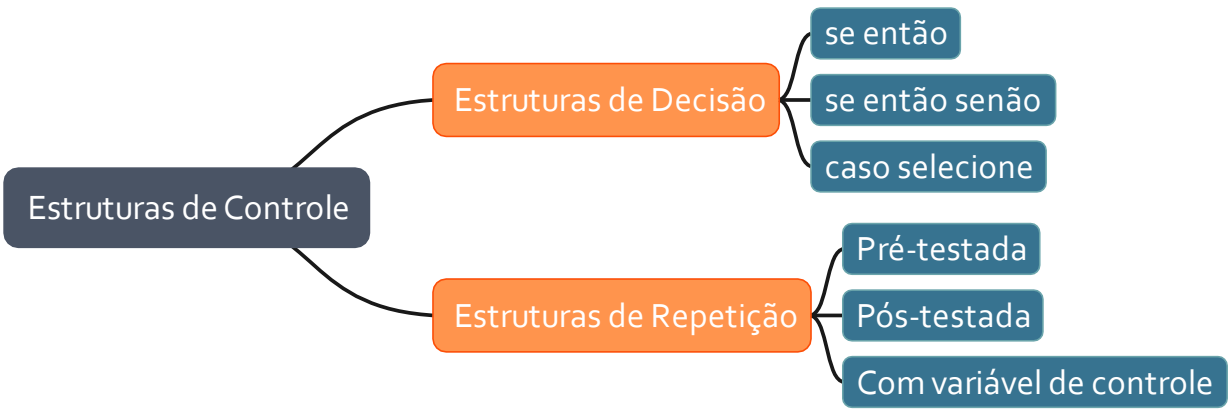
Voltando ao pseudocódigo do enunciado, como 27 > -27, a condição é verdadeira e, portanto, o retorno do algoritmo é "verdadeiro".

Resposta: Certo

Estruturas de controle

As **estruturas de controle** permitem ao programador mudar a direção do fluxo de execução das instruções do algoritmo baseado em certos parâmetros.

Tem dois tipos de estruturas de controle: as **estruturas de decisão** e as **estruturas de repetição**.



Estruturas de decisão

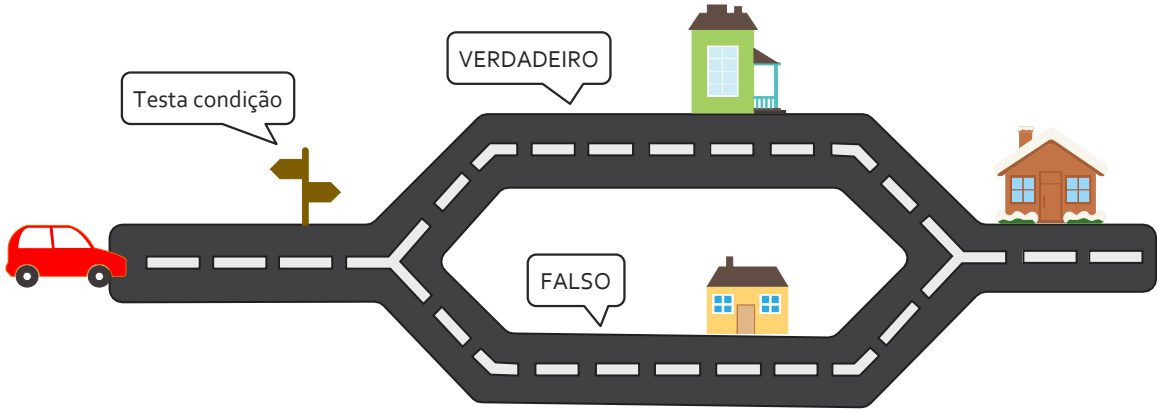
O algoritmo em pseudocódigo é executado sequencialmente de cima para baixo na ordem em que as instruções aparecem. Por padrão, nenhuma instrução é pulada, "faça chuva ou faça sol".

Podemos fazer uma analogia da **execução sequencial** de um algoritmo com um carro andando numa estrada sem nenhuma bifurcação. Partindo do ponto inicial, só há uma forma possível de percorrer todo o caminho pela estrada que é passando em sequência por todos os pontos da estrada. Observe que é um trajeto linear.

Um dos mecanismos que te permite quebrar essa linearidade e executar uma instrução somente sob determinada condição são as **estruturas de decisão**. Uma estrutura de decisão (também conhecida como **estrutura condicional** ou **estrutura de seleção**) permite desviar o fluxo de execução de um programa caso determinada condição seja satisfeita.

Na prática, o algoritmo raramente vai ser linear. Ele terá no meio da estrada bifurcações onde ele terá de decidir para que lado seguir.

Na analogia abaixo, antes da bifurcação, o algoritmo testa uma condição. Se verdadeira, segue pelo caminho de cima. Caso contrário, segue pelo de baixo.

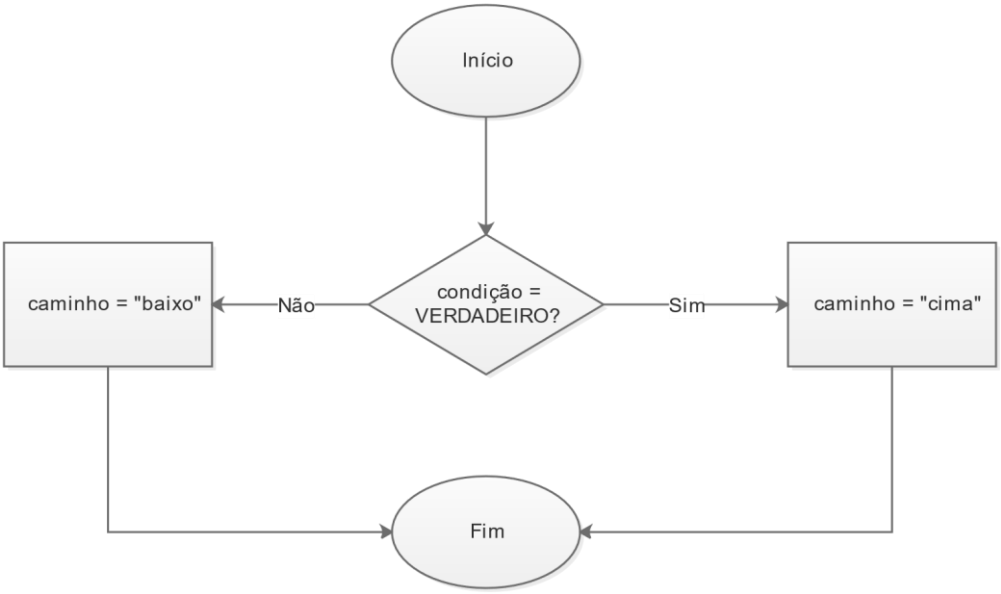


Em pseudocódigo, isso é representado pela estrutura **se-então-senão** (ou **if-then-else**).

Veja um exemplo.

```
se (condição = VERDADEIRO) então
    caminho = "cima"
senão
    caminho = "baixo"
```

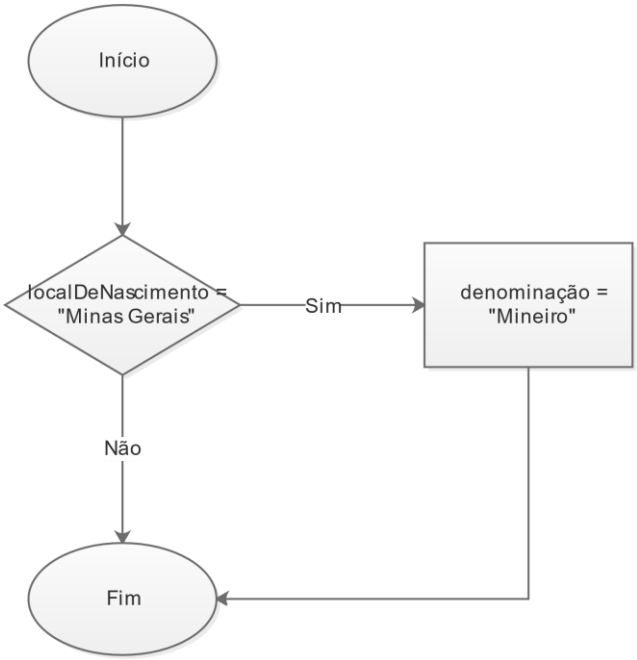
Esta mesma lógica também pode ser representada por um fluxograma.



O **senão** não é obrigatório. A estrutura condicional pode tomar uma forma mais simples, apenas com o **se-então,** conforme exemplo abaixo.

```
se (localDeNascimento = “Minas Gerais”) então
    denominação = “Mineiro”
```

O fluxograma correspondente a essa lógica não teria o caminho alternativo **senão**.



Temos também um terceiro caso de estrutura de decisão chamada de **caso-selecione** (ou **switch-case** ou **múltipla-escolha**).

```
selecione (númeroMês)
    caso 1:
      mês = "Janeiro"
    caso 2:
      mês = "Fevereiro"
    caso 3:
      mês = "Março"
    caso 4:
      mês = "Abril"
    caso 5:
      mês = "Maio"
    caso 6:
      mês = "Junho"
    caso 7:
      mês = "Julho"
    caso 8:
      mês = "Agosto"
    caso 9:
      mês = "Setembro"
    caso 10:
      mês = "Outubro"
    caso 11:
      mês = "Novembro"
    caso 12:
      mês = "Dezembro"
    caso outro:
      escreva("Você deve selecionar um número de 1 a 12")
fim selecione
```

Repare que na 1ª linha ele avaliou o valor da variável númeroMês e abriu 13 blocos para tratamento desse valor: 1 para cada mês do ano e ao final um **caso outro,** que serve para o algoritmo tratar um caso fora do esperado.

Os casos são exclusivos. Seja qual for o número armazenado na variável númeroMês, somente 1 caso será executado e os demais ignorados. O **caso outro** é executado caso o valor de númeroMês não caia em nenhum dos 12 casos previstos.

Para não perder o costume, vamos fazer duas questões de concurso para fixar esse conhecimento de estruturas de decisão.

(CESPE - 2013 - SERPRO - Programador de computador)

No que se refere a linguagens de programação e estruturas de decisão/repetição em algoritmos de programação, julgue os itens subsequentes.

A estrutura de decisão SE/ENTÃO/SENÃO, ou IF/THEN/ELSE, permite que seja sempre executado um comando. Isso porque, caso a condição seja verdadeira, o comando da condição SE/ENTÃO será executado; caso contrário, o comando da condição SENÃO (falsa) será executado.

RESOLUÇÃO:

A estrutura SE/ENTÃO/SENÃO garante que sempre será executado um comando. Ou o comando dentro do SE/ENTÃO ou o do SENÃO. Não tem como "escapar" de uma dessas duas possibilidades.

É como a figura da estrada com bifurcação. Ou o carro vai pelo trecho de cima ou pelo de baixo.

Resposta: Certo

(CESPE - 2014 - TJ-SE - Técnico Judiciário - Programação de Sistemas)

No que se refere à estrutura de programação e lógica, julgue os próximos itens.

No algoritmo abaixo, é apresentada uma estrutura de desvio condicional encadeada.

```
Se <condicao> então
      Comando1
      Comando2
Senão
     Se <condicao> então
         Comando3
         Comando4
     Senão Comando5
     Fim-Se
Fim-Se
```

RESOLUÇÃO:

Vamos entender a estrutura deste código.

```
Se <condicao> então
      Comando1
      Comando2
Senão
     Se <condicao> então
         Comando3
         Comando4
     Senão Comando5
     Fim-Se
Fim-Se
```

Bloco de comandos a ser executado caso <condição> seja VERDADEIRA.

Bloco de comandos a ser executado caso <condição> seja FALSA. Neste caso, o bloco é uma estrutura **se-então-senão encadeada**. Não tem problema nenhum nisso. Os blocos interiores ao **se-então** e ao **senão** podem ser quaisquer estruturas válidas em pseudocódigo, inclusive outro bloco **se-então-senão**.

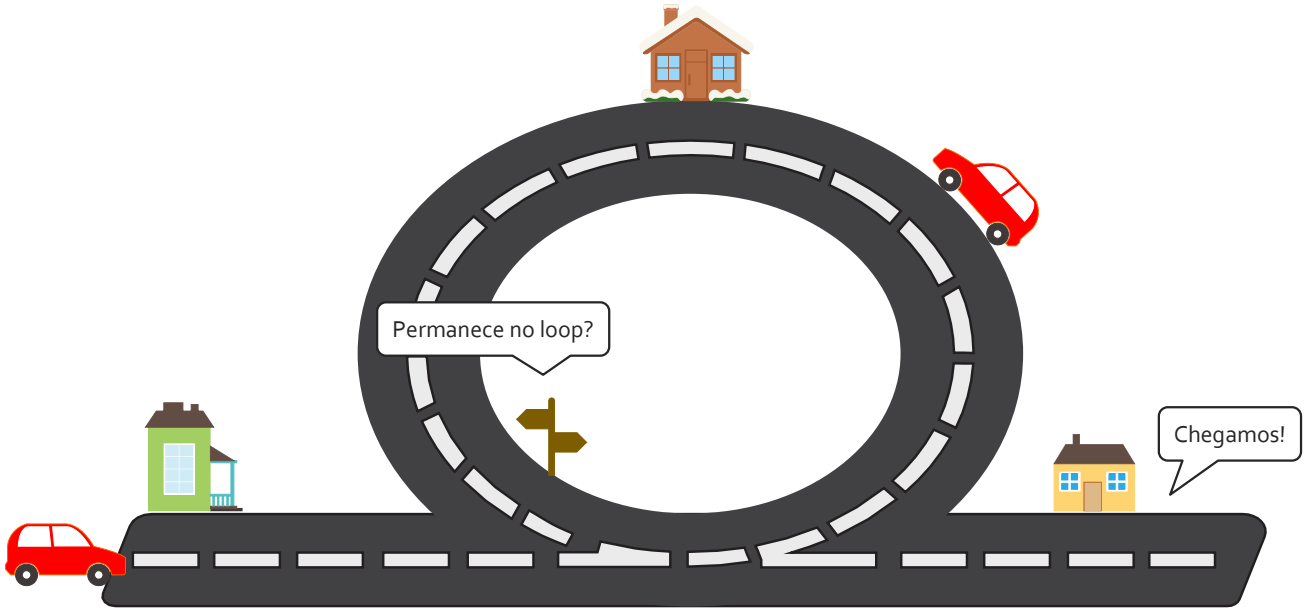
Está correto galera. É permitido encadear blocos **se-então-senão** dentro de outro bloco **se-então-senão**. Isso é chamado de **desvio condicional encadeado** (ou **aninhado**) e é muito comum em programação.

Resposta: Certo

Estruturas de repetição

Outro tipo de estrutura muito importante na lógica de programação são as **estruturas de repetição**. Estas estruturas, também chamadas de **laços** (ou **loops**), são empregadas quando o programador deseja que determinado bloco de instruções seja executado diversas vezes de forma repetitiva.

Observe na ilustração abaixo o princípio de funcionamento de uma estrutura de repetição. O algoritmo executa repetidamente a volta enquanto determinada condição é satisfeita (Permanece no loop?). Quando deixa de satisfazer, o algoritmo sai do loop e segue para executar os próximos blocos de instruções.



Existem 3 tipos de estruturas de repetição: a repetição **pré-testada**, a **pós-testada** e a **repetição com variável de controle**.

Repetição pré-testada

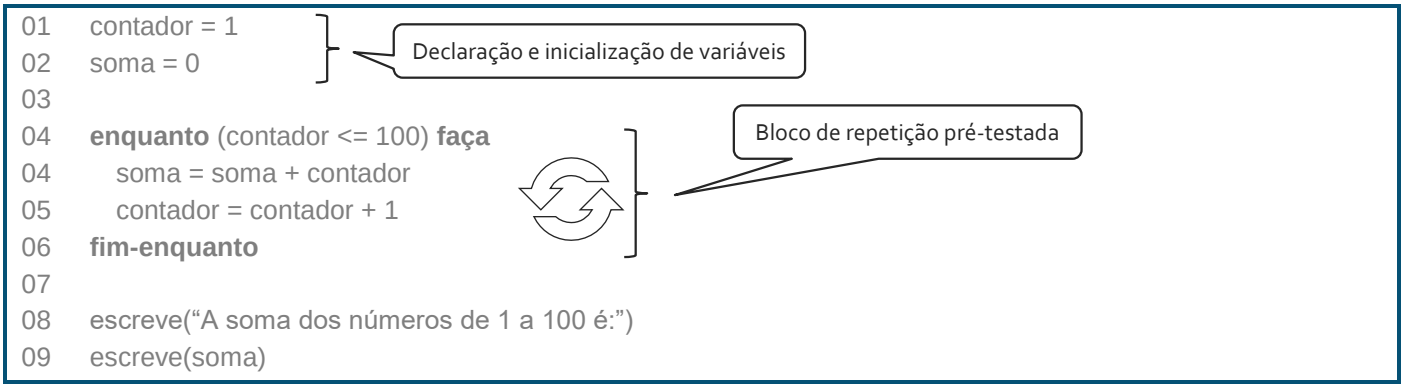
Esta estrutura (também conhecida como **while-do)** testa a condição antes de entrar no loop. Caso a condição seja verdadeira ele permanece no loop enquanto ela continuar sendo verdadeira. No momento em que deixa de ser verdadeira, sai do loop.

Usando nossa metáfora da estrada com loop, na repetição pré-testada é feito um teste inicial antes de entrar no loop a 1ª vez. Se esse teste der falso, ele nem entra no loop. A sintaxe em pseudocódigo da repetição pré-testada é a seguinte.

```
enquanto (condição = VERDADEIRO) faça
        <bloco de comandos>
fim-enquanto
```

Vamos a um exemplo mais prático. Suponha que você queira calcular a soma dos números de 1 a 100 utilizando uma repetição pré-testada.

```
01  contador = 1
02  soma = 0
03
04  enquanto (contador <= 100) faça
04     soma = soma + contador
05     contador = contador + 1
06  fim-enquanto
07
08  escreve("A soma dos números de 1 a 100 é:")
09  escreve(soma)
```



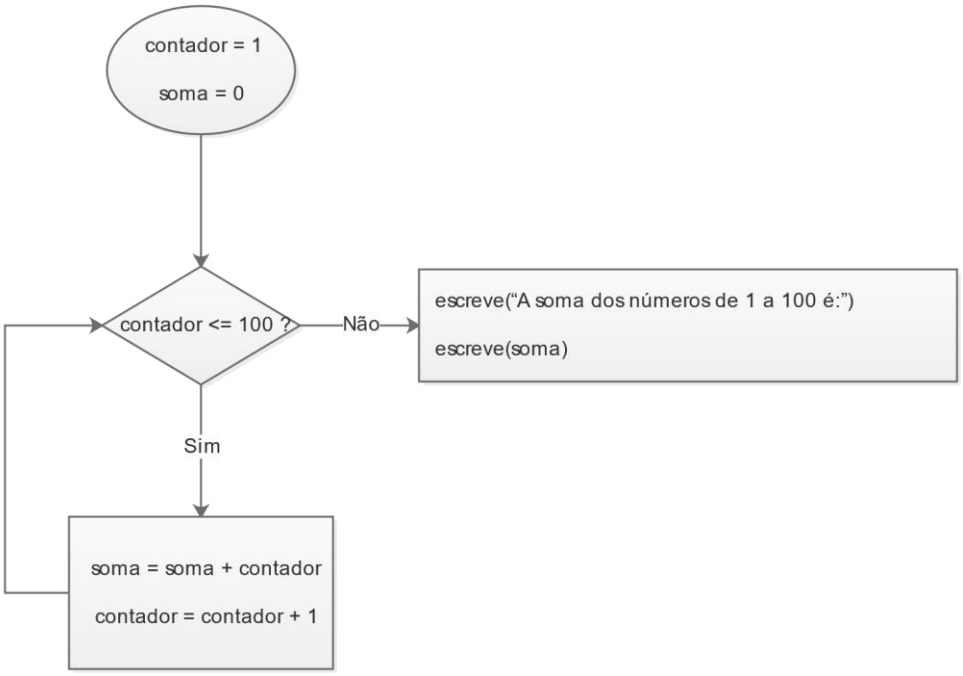
Observe que na linha 01 o programador declarou a variável contador e inicializou-a com o valor 1. O papel dessa variável será fazer a contagem de 1 a 100.

Na linha 02 foi declarada a variável soma. Essa variável terá a função de armazenar a soma dos números de 1 a 100.

Na linha 04 começa nossa estrutura de repetição. Reparem que ela faz um pré-teste (contador <= 100). Se o valor do contador não fosse menor ou igual a 100 neste ponto, o algoritmo não faria repetição nenhuma e pularia direto para a linha 8. Como o teste é verdadeiro, ele passa para o bloco de loop (linhas 04 e 05). Esse bloco será executado repetidamente 100 vezes. A cada passada, a variável soma vai somando o valor do contador e o contador é incrementado em 1 unidade para a próxima passagem pelo loop.

Na 101ª vez que o loop passar pelo teste da linha 4, a variável contagem terá o valor 101 e, portanto, o resultado do pré-teste será falso. Nesse ponto, ele irá sair do loop e passará para a linha 08. A essa altura, a variável soma já tem o resultado do cálculo desejado que é a soma de 1 a 100.

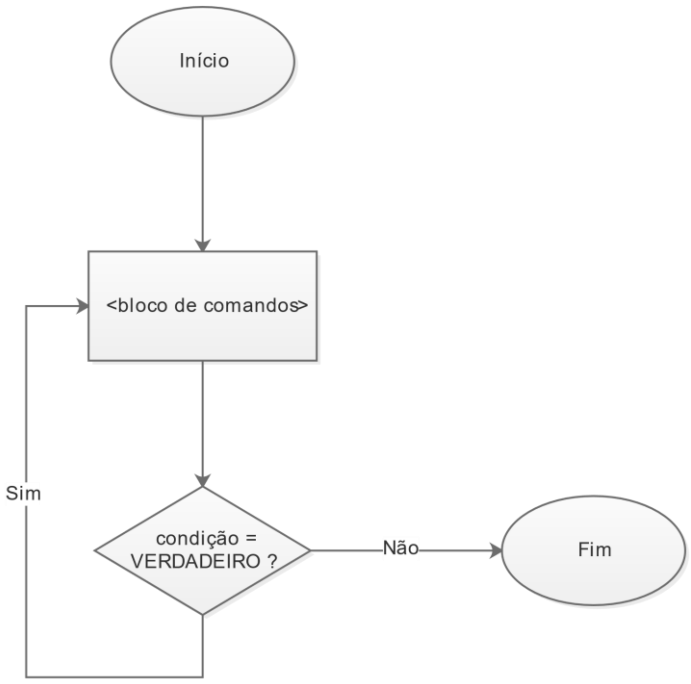
Podemos fazer um fluxograma para visualizar graficamente essa mesma lógica.



Repetição pós-testada

A **repetição pós-testada** (ou **do-while** ou **repeat-until**) executa primeiramente 1 vez o bloco de repetição e faz o teste para decidir se permanecerá repetindo após ao término dessa 1ª volta no loop. Por causa disso, ela executará no mínimo 1 loop, diferentemente da pré-testada, que pode não executar o loop nenhuma vez, dependendo do pré-teste. A estrutura de repetição **pós-testada** é a seguinte.

```
faça
    <bloco de comandos>
enquanto (condição = VERDADEIRO)
```



A diferença entre a repetição pré e a pós-testada é cobrada em concursos. Fique ligado! Veja as questões abaixo.

(CESPE - 2018 - Polícia Federal - Perito Criminal Federal - Área 3)

Em relação às estruturas de controle e de fluxo de execução, julgue o item seguinte.

O laço *do-while* será executado sempre que a condição for falsa e terminará quando esta for verdadeira, ao passo que o laço *repeat-until* será executado sempre que a condição for verdadeira e terminará quando esta for falsa.

RESOLUÇÃO:

Primeira coisa que devemos nos atentar é que os laços *do-while* e *repeat-until* são a mesma coisa: estrutura de repetição pós-testada. Só isso seria suficiente para matar a questão como errada.

Além disso, a questão diz que "O laço *do-while* será executado sempre que a condição for falsa". Na verdade, a 1ª rodada do laço é executada independentemente do valor da condição. A partir da 2ª rodada, só será executado se a condição for verdadeira.

Resposta: Errado

(CESPE - 2018 - Polícia Federal - Perito Criminal Federal - Área 3)

Em relação às estruturas de controle e de fluxo de execução, julgue o item seguinte.

Nos laços *while* e *repeat-until*, as sentenças serão executadas pelo menos uma vez.

RESOLUÇÃO:

No *while*, como o teste da condição para permanecer no loop é feito previamente, se ela não for verdadeira inicialmente, as sentenças do loop não serão executadas nenhuma vez.

Somente o *repeat-until* garante que as sentenças serão executadas pelo menos uma vez, devido ao fato de o teste ser feito após a 1ª execução do loop.

Resposta: Errado

Repetição com variável de controle

Na **repetição com variável de controle**, também conhecida como **para** ou **for**, o número de vezes que o loop será repetido é definido previamente por meio de uma variável de controle. A forma de declaração desta estrutura é a seguinte.

```
para contador de 1 até n faça
        <bloco de comandos>
fim-para
```

O bloco de comandos será repetido n vezes. A variável de controle foi chamada de contador, mas poderia ser qualquer variável do tipo inteiro. Observe que no código não há menção explícita ao incremento do contador a cada volta no loop. Esse incremento é implícito. O contador é inicializado com o valor 1 e a cada iteração é incrementado de 1 em 1 até atingir o valor n. Isso está implícito na declaração da estrutura do **for**.

Pessoal, agora que já estudamos **variáveis, constantes, operadores, estruturas de seleção** e de **repetição** podemos juntar isso tudo e fazer um algoritmo mais elaborado. Todos esses elementos de lógica de programação foram criados justamente para serem combinados uns com os outros e assim resolvermos problemas do mundo real por meio de algoritmos. Vamos a um caso concreto.

Suponha que você trabalhe numa agência que regula os preços dos pedágios nas estradas do seu estado. Seu chefe chega para você e passa a seguinte tabela, com os novos preços e forma de cálculo dos pedágios.

Categoria do veículo	Preço base	Multiplicador	Total (preço base x multiplicador)
0	0,00	0	0,00
1	3,00	1	3,00
2	3,00	1	3,00
3	3,00	2	6,00
4	3,00	2	6,00
5	3,00	3	9,00
6	3,00	3	9,00
7	3,00	4	12,00
8	3,00	1,5	4,50

Ele pede para você transformar essa tabela em um algoritmo a ser implantado nos caixas dos pedágios nas estradas. Você não tem acesso às máquinas. Portanto, não irá programar de fato os caixas. Sua função é especificar a lógica de programação que posteriormente será implementada no software dos caixas de pedágio.

E aí? Como você poderia especificar o algoritmo sem usar nenhuma linguagem de programação específica?

Por meio de um pseudocódigo! Essa questão caiu em um concurso de 2017 da banca FCC. Veja como:

(FCC - 2017 - ARTESP - Agente de Fiscalização à Regulação de Transporte - Tecnologia de Informação)

Considere o algoritmo em pseudocódigo abaixo.

```
Var pedagio, tm: real
    categoria: inteiro

Início
   tm ← 3.00
   enquanto (verdadeiro) faça
       imprima(" Digite a categoria do veiculo (0 a 8) ")
       leia (categoria)

       se (categoria < 0 e categoria > 8)
         então vá para FINALIZA
       fim se

       escolha(categoria)
         caso 0:    pedagio ← 0
         caso 1, 2: pedagio ← tm
         caso 3, 4: pedagio ← 2 * tm
         caso 5, 6: pedagio ← 3 * tm
         caso 7:    pedagio ← 4 * tm
         caso 8:    pedagio ← 1.5 * tm
       fim escolha

       imprima("O veiculo de categoria ",categoria, " pagara pedagio= ",pedagio)
   fim enquanto
   FINALIZA:
Fim.
```

Este algoritmo

(a) não poderia usar a categoria o no comando escolha, nem atribuir zero ao valor do pedágio

(b) apresenta erro de lógica na condição do comando condicional se.

(c) teria que usar uma condição no comando enquanto (verdadeiro) faça, pois este não pode avaliar apenas o valor lógico verdadeiro.

(d) tem erro de sintaxe, pois o comando escolha deveria estar dentro da cláusula senão do comando condicional se.

(e) tem erro de sintaxe, pois o comando escolha deveria ter a cláusula senão, que é obrigatória

RESOLUÇÃO:

Observe que está em português estruturado e seu entendimento é natural. Vamos analisar passo a passo. O pseudocódigo é executado sequencialmente na ordem das instruções, de cima para baixo.

```
Var pedagio, tm: real
    categoria: inteiro

Início
   tm ← 3.00
   enquanto (verdadeiro) faça
       imprima(" Digite a categoria do veiculo (0 a 8) ")
       leia (categoria)

       se (categoria < 0 e categoria > 8)
         então vá para FINALIZA
       fim se

       escolha(categoria)
         caso 0:    pedagio ← 0
         caso 1, 2: pedagio ← tm
         caso 3, 4: pedagio ← 2 * tm
         caso 5, 6: pedagio ← 3 * tm
         caso 7:    pedagio ← 4 * tm
         caso 8:    pedagio ← 1.5 * tm
       fim escolha

       imprima("O veiculo de categoria ",categoria, " pagara pedagio= ",pedagio)
   fim enquanto
   FINALIZA:
Fim.
```

Declaração de 3 variáveis: **pedagio**, **tm** e **categoria**

Variável **tm** recebe inicialmente o valor de 3,00. Ela representa a coluna **Preço base** da tabela com a regra de cálculo do pedágio no exemplo que passei anteriormente.

Pergunta ao usuário a categoria do veículo

Lê a categoria informada pelo usuário e armazena o valor dentro da variável **categoria**

Tem um erro aqui! O correto neste ponto seria testar se a categoria informada pelo usuário está entre o e 8.
Deveria ser escrito **se (categoria < o ou categoria > 8)** e não **se (categoria < o e categoria > 8)** para funcionar corretamente.

Para cada categoria possível, o algoritmo aplica a correta forma de cálculo e armazena o resultado do cálculo na variável **pedagio**

Retorna o valor total do pedágio calculado

Finaliza

Do jeito que está escrito, esse código não vai funcionar corretamente. Na linha **se (categoria < o e categoria > 8)** o teste lógico sempre vai dar FALSO. Afinal, não existe nenhum número que seja menor que o e maior que 8 simultaneamente, não é mesmo? O correto seria a instrução **se (categoria < o ou categoria > 8)**. Desta forma, se fosse informada uma categoria com valor 10, por exemplo, o teste lógico iria dar VERDADEIRO e o algoritmo iria pular direto para a instrução FINALIZA, sem calcular o pedágio. Esse seria o comportamento correto. Como o teste lógico sempre retorna FALSO, o algoritmo não será finalizado e continuará para as próximas instruções erroneamente.

Resposta: B

Módulos

Vocês viram no algoritmo anterior como ficou grande o código-fonte do algoritmo? Imagine um sistema complexo do mundo real, como um sistema operacional ou um simulador meteorológico. Pode chegar a ter centenas de milhares de linhas de código!

Para lidar com a crescente complexidade dos algoritmos, uma das estratégias mais usadas é a **modularização**: quebrar um problema maior em partes menores (**módulos** ou **sub-rotinas** ou **subprogramas**) e resolver essas partes menores, uma por vez. Além dessa vantagem, a criação de **módulos** em um algoritmo permite a reutilização de código.

Professor, como assim reutilização de código? 🤔

Suponha que você está desenvolvendo um sistema para um tribunal. Sempre que o juiz manda pagar uma indenização, ele estipula o *valor a ser pago* e a *data-base* em que esse valor foi definido. Obviamente, quem foi condenado a pagar a indenização não vai ter que fazê-lo imediatamente. Ele vai ter um prazo para isso. Só que, decorrido o prazo, tem a correção monetária né? O valor original da sentença tem que ser corrigido monetariamente. Vamos supor que a correção monetária seja 0,5 % ao mês.

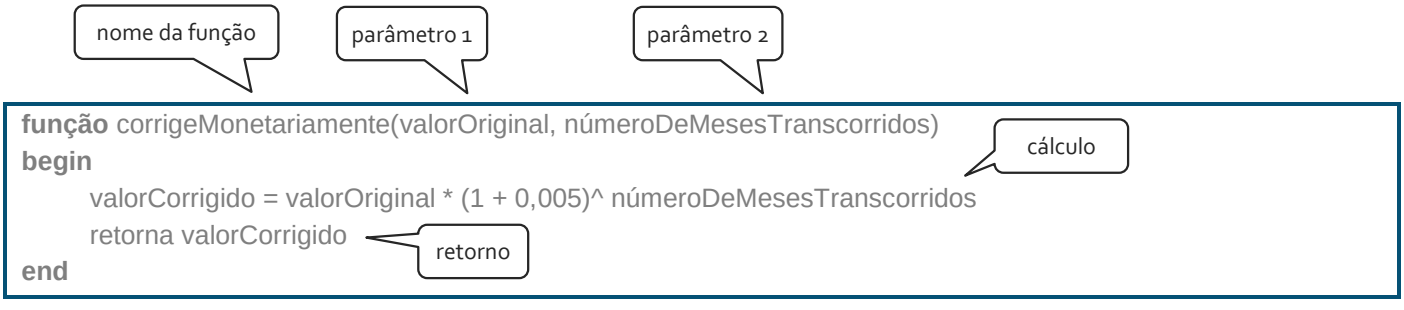
```
valorOriginal = 0
númeroDeMesesTranscorridos = 0
valorCorrigido = 0

leia(valor)
leia(númeroDeMesesTranscorridos)

valorCorrigido = valorOriginal * (1 + 0,005)^ númeroDeMesesTranscorridos
```

O algoritmo acima resolve o nosso problema. Dado o valorOriginal e númeroDeMesesTranscorridos ele calcula o valorCorrigido pela taxa de 0,5% ao mês.

Só que esse trecho de código faz parte de algo maior que é o sistema de gerenciamento dos processos do tribunal. No sistema, esse cálculo de corrigir monetariamente um valor vai ser necessário em diversos momentos.

Não é viável você sair replicando essas linhas de código no meio do sistema sempre que for preciso fazer correção monetária. Além disso, se a taxa deixar de ser 0,5% ao mês e passar para 0,6%? Você teria que sair "caçando" o 0,5% e substituindo por 0,6%. Grande chance de esquecer alguma ocorrência.

A solução ideal nesse caso seria criar uma função cujo único objetivo é fazer o cálculo da correção monetária.



Com a função declarada, você pode chamá-la quantas vezes quiser no seu algoritmo.

```
01  valorOriginal = 1000.00
02  númeroDeMesesTranscorridos = 2      1000.00          2
03
04  valorCorrigido= corrigeMonetariamente(valorOriginal, númeroDeMesesTranscorridos)
05  escreva(valorCorrigido)
06
07
08  valorOriginal = 2500.00
09  númeroDeMesesTranscorridos = 7      2500.00          7
10
11  valorCorrigido= corrigeMonetariamente(valorOriginal, númeroDeMesesTranscorridos)
12  escreva(valorCorrigido)
```

No pseudocódigo acima, a função é invocada pela 1ª vez na linha 04 recebendo como parâmetros valorOriginal = 1000.00 e númeroDeMesesTranscorridos = 2. Na mesma linha 04, a variável valorCorrigido receberá o resultado do cálculo da correção monetária para esses parâmetros.

Na linha 11, a função é invocada pela 2ª vez. Agora com os parâmetros valorOriginal = 2500.00 e númeroDeMesesTranscorridos = 7.

Há 2 tipos de **módulos**: as **funções**, que acabamos de ver, e os **procedimentos**.

Funções

Uma **função** (ou **function**) é um módulo de código delimitado que recebe **parâmetros**, realiza um **processamento** e retorna um **resultado** obrigatoriamente. Os parâmetros não são obrigatórios. Uma função pode ser criada recebendo nenhum parâmetro. Já o retorno é obrigatório.

```
função nomeDaFunção(parâmetro1, parâmetro2, ..., parâmetroN)
begin
    <bloco de instruções>
    Retorna <alguma coisa>
end
```

Procedimentos

Procedimentos (ou **procedures**) são módulos de código semelhantes às funções pois recebem parâmetros e fazem processamentos. A única diferença é que o **procedimento** não retorna nada.

```
procedimento nomeDoProcedimento(parâmetro1, parâmetro2, ..., parâmetroN)
begin
    <bloco de instruções>
end
```

Resumindo:

	Função	Procedimento
Recebe parâmetros?	Sim, mas não é obrigatório	Sim, mas não é obrigatório
Tem retorno?	Sim	Não

Há 2 formas de passar parâmetros para uma função (ou procedimento). Por **valor** ou por **referência**.

A **passagem por valor** cria uma cópia da variável que será válida somente dentro do contexto da função/procedimento. Assim, caso o valor do parâmetro seja alterado dentro da função, o que vai ser alterado é a cópia. O valor do parâmetro original (fora do contexto da função) não será alterado.

Por outro lado, a **passagem por referência** não cria cópia do parâmetro para ser manipulado dentro da função/procedimento. Passa somente um ponteiro para a região de memória da variável. Desta forma, caso o parâmetro seja alterado dentro da função, essa alteração será visível fora do contexto da função.

(CESPE - 2016 - TCE-PA - Auxiliar Técnico de Controle Externo - Área Informática)

Acerca de funções e procedimentos em subprogramas, julgue o item que se segue.

A passagem de parâmetro em uma rotina pode ocorrer de duas maneiras: por valor ou por referência. Em se tratando da passagem por valor, alteram-se os valores dos parâmetros que foram passados para a função.

RESOLUÇÃO:

Na passagem por valor, os valores dos parâmetros que foram passados para a função não se alteram porque são criadas cópias dos valores para serem manipulados dentro do contexto da rotina.

Resposta: Errado

Recursividade

A **recursividade** é uma técnica de programação em que uma função chama a si mesma com o objetivo de resolver um problema.

Professor, uma função pode fazer uma chamada a si mesma?!

Pode sim! Pense naquela boneca russa que, ao ser aberta, tem uma boneca menor dentro. O nome dessa boneca é Matryoshka.

Se você tivesse que escrever um algoritmo para contar o número total de bonecas dentro de uma Matryoshka, poderia resolver o problema por meio de uma função recursiva da seguinte forma.

```
01  função contaMatryoshka(Boneca boneca)
02  begin
03     se (boneca.interior() <> Vazio) então
04              retorna 1 + contaMatryoshka(boneca.interior())
05     senão
06              retorna 1
07  end
```

Repare que a função contaMatryoshka faz uma chamada ela mesmo na linha 04.

Ao escrever uma função recursiva, um erro comum é entrar em um **loop infinito**. Pense bem, se a função chama a ela mesma, esse processo pode se repetir indefinidamente até estourar a memória do computador, concorda?

Para evitar que isso aconteça, dentro da função recursiva é necessário definir uma **solução trivial**. Essa solução trivial (ou **caso-base** ou **condição de saída**) é o "freio" que impede que a recursividade entre em loop infinito. Na nossa função contaMatryoshka a solução trivial é o ***senão retorna*** **0** das linhas 05 e 06. A função recursiva recai nesse caso trivial quando chega na última boneca, que não tem nenhuma interior a ela. Ao chegar no caso trivial, as chamadas recursivas cessam e o algoritmo é encerrado retornando a solução do problema.

Existem diversos problemas matemáticos que podem facilmente ser resolvidos com a técnica de recursividade.

O contrário da **recursividade** é a **iteratividade**. Resolver um problema iterativamente significa solucioná-lo sem utilizar funções recursivas. A vantagem da recursividade é a facilidade para solucionar o problema. A desvantagem é que chamar funções recursivamente consome muita memória. Alternativamente, as soluções iterativas são mais difíceis de implementar, mas consomem menos memória, isto é, são mais eficientes.

	Recursividade	**Iteratividade**
Vantagem	Facilidade da solução	Consome pouca memória
Desvantagem	Consome muita memória	Dificuldade da solução

Vamos exercitar o conceito de recursividade com a seguinte questão.

(IF-PR - 2010 - IF-PR - Técnico de Tecnologia da Informação) Considere as seguintes afirmativas:

1) O cálculo do fatorial de um número (n!=n*(n-1)*(n-2)*...*1) só pode ser obtido com o uso de recursão.

2) Procedimentos recursivos aceitam parâmetros passados por referência.

3) Ao ser executado, o procedimento abaixo indica corretamente o valor do fatorial para qualquer "n" maior do que zero.

```
função Fat ( n ) {
  se n=2, então retorna 2;
       senão retorna n*Fat(n-1);
}
```

Assinale a alternativa correta.

(a) Somente a afirmativa 1 é verdadeira.

(b) Somente a afirmativa 2 é verdadeira.

(c) Somente a afirmativa 3 é verdadeira.

(d) Somente as afirmativas 1 e 3 são verdadeiras.

(e) As afirmativas 1, 2 e 3 são verdadeiras.

RESOLUÇÃO:

O cálculo do fatorial é o exemplo clássico da literatura quando se fala de recursividade.

Relembrando a definição de fatorial: n! = n*(n-1)*(n-2)*...*3*2*1.

Ou seja, fatorial de n é o produtório de todos os números naturais entre 1 e n.

Exemplo: 6! = 6*5*4*3*2*1 = 720

Vamos analisar as afirmativas.

1) O cálculo do fatorial de um número (n!=n(n-1)*(n-2)*...*1) só pode ser obtido com o uso de recursão.*

Isso é falso galera. O cálculo do fatorial também pode ser feito iterativamente. Não é obrigado a usar a recursão.

2) Procedimentos recursivos aceitam parâmetros passados por referência.

Certo. Um procedimento pode receber parâmetros passados por referência ou por valor, independentemente do fato de o procedimento ser recursivo ou não. Uma coisa não tem nada a ver com a outra.

3) Ao ser executado, o procedimento abaixo indica corretamente o valor do fatorial para qualquer "n" maior do que zero.

```
função Fat ( n ) {
  se n=2, então retorna 2;
        senão retorna n*Fat(n-1);
}
```

Errado. Essa função Fat(n) não calcula corretamente o fatorial quando n = 1. Para todos os outros valores de n maiores do que 1, ela calcula corretamente.

Resposta: B

A solução recursiva correta para calcular o fatorial de qualquer número inteiro maior que zero é a seguinte.

```
função Fat(n) {
      se n = 1 então retorna 1
      senão retorna n*Fat(n-1)
}
```

Teste de mesa

Para analisar um algoritmo passo a passo, é recomendável a realização do chamado teste de mesa, também chamado de teste chinês. É como executar um programa no papel, mantendo uma tabela com os valores das variáveis e registrando as mudanças nestas variáveis ao longo da execução do mesmo. Assim fica mais fácil acompanhar a execução do algoritmo e não se perder nas contas.

Nos exercícios, teremos a oportunidade de fazer e acompanhar vários testes de mesa.

Questões comentadas pelo professor

1. (CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o próximo item, concernente ao conceito relacionado a algoritmos e linguagens de programação.

Em um algoritmo, uma constante é um espaço físico na memória, e é identificada por um nome que não sofre alteração durante a execução do programa.

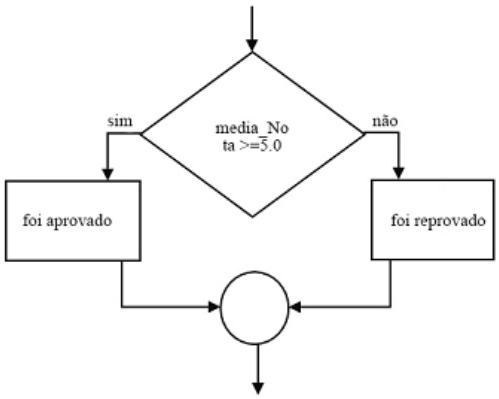
RESOLUÇÃO:

Questão bem básica sobre o conceito de constante. Como vimos, a constante é sim um espaço físico na memória e é identificada por um nome. Além disso, o valor da constante não sofre alteração durante a execução do programa.

Resposta: Certo

---

2. (CESPE - 2017 - TRT - 7ª Região (CE) - Analista Judiciário - Tecnologia da Informação)



A estrutura lógica presente no diagrama apresentado é do tipo

(a) SE ENTÃO.
(b) CASO SELECIONE.
(c) CASO REPITA
(d) SE ENTÃO SENÃO.

RESOLUÇÃO:

O fluxograma tem um teste de condição (media_Nota >= 5.0) com 2 saídas: sim para "foi aprovado" e não para "foi reprovado". Se tivesse só a saída sim, seria uma estrutura do tipo SE ENTÃO. Como tem também a saída não, que representa o SENÃO, a estrutura é do tipo SE ENTÃO SENÃO.

Resposta: D

---

3. (FCC - 2018 - SABESP - Analista de Gestão - Sistemas)

De acordo com dados da SABESP, um pequeno buraco de 2 milímetros no encanamento desperdiça 3,2 mil litros de água em um dia. Um Analista escreveu o algoritmo em pseudocódigo abaixo para calcular o desperdício de água em função de buracos em encanamentos.

```
Var largburaco, desperdicio: real
       dias: inteiro
Início
         imprima("Digite a largura do buraco em milímetros: ")
         leia(largburaco)
         imprima("Digite o número de dias do vazamento: ")
         leia(dias)
         se  I
                  então
                       II
                       imprima("Em ", dias, " dias foram desperdiçados ", desperdicio,
                       " mil litros de água ")
                  senão   imprima("Dado(s) inválido(s)")
         fimse
Fim
```

O comando que preenche corretamente a lacuna

(a) **I** é: (largburaco > 0 ou dias > 0)
(b) **I** é: (largburaco > 0 e desperdicio > 0)
(c) **II** é: desperdicio ← (largburaco/2.0) * 3.2 * dias
(d) **II** é: desperdicio ← desperdicio + (largburaco/2.0) * 3.2
(e) **II** é: desperdicio ← (largburaco/3.2) * 2.0 * dias

RESOLUÇÃO:

O que se encaixaria corretamente na lacuna I seria um teste para garantir que as 2 variáveis necessárias para calcular o desperdício (largburaco e dias) são maiores que zero. Não faz sentido calcular desperdício se alguma delas for zero. Esse teste seria (largburaco > 0 e dias > 0), com o operador **e** lógico. Não confundir com a alternativa (a) que usa o **ou** lógico.

Portanto, nem a alternativa (a) nem a (b) se encaixam corretamente na lacuna I.

Na lacuna II se encaixa o cálculo do desperdício. Pelo enunciado da questão, a fórmula correta é

desperdicio ← (largburaco/2.0) * 3.2 * dias

Resposta: C

4. (CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 8)

Julgue o item subsequente, relativo à lógica de programação.

Na passagem de parâmetro por referência, é possível alterar o valor da variável que é apontada por referência.

RESOLUÇÃO:

Correto. Na passagem de parâmetro por referência, o que é passado para a função é uma referência à região de memória onde se encontra a variável. Com essa referência é possível alterar o valor da variável.

Resposta: Certo

---

5. (VUNESP - 2016 - MPE-SP - Analista Técnico Científico - Engenheiro de Computação)

No contexto de passagem de parâmetros para uma sub-rotina, existe a denominada passagem de parâmetro por valor. Nesse caso,

(a) o parâmetro pode ser passado para a sub-rotina, desde que ela seja uma sub-rotina de tratamento de interrupção.
(b) o endereço onde se encontra o valor a ser passado como parâmetro é fornecido para a sub-rotina.
(c) um ponteiro para o endereço onde se encontra o valor a ser passado como parâmetro é fornecido para a sub-rotina.
(d) um registrador que aponta para o valor a ser passado como parâmetro é fornecido para a sub-rotina.
(e) uma cópia do valor do parâmetro é fornecida para a sub-rotina.

RESOLUÇÃO:

O examinador está questionando a respeito da passagem de parâmetro por valor. Vamos lá.

A alternativa (a) está errada. Não existe obrigatoriedade de a sub-rotina ser de tratamento de interrupção.

As alternativas (b) e (c) referem-se à passagem por referência.

A (d) está errada. No nível de pseudocódigo não se manipula registrador. Isso é uma estrutura no nível de máquina.

A (e) está correta. Na passagem por valor, é feita uma cópia do valor do parâmetro que é passada para a sub-rotina.

Resposta: E

---

6. (FUMARC - 2013 - TJM-MG - Oficial Judiciário - Assistente Técnico de Manutenção de Informática)

Em relação aos conceitos de lógica de programação, analise as seguintes afirmativas:

(I) Uma constante é um determinado valor fixo que não se modifica durante a execução de um programa.
(II) Uma variável está associada a uma posição de memória, cujo conteúdo pode ser modificado durante a execução do programa.
(III) Toda variável é identificada por uma constante

Estão CORRETAS as afirmativas:

(a) I e II, apenas.
(b) I e III, apenas.
(c) II e III, apenas.
(d) I, II e III.

RESOLUÇÃO:

A (I) está correta. Essa é a definição de constante.

(II) Correta. É a definição de variável.

(III) Errada. A variável é identificada por um nome!

Resposta: A

7. (CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o próximo item, concernentes aos conceitos relacionados a algoritmos e linguagens de programação.

A resposta do algoritmo seguinte é 8.

```
função pfactor(num) {
    if (num <= 1) return 1;
    return pfactor(num-1) + pfactor(num-2);
}
x=5;
factors = pfactor(x);
escreva(factors);
```

RESOLUÇÃO:

Em matemática existe uma sequência clássica chamada Sequência de Fibonacci. É uma sucessão infinita de números que obedecem um padrão em que cada elemento é a soma dos dois anteriores. Observe a sequência:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584...

O 3º elemento é a soma do 1º e do 2º: 3 = 1 + 2

O 4º elemento é a soma do 2º e do 3º: 5 = 2 + 3

O 5º elemento é a soma do 3º e do 4º: 8 = 3 + 5

E assim sucessivamente.

Como vimos, o valor do enésimo elemento da sequência é a soma dos 2 anteriores.

Podemos fazer uma fórmula genérica para o enésimo elemento da sequência $\boldsymbol{F(n) = F(n-1) + F(n-2)}$

O algoritmo do enunciado da questão é exatamente a implementação da sequência de Fibonacci.

A função **pfactor** calcula o enésimo termo da sequência de forma recursiva.

No enunciado, após declarar essa função, o algoritmo manda calcular **pfactor(5)**, isto é o 5º elemento da sequência de Fibonacci, que é 8.

Resposta: Certo

8. (CONSULPAM - 2018 - Câmara de Juiz de Fora - MG - Assistente Legislativo – Técnico em Informática)

Analise os itens abaixo que versam sobre Lógica de Programação e depois responda:

(I) Lógica de programação é o modo como se escreve um programa de computador, um algoritmo. Um algoritmo é uma sequência de passos para se executar uma função.

(II) A linguagem de programação é como uma língua normal, um grupo de palavras com significados. No caso da programação, a maioria das linguagens é escrita em Inglês. Estas linguagens fazem o computador assimilar cada comando e função de um algoritmo, depois executar cada função.

(III) Na hora de programar alguns passos são indispensáveis, como Declarar Variáveis. Variáveis são escritas exclusivamente por letras, que representam um valor que pode ser mudado a qualquer momento.

(IV) Saber lógica de programação é saber o melhor jeito de escrever um código, para o computador interpretar corretamente. É saber se comunicar com a máquina a partir de uma linguagem seja lá qual for.

Analisados os itens é CORRETO afirmar que:

(a) Todos os itens estão corretos.
(b) Apenas o item IV está incorreto.
(c) Apenas o item III está incorreto.
(d) Conjunto finito de passos.

RESOLUÇÃO:

(I) Correto.

(II) Correto

(III) Errado. O Erro ocorre quando o examinador afirma que variáveis são escritas exclusivamente por letras. Na verdade, o nome de uma variável deve começar por uma letra mas pode conter também números. Exemplos de nomes de variáveis válidos: nome1, contador123, etc.

(IV) Correto

Resposta: C

9. (CONSULPAM - 2018 - Câmara de Juiz de Fora - MG - Assistente Legislativo – Técnico em Informática)

Algoritmo é uma sequência finita e bem definida de passos que, quando executados, realizam uma tarefa específica ou resolvem um problema. NÃO é uma das propriedades do algoritmo:

(a) Composto por ações complexas e por vezes indefinidas.
(b) Composto por ações simples e bem definidas (não pode haver ambiguidade, ou seja, cada instrução representa uma ação que deve ser entendida e realizada).
(c) Sequência ordenada de ações.
(d) Conjunto finito de passos.

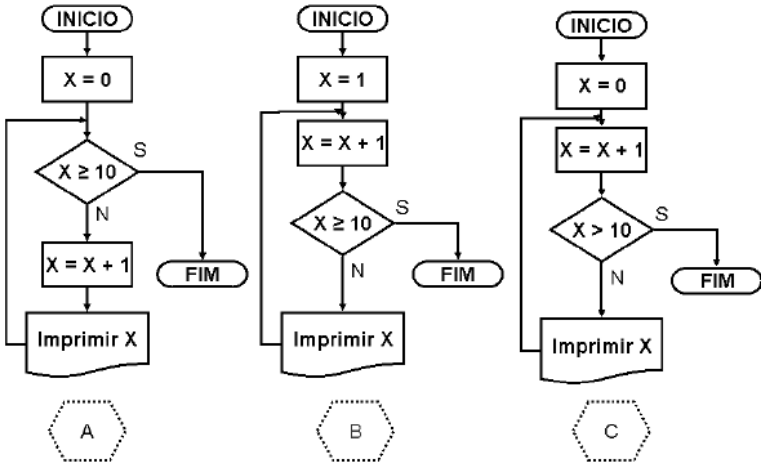
RESOLUÇÃO:

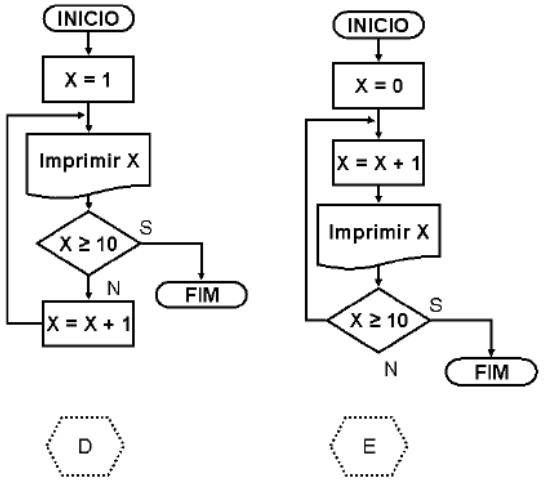
(a) Errado. As ações de um algoritmo até podem ser complexas, mas não podem ser indefinidas. Quem trabalha com coisas indefinidas (subjetivas) somos nós humanos. Algoritmo só trabalha com coisas bem definidas (objetivas).
(b) Correto
(c) Correto
(d) Correto. A memória do computador é finita. Então, o conjunto de passos do algoritmo também tem que ser finito.

Resposta: A

10. (FAURGS - 2018 - TJ-RS - Programador)

Considere os cinco fluxogramas abaixo, identificados pelas letras A, B, C, D e E, que geram valores da variável X e imprimem esses valores dentro de uma faixa controlada.





O objetivo dos fluxogramas é imprimir valores de X, na faixa de 1 a 10 (incluindo os limites). Porém um dos fluxogramas imprime valor(es) fora desta faixa. Qual é este fluxograma?

RESOLUÇÃO:

Todos os algoritmos imprimem corretamente os números de 1 a 10 exceto o da letra B, que imprime de 2 a 10.

Resposta: B

11.(CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o item a seguir, relativo ao conceito de construção de algoritmos.

O algoritmo a seguir apresenta um exemplo de busca sequencial.

```
var a:vetor[1..5] de inteiro;
    temp:inteiro;
    i,k:inteiro;
inicio
    a[1]:=57;
    a[2]:=58;
    a[3]:=60;
    a[4]:=56;
    a[5]:=59;
    para i:=1 até 4 faça inicio
        para k:=i+1 até 5 faça inicio
            se a[i] > a[k] então inicio
               temp:=a[i];
               a[i]:=a[k];
               a[k]:=temp;
            fim se;
        fim para;
    fim para;
    escreva('Resultado: ');
    para i:=1 até 5 faça inicio
        escreva(a[i],',');
    fim para;
fim;
```

RESOLUÇÃO:

Vamos entender passo a passo o que este algoritmo faz.

```
var a:vetor[1..5] de inteiro;
    temp:inteiro;
    i,k:inteiro;
inicio
    a[1]:=57;
    a[2]:=58;
    a[3]:=60;
    a[4]:=56;
    a[5]:=59;
    para i:=1 até 4 faça inicio
        para k:=i+1 até 5 faça inicio
            se a[i] > a[k] então inicio
                temp:=a[i];
                a[i]:=a[k];
                a[k]:=temp;
            fim se;
        fim para;
    fim para;
    escreva('Resultado: ');
    para i:=1 até 5 faça inicio
        escreva(a[i],',');
    fim para;
fim;
```

Declaração de um vetor de inteiros com 5 posições de nome **a**

Declaração de 3 variáveis auxiliares do tipo inteiro: **temp**, **i** e **k**

Preenche as 5 posições do vetor **a** de inteiros

57	58	60	56	59

Percorre os elementos do vetor **a** com os índices **i** e **k**, sendo que **k** está sempre à direita de **i**.

Quando encontra um valor no índice **k** menor que no índice **i**, troca os elementos de posição para que o menor fique à esquerda.

Imprime o resultado após manipulação dos elementos do vetor

Ao percorrer o vetor e rearranjando seus elementos para que o menor esteja sempre à esquerda, o algoritmo em questão está ordenando os elementos do vetor na ordem crescente. Esse algoritmo tem o nome de Bubble Sort.

Portanto, trata-se de um algoritmo de ordenação e ***não*** de busca sequencial como afirma o enunciado.

Resposta: Errado

12. (IF-MT - 2018 - IF-MT - Informática)

Analise o trecho do algoritmo abaixo representado em português estruturado:

```
COMECO < - 1
        FINAL < - 10
        ACHA <- falso
        enquanto (COMECO <= FINAL) e (ACHA = falso) faca
                MEIO <- (COMECO + FINAL) div 2
                se (PESQ = NOME[MEIO]) entao
                        ACHA < - verdadeiro
                senao
                        se (PESQ < NOME[MEIO]) entao
                                FINAL <- MEIO - 1
                        senao
                                COMECO <- MEIO + 1
                        fimse
                fimse
        fimenquanto
```

É correto afirmar que:

(a) Trata-se do trecho de um algoritmo de pesquisa binária que só é possível ser empregada se os dados estiverem ordenados no sistema FIFO.
(b) Trata-se do trecho de um algoritmo de pesquisa sequencial que deve ter ordenado o vetor anteriormente para que pudesse ser empregada.
(c) Trata-se de um algoritmo de ordenação do tipo bubble sort onde é enviado ao fim da fila o maior elemento da lista.
(d) Trata-se de um trecho de algoritmo de pesquisa binária em um vetor que deve estar previamente ordenado em ordem crescente para seu emprego.
(e) Trata-se de um algoritmo de ordenação sequencial para que posteriormente seja possível pesquisar no vetor Nome.

RESOLUÇÃO:

Analisando o funcionamento do algoritmo em português estruturado.

```
COMECO <- 1
FINAL <- - 10
ACHA <- falso
enquanto (COMECO <= FINAL) e (ACHA = falso) faca
    MEIO <- (COMECO + FINAL) div 2
    se (PESQ = NOME[MEIO]) entao
        ACHA <- - verdadeiro
    senao
        se (PESQ < NOME[MEIO]) entao
            FINAL <- MEIO - 1
        senao
            COMECO <- MEIO + 1
        fimse
    fimse
fimenquanto
```

Declaração e inicialização de 3 variáveis:
COMECO para identificar a posição inicial do trecho de pesquisa do vetor.
FINAL para identificar a posição final do trecho de pesquisa do vetor
ACHA é um booleano que mudará para verdadeiro quando o termo **PESQ** for encontrado dentro do vetor **NOME**

Percorre o vetor do início ao fim

Coloca o índice **MEIO** na mediana do vetor

Verifica se o termo pesquisado **PESQ** encontra-se no **MEIO** do vetor **NOME**

Se achar, o objetivo foi atingido e a pesquisa termina. Sai do laço **enquanto** porque **ACHA** = **verdadeiro**

Se não achar, continua procurando em um subvetor de **NOME**

Escolhe o subvetor da esquerda porque o termo procurado **PESQ** é menor que o que se encontra no **MEIO**

Escolhe o subvetor da direita porque o termo procurado **PESQ** é maior que o que se encontra no **MEIO**

Este algoritmo é um clássico da computação chamado de pesquisa binária. Ele é um pouco mais esperto que a pesquisa sequencial. Na pesquisa sequencial, o algoritmo simplesmente percorre todas as posições do vetor, da 1ª à última, para ver se encontra o termo pesquisado.

O algoritmo do enunciado, a pesquisa binária, é mais inteligente. Em vez de percorrer todo o vetor, ele vai direto na posição do meio e verifica se o termo procurado está lá. Se estiver acabou. O termo foi encontrado. Se não, ele descarta metade do vetor e continua procurando na outra metade. A metade que ele vai escolher pode ser a direita ou esquerda, dependendo se o termo procurado é menor ou maior que o valor que se encontra no meio do vetor, respectivamente.

Quando ele passar para a metade escolhida, a estratégia vai ser a mesma. Ele vai achar o valor da posição mediana dessa metade e caso o termo procurado não esteja lá, quebra a metade em 2 partes e escolhe uma dessas partes para prosseguir na busca.

Essa estratégia da pesquisa binária é bem mais eficiente que a pesquisa sequencial só que a binária só funciona se o vetor a ser pesquisado for ordenado. Já a pesquisa sequencial funciona mesmo para vetores desordenados.

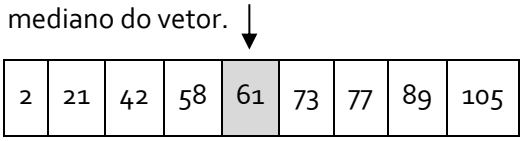
Para facilitar o entendimento, vamos visualizar o funcionamento da pesquisa binária com um caso concreto.

Seja o seguinte vetor ordenado de inteiros.

2	21	42	58	61	73	77	89	105

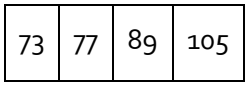
Vamos aplicar o algoritmo de pesquisa binária neste vetor para achar em que posição se encontra o valor 89.

O passo 1 é achar o elemento mediano do vetor.

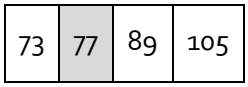


O valor procurado (89) está na mediana? Não.

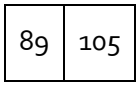
Então, precisamos escolher uma metade do vetor para prosseguir na busca. Pela regra do algoritmo, iremos escolher a metade direita porque 89 > 61. Ficamos agora com o seguinte subvetor.



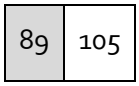
Vamos localizar o ponto mediano.



O valor procurado (89) está na mediana? Não. 89 é maior que a mediana. Então, descartamos o subvetor da esquerda e prosseguimos com a pesquisa no subvetor direito.



Vamos localizar o ponto mediano desse subvetor.



O valor procurado (89) está na mediana? Sim. Pronto! Problema resolvido. O algoritmo retorna ACHA = verdadeiro que significa valor encontrado.

Agora que você entendeu o que é pesquisa binária, vamos voltar à questão do concurso e analisar as alternativas.

(a) Trata-se do trecho de um algoritmo de pesquisa binária que só é possível ser empregada se os dados estiverem ordenados no sistema FIFO.

Errado. Não há essa obrigatoriedade de ordenação FIFO (*first in, first out*). Bastava estar ordenado em ordem crescente.

(b) Trata-se do trecho de um algoritmo de pesquisa sequencial que deve ter ordenado o vetor anteriormente para que pudesse ser empregada.

O algoritmo descrito no enunciado é uma pesquisa binária e não uma pesquisa sequencial. Errado.

(c) Trata-se de um algoritmo de ordenação do tipo *bubble sort* onde é enviado ao fim da fila o maior elemento da lista.

Errado. *Bubble sort* é o algoritmo da questão anterior (questão 11). Nesta questão, trata-se de uma pesquisa binária.

(d) Trata-se de um trecho de algoritmo de pesquisa binária em um vetor que deve estar previamente ordenado em ordem crescente para seu emprego.

Correto!

(e) Trata-se de um algoritmo de ordenação sequencial para que posteriormente seja possível pesquisar no vetor Nome.

Errado. Não é pesquisa sequencial.

Resposta: D

13.(CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 8)

Considere o pseudo-código abaixo:

```
F0 = 0
F1 = 1
I = 0
PARA I DE 1 ATÉ 10, FAÇA
    T = F1
    F1 = F1 + F0
    F0 = T
FIM PARA
MOSTRE F1
```

O valor da variável F1 exibido é

(a) 54
(b) 33
(c) 89
(d) 93
(e) 141

RESOLUÇÃO:

As 3 instruções dentro da estrutura de repetição PARA compõem o bloco que será executado repetidamente. Esse loop é do tipo repetição com variável de controle onde I é a variável de controle.

Para reproduzir os cálculos realizados pelo algoritmo vamos montar um teste de mesa para observar o valor das variáveis em cada ponto da execução do código.

Nas 3 primeiras linhas de código as variáveis Fo, F1 e I são inicializadas como Fo = 0, F1 = 1 e I = 0. Portanto, o estado inicial das variáveis é o seguinte.

I	0									
T										
F1	1									
F0	0									

Na linha 4 ele entra na estrutura condicional PARA usando I como variável de controle. Serão feitas 10 repetições com esse PARA, com I variando de 1 a 10.

Na **1ª iteração** (I = 1) ele executa as 3 instruções

```
T = F1 = 1
F1 = F1 + F0 = 1 + 0 = 1
F0 = T = 1
```

Ao final do 1º loop, nossa mesa será a seguinte.

I	0	1								
T		1								
F1	1	1								
F0	0	1								

Na **2ª iteração** (I = 2) ele executa as 3 instruções

T = F1 = 1
F1 = F1 + F0 = 1 + 1 = 2
F0 = T = 1

Ao final do 2º loop, nossa mesa será a seguinte.

I	0	1	2							
T		1	1							
F1	1	1	2							
F0	0	1	1							

Na **3ª iteração** (I = 3) ele executa as 3 instruções

T = F1 = 2
F1 = F1 + F0 = 2 + 1 = 3
F0 = T = 2

Ao final do 3º loop, nossa mesa será a seguinte.

I	0	1	2	3						
T		1	1	2						
F1	1	1	2	3						
F0	0	1	1	2						

Na **4ª iteração** (I = 4) ele executa as 3 instruções

T = F1 = 3
F1 = F1 + F0 = 3 + 2 = 5
F0 = T = 3

Ao final do 4º loop, nossa mesa será a seguinte.

I	0	1	2	3	4					
T		1	1	2	3					
F1	1	1	2	3	5					
F0	0	1	1	2	3					

Na **5ª iteração** (I = 5) ele executa as 3 instruções

T = F1 = 5
F1 = F1 + F0 = 5 + 3 = 8
F0 = T = 5

Ao final do 5º loop, nossa mesa será a seguinte.

I	0	1	2	3	4	5				
T		1	1	2	3	5				
F1	1	1	2	3	5	8				
F0	0	1	1	2	3	5				

Na **6ª iteração** (I = 6) ele executa as 3 instruções

T = F1 = 8
F1 = F1 + F0 = 8 + 5 = 13
F0 = T = 8

Ao final do 6º loop, nossa mesa será a seguinte.

I	0	1	2	3	4	5	6			
T		1	1	2	3	5	8			
F1	1	1	2	3	5	8	13			
F0	0	1	1	2	3	5	8			

Na **7ª iteração** (I = 7) ele executa as 3 instruções

T = F1 = 13
F1 = F1 + F0 = 13 + 8 = 21
F0 = T = 13

Ao final do 7º loop, nossa mesa será a seguinte.

I	0	1	2	3	4	5	6	7		
T		1	1	2	3	5	8	13		
F1	1	1	2	3	5	8	13	21		
F0	0	1	1	2	3	5	8	13		

Na **8ª iteração** (I = 8) ele executa as 3 instruções

T = F1 = 21
F1 = F1 + F0 = 21 + 13 = 34
F0 = T = 21

Ao final do 8º loop, nossa mesa será a seguinte.

I	0	1	2	3	4	5	6	7	8	
T		1	1	2	3	5	8	13	21	
F1	1	1	2	3	5	8	13	21	34	
F0	0	1	1	2	3	5	8	13	21	

Na **9ª iteração** (I = 9) ele executa as 3 instruções

T = F1 = 34
F1 = F1 + F0 = 34 + 21 = 55
F0 = T = 34

Ao final do 9º loop, nossa mesa será a seguinte.

I	0	1	2	3	4	5	6	7	8	9
T		1	1	2	3	5	8	13	21	34
F1	1	1	2	3	5	8	13	21	34	55
F0	0	1	1	2	3	5	8	13	21	34

Na **10ª iteração** (I = 10) ele executa as 3 instruções

T = F1 = 55
F1 = F1 + F0 = 55 + 34 = 89
F0 = T = 55

Ao final do 10º loop, nossa mesa será a seguinte.

I	0	1	2	3	4	5	6	7	8	9	10
T		1	1	2	3	5	8	13	21	34	55
F1	1	1	2	3	5	8	13	21	34	55	**89**
F0	0	1	1	2	3	5	8	13	21	34	55

Após 10 loops chega do fim a estrutura de repetição. O valor de F1 após as 10 iterações é **89**. Resposta letra C.

Obs: Esse algoritmo é a clássica sequência de Fibonacci. Falamos com detalhe sobre essa sequência especial na questão 7. Dê uma olhada lá.

Resposta: C

14. (SUGEP - UFRPE - 2018 - UFRPE - Técnico de Tecnologia da Informação - Sistemas)

Considere a função recursiva 'func' definida por

func(1) = 1
func(n) = (n - 1) * func(n - 1)

Quais são os valores de func(4) e func(5), respectivamente?

(a) 24 e 120
(b) 12 e 24
(c) 6 e 24
(d) 1 e 2
(e) 2 e 6

RESOLUÇÃO:

Vamos matar esta no peito por meio de debug.

func(1) = 1 , por definição do enunciado

func(2) = (2 – 1)*func(2 – 1) = func(1) = 1

func(3) = (3 – 1)*func(3 – 1) = 2*func(2) = 2*1 = 2

func(4) = (4 – 1)*func(4 – 1) = 3*func(3) = 3*2 = 6

func(5) = (5 – 1)*func(5 – 1) = 4*func(4) = 4*6 = 24

Portanto, func(4) = 6 e func(5) = 24.

Obs: A func(n), na forma como foi definida pelo avaliador, nada mais é que (n-1)!, isto é, fatorial de (n-1).

Resposta: C

15.(SUGEP - UFRPE - 2018 - UFRPE - Técnico de Tecnologia da Informação - Sistemas)

Considere o algoritmo a seguir.

```
Inteiro x1 =2, x2 = -1, x3 = 4
Enquanto (x1 > 0) faça
    x2 = x3/3-x2*4
    x1 = x3 % x1
Fim enquanto
Imprime(x2)
```

O que será impresso ao final do programa?

(a) 0
(b) 5
(c) 10
(d) -1
(e) 4

RESOLUÇÃO:

O loop **Enquanto-faça** é a estrutura de repetição pré-testada que estudamos.

Uma boa estratégia para resolver questões desse tipo é construir uma tabela auxiliar com o estado das variáveis. Na hora da prova, você pode rabiscar a tabela e resolver da mesma forma. Exercite para conseguir fazer isso certo e rápido. Concurso é um teste de rapidez!

	x1	x2	x3
Estado inicial	2	-1	4
1ª repetição do loop	**4%2 = 0**	4/3 – (-1)*4 = 4/3 + 4 = 1,33... + 4 = 5,33... = 5	4

Opa! Já na 1ª repetição do loop, x1 fica igual a 0. Lembre-se que a operação 4%2 representa o resto da divisão inteira de 4 por 2, que é 0. Antes de entrar na 2ª repetição, o algoritmo irá fazer o pré-teste x1 > 0 e o teste dará FALSO porque x1 = 0. Isso ocasiona a saída do loop. Portanto, Imprime(x2) irá imprimir 5. Veja que o cálculo de x2 na tabela deu 5,33333..... Mas como x2 foi declarado como inteiro na 1ª linha, desprezamos a parte decimal e ficamos só com a parte inteira 5.

Resposta: B

16. (Quadrix - 2018 - CRM-PR - Técnico em Tecnologia da Informação)

A respeito de análise e desenvolvimento de sistemas, julgue o item subsequente.

Os algoritmos são sequências finitas de instruções que, quando corretamente executadas, levam à solução de um problema.

RESOLUÇÃO:

Correto. Questão elementar sobre a definição de algoritmo.

Resposta: Certo

---

17. (Quadrix - 2018 - CRM-PR - Técnico em Tecnologia da Informação)

A respeito de análise e desenvolvimento de sistemas, julgue o item subsequente.

Em um fluxograma, as caixas de decisão são como "caixas pretas", uma vez que não se tem clareza da ação que será executada.

RESOLUÇÃO:

O fluxograma não é uma "caixa preta". Pelo contrário, ele clarifica o entendimento do sistema.

Resposta: Errado

---

18. (FAURGS - 2018 - TJ-RS - Programador)

Instrução: A questão refere-se ao algoritmo abaixo, escrito em uma pseudolinguagem. Considere X um arranjo; length, uma função que devolve o tamanho do arranjo passado como parâmetro. A endentação demarca blocos de comandos.

```
1  for j=2 to length(X)
2      do      valor = X[ j ]
3              i = j-1
4              while i > 0 e X[ i ] > valor
5                      do      X[i+1] = X[ i ]
6                              i = i-1
7              X[i+1] = valor
```

Qual é a característica principal desse algoritmo?

(a) É baseado na utilização de recursividade.
(b) É controlado por comandos de desvio incondicional.
(c) Utiliza comandos de repetição.
(d) Pode permanecer em laço infinito.
(e) É baseado na abordagem dividir e conquistar.

RESOLUÇÃO:

(a) É baseado na utilização de recursividade.
Não. Para haver recursividade é preciso que exista uma declaração de função que invoque a ela mesma. Não há nenhuma função no trecho de código.

(b) É controlado por comandos de desvio incondicional.
Não ocorre desvio incondicional no código em questão. Desvio incondicional seria o comando GO TO, mandando o algoritmo pular para determinada linha, independentemente de qualquer condição.

(c) Utiliza comandos de repetição.
Correto. O pseudocódigo utiliza 2 estruturas de repetição: o **for** e o **while**.

(d) Pode permanecer em laço infinito.
O **for** repete enquanto a variável de controle j estiver no intervalo 2 e length(X). Como length(X) representa o tamanho do vetor X, é um número finito. Já o **while** repete enquanto i > 0 e X[i] > valor. Como i é decrementado a cada iteração do **while**, em algum momento ele irá deixar de atender a condição i > 0 fazendo com que o **while** seja terminado também. Portanto, tanto o **for** quanto o **while** são finitos.

(e) É baseado na abordagem dividir e conquistar.
Dividir para conquistar está mais ligado à modularização do código. Dividir o algoritmo maior em funções ou procedures menores para resolver problemas específicos. Errado.

Resposta: C

19. (FCC - 2016 - TRF - 3ª REGIÃO - Técnico Judiciário - Informática)

Em uma linguagem de programação, os tipos primitivos de dados:

(a) são sempre verificados pelo compilador. Caso se extrapole a capacidade do tipo, um erro ocorre e o programa é abortado.
(b) mais comuns e mais utilizados são as matrizes e os registros.
(c) são associados a um descritor. Um descritor é uma estrutura de dados, que não ocupa espaço na memória, que armazena os atributos do tipo de dados.
(d) na forma de caracteres geralmente são armazenados como codificações numéricas, como o padrão UTF.
(e) inteiros são sempre representados como uma cadeia de caracteres. O caracter mais à esquerda representa o sinal positivo ou negativo.

RESOLUÇÃO:

(a) são sempre verificados pelo compilador. Caso se extrapole a capacidade do tipo, um erro ocorre e o programa é abortado.
Nem sempre. O tipo primitivo só é verificado pelo compilador nas linguagens fortemente tipadas (exemplos C, C++, Java, etc.). Nas linguagens fracamente tipadas (ex. Javascript, R, etc.) o tipo só é verificado em tempo de execução.

(b) mais comuns e mais utilizados são as matrizes e os registros.
Errado. Matrizes e registros são tipos compostos.
(c) são associados a um descritor. Um descritor é uma estrutura de dados, que não ocupa espaço na memória, que armazena os atributos do tipo de dados.

Descritores são tipos compostos também.

(d) na forma de caracteres geralmente são armazenados como codificações numéricas, como o padrão UTF.
Correto. O padrão UTF associa a cada caractere um número e é esse número que é armazenado.

(e) inteiros são sempre representados como uma cadeia de caracteres. O caracter mais à esquerda representa o sinal positivo ou negativo.
Errado. Inteiro é um número sem parte fracionária. Cadeia de caracteres (strings) são vetores de dígitos.

Resposta: D

20. (FGV - 2015 - PGE-RO - Técnico da Procuradoria - Tecnologia da Informação)

Analise o pseudocódigo mostrado a seguir.

```
var i: inteiro
var j: inteiro
para i:= 1 até 2
begin
     if i < 2
     then k=i*2
     else k=i

     para j:= i até k
     begin
          print (i+j)
     end
end
```

Sabendo-se que nesse código cada ocorrência do comando print produz uma linha na saída, está correto afirmar que o número de linhas produzidas é:

(a) 3
(b) 5
(c) 7
(d) 9
(e) 11

RESOLUÇÃO:

Vamos resolver esta questão com uma tabela auxiliar para não nos perdermos nas contas.

Linha de código	i	j	k
para i:= 1 até 2	1		
i < 2 ? VERDADEIRO			
k=i*2	1		2
para j:= i até k	1	1	2
print (i+j) = 2	1	1	2
para j:= i até k	1	2	2
print (i+j) = 3	1	2	2
para i:= 1 até 2	2	2	2
i < 2 ? FALSO			
k=i	2	2	2
para j:= i até k	2	2	2
print (i+j) = 4			

```
var i: inteiro
var j: inteiro
para i:= 1 até 2
begin
     if i < 2
     then k=i*2
     else k=i

     para j:= i até k
     begin
         print (i+j)
     end
end
```

Portanto, temos 3 ocorrências de print, a 1ª retornando 2, a 2ª retornando 3 e a 3ª retornando 4.

Resposta: A

21. (FCC - 2015 - DPE-SP - Programador)

Considere o algoritmo em pseudocódigo no qual DIV calcula o quociente da divisão inteira e MOD o resto da divisão inteira:

```
Var  taxa, cinco, tres, quociente, resto: inteiro

Início
    imprima ("Digite um numero inteiro maior ou igual a 8: ")
    leia(taxa)
    quociente ← taxa DIV 5
    resto ← taxa MOD 5
    tres ← 0
    cinco ← 0
    caso resto seja
         0: cinco ← quociente
            tres ← 0
         1: cinco ← quociente -1
            tres ← 2
         2: cinco ← quociente -2
            tres ← 4
         3: cinco ← quociente
            tres ← 1
         4: cinco ← quociente -1
            tres ← 3
   fim caso
   imprima(taxa, cinco, tres)

Fim.
```

O algoritmo em pseudocódigo acima

(a) garante que o valor de entrada seja maior ou igual a 8 para que seja possível dividir a taxa por 5 e por 3.
(b) para o valor inicial da taxa = 22 finaliza com cinco= 2 e tres=4.
(c) determina o maior número de 5 e de 3 unidades cuja soma dá o valor da taxa.
(d) para o valor inicial da taxa = 17 finaliza com cinco= 3 e tres=2.
(e) sempre finaliza com valores da variável cinco maiores ou igual a 1, mas a variável tres pode ter valor 0.

RESOLUÇÃO:

(a) garante que o valor de entrada seja maior ou igual a 8 para que seja possível dividir a taxa por 5 e por 3.
Errado. Não há essa garantia no código. O fato de imprimir "Digite um número inteiro maior ou igual a 8:" não garante nada. É apenas uma mensagem informativa.

(b) para o valor inicial da taxa = 22 finaliza com cinco= 2 e tres=4.
Se **taxa** = 22, **quociente** = 22 div 5 = 4 e **resto** = 22 mod 5 = 2.
Assim, o **caso (resto)** vai cair na opção **2**.
Na opção 2 do caso, a variável **cinco** = **quociente** – 2 = 4 – 2 = **2**. E a variável **tres** = 4.
Portanto a alternativa está correta. No final do algoritmo, **cinco** = 2 e **tres** = 4. O nome das variáveis é só para confundir a cabeça.

(c) determina o maior número de 5 e de 3 unidades cuja soma dá o valor da taxa.
Errado. Isso não é feito.

(d) para o valor inicial da taxa = 17 finaliza com cinco= 3 e tres=2.

Se **taxa** = 17, **quociente** = 17 div 5 = 3. E **resto** = 17 mod 5 = 2. Se **resto** = 2, o **caso (resto)** recai no caso 2, onde a variável **cinco** = quociente – 2 = 3 – 2 = 1. E a variável **tres** = 4. Resumindo, no final do algoritmo, **cinco** = 1 e **tres** = 4. Alternativa errada.

(e) sempre finaliza com valores da variável cinco maiores ou igual a 1, mas a variável tres pode ter valor 0.
Errado. Se a variável **taxa** receber o valor 12, por exemplo, **quociente** = 2 e **resto** = 2. O **caso (resto)** cai na opção 2, e nessa opção **cinco** = **quociente** – 2 = 2 – 2 = 0. Portanto, neste caso **cinco** = 0. Ou seja, nem sempre a variável **cinco** é maior que 1.

Resposta: B

22. (FCC - 2015 - DPE-SP - Programador)

Considere a função Divide apresentada em pseudocódigo.

```
Função Divide (N1, N2: inteiro): real
Var result: real

Início
   result  ←  N1/N2

   Retorne result
Fim
```

Em relação aos conceitos de função e à função Divide acima, é correto afirmar:

(a) Quando são passados valores para os parâmetros da função Divide, os valores são copiados para a função. Este tipo de chamada em que se faz apenas a cópia dos valores é denominado passagem de parâmetro por valor.
(b) Pode-se, no programa principal, usar o comando: imprima (Divide(5,0)) e este comando exibirá 0.
(c) Para chamar a função Divide no programa principal é necessário que sejam declaradas 2 variáveis globais do mesmo tipo e com os mesmos identificadores utilizados na função.
(d) Para chamar a função Divide no programa principal é necessário que seja declarada uma variável real para receber o resultado retornado pela função.
(e) Quando são passados valores para os parâmetros da função Divide, são passados os endereços das variáveis. Este tipo de chamada em que utilizam-se endereços é denominado passagem de parâmetro por valor.

RESOLUÇÃO:

(a) Quando são passados valores para os parâmetros da função Divide, os valores são copiados para a função. Este tipo de chamada em que se faz apenas a cópia dos valores é denominado passagem de parâmetro por valor.
Correto.
(b) Pode-se, no programa principal, usar o comando: imprima (Divide(5,0)) e este comando exibirá 0.
Errado. Ao chamar Divide(5,0) o programa vai dar um erro de tentativa de divisão por 0.
(c) Para chamar a função Divide no programa principal é necessário que sejam declaradas 2 variáveis globais do mesmo tipo e com os mesmos identificadores utilizados na função.
Errado. Os 2 parâmetros N1 e N2 são variáveis locais, só existem dentro do escopo da função Divide. Não é necessária criação de variáveis globais.

(d) Para chamar a função Divide no programa principal é necessário que seja declarada uma variável real para receber o resultado retornado pela função.
Errado. Isso pode acontecer, mas não é obrigatório. O resultado da função pode ser usado implicitamente usando o próprio nome da função. Por exemplo, podemos escrever **se Divide(6,3) = o então escreve("seis é divisível por três")**. Neste caso, a função Divide foi chamada sem que tenha sido criada nenhuma variável para armazenar o retorno do Divide.

(e) Quando são passados valores para os parâmetros da função Divide, são passados os endereços das variáveis. Este tipo de chamada em que utilizam-se endereços é denominado passagem de parâmetro por valor.
Errado. Isso não ocorre. A passagem é feita por valor e não por referência.

Resposta: A

23.(CESGRANRIO - 2014 - Petrobras - Técnico(a) de Informática Júnior)

Considere a situação abaixo.

```
Algoritmo faca_contas
inicio
numero = 3
para i de 1 ate 5, faca
      leia (X)
      se X > 4, entao faca numero ← numero + X
                caso contrario, faca numero ← numero - X
      fimse

fimpara
escreva (numero)
fimalgoritmo
```

Qual é a saída do algoritmo faca_contas para a entrada 7, 3, 5, 2, 3?

(a) 3
(b) 6
(c) 7
(d) 10
(e) 12

RESOLUÇÃO:

No início, numero = 3.

O algoritmo entra na estrutura de repetição.

Quando i = 1, X = 7	X > 4 ? VERDADEIRO Então numero = 3 + 7 = 10
Quando i = 2, X = 3	X > 4 ? FALSO Então numero = 10 - 3 = 7
Quando i = 3, X = 5	X > 4 ? VERDADEIRO Então numero = 7 + 5 = 12
Quando i = 4, X = 2	X > 4 ? FALSO Então numero = 12 – 2 = 10
Quando i = 5, X = 3	X > 4 ? FALSO Então numero = 10 – 3 = 7

No final, ele chama escreve(numero) pela última vez, imprimindo o número 7 na tela.

Resposta: C

24. (CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 9)

Julgue o item seguinte a respeito da construção de algoritmos, dos conceitos de variáveis e de bloco de comandos e das estruturas de controle.

Durante a execução de um programa, o conteúdo de uma variável pode mudar ao longo do tempo, no entanto ela só pode armazenar um valor por vez.

RESOLUÇÃO:

Correto. Essa é a definição de variável. Se fosse constante, pelo contrário, o conteúdo não poderia mudar ao longo do tempo.

Resposta: Certo

25.(CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 9)

Julgue o item seguinte a respeito da construção de algoritmos, dos conceitos de variáveis e de bloco de comandos e das estruturas de controle.

Na lógica de programação, um bloco de comando é definido como um conjunto de ações para determinada função e tem como delimitadores as palavras reservadas INPUT e OUTPUT.

RESOLUÇÃO:

Errado. Os delimitadores de bloco de comando são BEGIN e END.

Resposta: Errado

26. (FCC – 2012 - TRE/SP – Técnico Judiciário - Programação de Sistemas)

Analise o algoritmo a seguir:

```
Algoritmo Cálculo
var n, r, cont: inteiro
início
        r ← 1
        cont ← 1
        enquanto (cont<=5) faça
            leia(n)
            r ← r * n
            cont ← cont + 1
        fim_enquanto
        imprima (r)
fim
```

Se forem lidos os valores 2, 5, 7, 3 e 4,

(a) a saída será 840.
(b) haverá um erro, pois o resultado de um cálculo envolvendo a variável r não pode ser armazenado na própria variável r.
(c) a saída será 210.
(d) haverá um erro, pois o valor gerado será maior do que uma variável do tipo inteiro pode suportar.
(e) a saída será 0.

RESOLUÇÃO:

Inicialmente, são criadas 3 variáveis, **n, r, cont,** todas do tipo inteiro. Preparemos o nosso teste de mesa!

n	r	cont

O código inicia atribuindo a **r** e **cont** o valor **1**.

n	r	cont
	1	1

Na sequência, percebe-se que, enquanto **cont** for menor que 5, ficaremos dentro do laço. **n** é variável inserida pelo usuário, e a questão já nos adianta que os valores inseridos serão **2**, **7**, **5**, **3** e **4**.

Na primeira iteração do laço, **r** recebe o valor de **r** vezes **n**.

n	r	cont
2	1	1
	2	

cont é incrementado em 1, e o laço reinicia, pois a condição permanece válida. O usuário entra com o valor **7** e **r** recebe o valor de **r** vezes **n**.

n	r	cont
2	1	1
7	2	2
	14	

Já sabe onde isso vai dar?

n	r	cont
2	1	1
7	2	2
5	14	3
3	70	4
4	210	5
	840	6

Concorda comigo? Faça com calma, à mão. Perceba que **cont** alcança o valor **6**, para então interromper o laço. Dessa forma, **r**, ao ser impresso, mostra **840** na tela do computador.

Resposta: A

27.(UEL – 2011 - CMTU – Analista Administrativo – Tecnologia da Informação)

Observe o trecho do algoritmo a seguir

```
atribuir 50 a I
atribuir 0 a TOTAL
atribuir 0 a K

enquanto K < I faca

    inıcio
      somar 10 a K;
      atribuir TOTAL+K a TOTAL
      imprimir(K);
    fim;

fim-enquanto;

imprimir(TOTAL);
```

Ao final do processamento, a variável TOTAL e o número de vezes que a K será impressa são, respectivamente:

(a) 100 e 4
(b) 150 e 5
(c) 150 e 8
(d) 150 e 9
(e) 210 e 6.

RESOLUÇÃO:

A estrutura de repetição, novamente, será a que lhe exigirá mais atenção no código:

```
enquanto K < I faca

    inıcio
      somar 10 a K;
      atribuir TOTAL+K a TOTAL
      imprimir(K);
    fim;

fim-enquanto;
```

Essa estrutura do tipo "enquanto...faça" faz exatamente isso. Verifica uma condição, no caso, o valor de K, e, enquanto a condição estiver sendo atendida, ele executa o código no seu interior. Ao término, volta a verificar a condição e executa quantas vezes a condição estiver sendo atendida. Daí você conclui que, dentro desta estrutura, a variável de verificação (no caso, K), deve ser modificada ao longo da execução. Caso isso não acontecesse, teríamos um "loop infinito".

Enfim, quando a condição de verificação deixar de ser atendida, a estrutura "enquanto... faça" será pulada e o restante do código continuará a ser executado.

Voltando ao exercício, temos primeiro algumas atribuições de variáveis:

atribuir 50 a I – I passa a valer 50

atribuir 0 a TOTAL – TOTAL passa a valer 0

atribuir 0 a K – K passa a valer 0

Vejamos agora a estrutura mais delicada, a condicional:

```
enquanto K < I faca

    inıcio
      somar 10 a K;
      atribuir TOTAL+K a TOTAL
      imprimir(K);
    fim;

fim-enquanto;
```

Um dos objetivos é saber quantas vezes K será impresso. K será impresso tantas vezes quantas adentrar-se na estrutura de repetição.

Com K = 0, entrar-se-á uma vez na estrutura. K passará a valer 10 (somar 10 a K). Entrar-se-á na estrutura novamente.

Contando nos dedos da mão (sim, essa é a melhor forma de se contar, fazemos isso o tempo todo), percebe-se que a estrutura será executada com k = 0,10,20,30,40 ou seja , 5 vezes. Perceba que é K<I, e não K<=I, o que são duas coisas diferentes!

Tendo metade da resposta, falta agora saber o valor de TOTAL.

Na primeira vez, TOTAL = TOTAL(0) + K(10) -> perceba que soma-se 10 a K antes da atribuição da soma a total;

Na segunda vez, TOTAL = TOTAL(10) + K(20);

Na terceira vez, TOTAL = TOTAL(30) + K(30);

Na quarta vez, TOTAL = TOTAL(60) + K(40);

Na quinta vez, TOTAL = TOTAL(100) + K(50);

Ainda, observe que o último valor de K, 50, será reprovado na próxima verificação, por ser igual a I, e toda a estrutura "enquanto" será pulada. Entretanto, nada impediu que, quando dentro da estrutura, na passagem anterior, K, que valia 40, passasse a valer 50, pois a verificação somente ocorre no início.

Pulada a estrutura "enquanto...faça", o valor de TOTAL será impresso, e este equivale a 150.

Portanto, nossa resposta certa é a letra **b**.

Resposta: B

28. (FCC – 2012 - ARCE – Analista de Regulação - Analista de Sistemas)

Considere o algoritmo em pseudocódigo:

```
Var A, B, C: lógico
Var X, Y: real
Var R1, R2, R3: inteiro
Início
    A ← verdadeiro
    B ← falso
    C ← verdadeiro
    X ← 2.5
    Y ← 3.5
    se ( C ou (X-Y > 0) )
      então
        R1 ← 1
      senão
        R1 ← 0

    se ( ( (A e B) ou C ) e ( (X > Y) ou não(A) ) )
      então
        R2 ← 1
      senão
       R2 ← 0

   se ( (B ou (X-Y <0)) e ( (B e não(A)) ou (não (C) ) )
     então
       R3 ← 1
     senão
     R3 ← 0
Fim
```

Os valores de R1, R2 e R3, após a execução do algoritmo são:

(a) R1=0, R2=1, R3=1
(b) R1=1, R2=1, R3=0
(c) R1=1, R2=0, R3=0
(d) R1=0, R2=0, R3=1
(e) R1=0, R2=0, R3=0

RESOLUÇÃO:

Pois bem, pessoal, vamos analisar o nosso exercício.

Antes da execução do algoritmo, propriamente dita, ocorre a **declaração de variáveis**.

A,B e C são variáveis lógicas, também chamadas de **booleanas**, pois podem assumir apenas dois valores: certo e errado, true ou false, 1 ou 0;

X e Y podem assumir valores de números reais;

R1, R2 e R3 podem assumir valores de números inteiros.

Quando o algoritmo se inicia, ocorre uma série de atribuição de valores. A e C são verdadeiros, B é falso, X vale 2,5 e Y 3,5.

Na sequência, encontramos nossa primeira **estrutura condicional**:

se (C ou **(X – Y > 0)**)

OU é uma estrutura do tipo "caso uma das condições seja verdadeira, tudo é verdadeiro". X-Y será menor que zero, mas, como C é verdadeiro, isto significa que a condição foi atendida e o conteúdo condicional será executado. **Então,** R1 receberá o valor **1**.

Nossa segunda estrutura condicional diz que:

se (((A e B) ou C) e ((**X** > **Y**) ou não(A)))

Mais uma vez, temos que descobrir se a sentença nos retorna **verdadeiro** ou **falso**, para executarmos o **então** ou o **senão**. Como a nossa matemática nos ensina, vamos resolver os parênteses de dentro para fora:

A e B -> B é falso, logo, A e B -> **falso**;

falso ou C -> C é verdadeiro, logo, falso ou C -> **verdadeiro**;

X>Y é **falso**;

não(A) é **falso**, uma vez que A é verdadeiro;

falso ou falso é **falso**;

Então, aquela complexa estrutura condiciona ficou igual a:

se (**verdadeiro** e **falso**)

Assim sendo, a sentença é **falsa** e o próximo passo do algoritmo será executar o **senão**. Logo, R2 recebe o valor **0**.

Por fim, temos a terceira estrutura condicional, cuja sentença é a mais extensa:

se ((B ou (**X**-**Y**<0)) e ((B e não(A)) ou (não(C)))

Vamos estudá-la em detalhes, e reescrevendo-a a cada passo.

X-Y < 0 -> 2,5 – 3,5 < 0 -> **verdadeiro**;

não(A) -> não (verdadeiro) -> **falso**;

não(C) -> não (verdadeiro) -> **falso**;

B -> **falso**;

Com isso, temos:

se ((**falso** ou **verdadeiro**) e ((**falso** e **falso**) ou **falso**))

falso ou verdadeiro -> **verdadeiro**;

falso e falso -> **falso**; -> falso ou falso -> **falso**;

E a sentença se resume a se (**verdadeiro** e **falso**), o que nos leva a **falso**, e, novamente, a condição executada será o **senão**. Isto posto, R3 recebe o valor **0**.

Finalizado o algoritmo, **R1** vale **1**, **R2** e **R3** valem **0**.

Conseguiu acompanhar? Um programa nada mais é que uma série de procedimentos simples, que, uma vez unidos, se tornam um todo altamente complexo. Portanto, para "debugá-lo", o procedimento é decompor o problema até o nível que você enxerga passos simples, e assim consegue analisá-lo.

Resposta: C

29. (FCC – 2010 - TRF/4ª Região – Analista Judiciário – Informática)

Considere:

```
algoritmo PROVA
var N,X,CONTA,CONTB,EXP,RESP: inteiro
inicio
    N ← 4
     X ← 2
    RESP← X
    para  CONTA←2 até N passo 2 faça
                EXP ← X
                CONTB ← 1
                para CONTB=1 até CONTA-1 passo 1 faça
                            EXP ← EXP * X
                fim_para
                RESP ← RESP + EXP
    fim_para
    imprima (RESP)
fim
```

Dado o algoritmo representado na forma de português estruturado, o valor de saída contido na variável RESP será

(a) 6.
(b) 22.
(c) 86.
(d) 0.
(e) 342

RESOLUÇÃO:

Este exercício é um pouco mais complexo, pois possui uma estrutura de repetição dentro da outra. Tal visualização é possível por causa da **indentação**, que é esse recurso visual de tabular mais à direita as estruturas que devem ser executadas dentro de outras estruturas.

Quando nos deparamos com esse tipo de exercício, a melhor coisa que se pode fazer, para não se perder durante a "debugação", é fazer uma tabela com todas as variáveis e os valores que elas assumem ao longo da execução do programa. Assim:

N	4					
X	2					
CONTA						
CONTB						
EXP						
RESP	2					

Agora, toda vez que alguma variável sofrer uma modificação, basta escrevê-la na tabela, para não se perder. Sem mais rodeios, vamos encarar o algoritmo:

Com N,X e RESP recebendo seus valores iniciais, temos a primeira estrutura **para.** Nela, CONTA recebe o valor 2 e tal estrutura deverá ser repetida até que CONTA alcance o valor de N (ou seja, quando o valor de CONTA for maior ao valor de N a estrutura deverá ser pulada). Detalhe: a estrutura tem **passo 2**, ou seja, o contador incrementa de 2 em 2 inteiros. Como 2 é menor do que 4, entra-se.

Ao entrar no primeiro **para**, EXP recebe o valor de X, **2**, CONTB recebe o valor de **1**, e precisamos entrar na segunda estrutura **para.** Mantenhamos o controle do valor das variáveis!

N	4					
X	2					
CONTA	2					
CONTB	1					
EXP	2					
RESP	2					

No segundo **para**, a estrutura deverá ser executada até que CONTB atinja o VALOR de CONTA-1, com **passo 1.** Como 1 é igual a 1, entra-se.

Dentro do segundo **para**, EXP recebe o próprio valor multiplicado por X, ou seja, 2*2, que é igual a 4. EXP passa a valer 4.

No **fim_para**, volta-se ao seu início, incrementa-se o contador e verifica-se a condição novamente. CONTB agora vale 2, CONTA -1 continua sendo 1. Logo, o segundo **para** deve ser pulado.

Ainda estamos dentro do primeiro **para**! Chegamos á linha remanescente, na qual RESP deverá receber seu próprio valor acrescido de EXP. 2 + 4, 6. Você continua controlando as variáveis?

N	4					
X	2					
CONTA	2					
CONTB	1	2				
EXP	2	4				
RESP	2	6				

Chegamos ao fim do primeiro ciclo do primeiro **para**. Logo, devemos incrementar o contador, de **passo 2**, e tentar executar a iteração novamente.

CONTA agora vale 4, ficando igual a N. Podemos prosseguir.

EXP novamente assume o valor 2, e CONTB volta a assumir o valor 1. Tudo sob controle?

N	4					
X	2					
CONTA	2		4			
CONTB	1	2	1			
EXP	2	4	2			
RESP	2	6				

No segundo **para** CONTB vale 1 e deverá ser incrementado até CONTA-1, ou seja, até **3.**

Na prática, EXP receberá o seu próprio valor multiplicado por X várias vezes.

Quando CONTB = 1, EXP será 2*2, **4**;

Quando CONTB = 2, EXP será 4*2, **8**;

Quando CONTB = 3, EXP será 8*2, **16**;

Quando CONTB = 4, pula-se o segundo **para**.

Acompanhou?

N	4						
X	2						
CONTA	2		4				
CONTB	1	2	1	1	2	3	4
EXP	2	4	2	4	8	16	16
RESP	2	6					

Pulado o segundo **para**, executa-se a última linha do primeiro **para**, atribuindo a RESP o seu próprio valor acrescido de EXP. 6 + 16, **22**.

Agora CONTA passou a valer **6**, por causa do passo **2**, e como 6 é maior que 4, o primeiro **para** deve ser pulado. Mais controle!

N	4						4
X	2						2
CONTA	2		4				6
CONTB	1	2	1	1	2	3	4
EXP	2	4	2	4	8	16	16
RESP	2	6					22

RESP foi impresso, com o valor **22**, e você já sabe os valores finais de todas as variáveis do código. É importante manter esse controle, para que você não se perca.

Resposta: B

30. (FCC – 2012 - METRÔ/SP – Analista Desenvolvimento Gestão Júnior – Ciências da Computação)

Analise a estrutura abaixo.

```
Algoritmo Cálculo
var n, cont, resp, s: inteiro
início
    leia(n)
    resp ← 0
    cont ←1
    s ← -1
    enquanto (cont<=n) faça
        se (cont mod 2 = 0)
            então resp ← resp + cont * s
            senão resp ← resp + cont
        fim_se
        cont ← cont+1
    fim_enquanto
    imprima(resp)
fim
```

Considere mod o operador que calcula o resto da divisão entre dois números inteiros.

Por meio de um teste de mesa é possível constatar que o portugol mostrado representa a resolução da equação:

(a) resp = 0 + 1 – 2 + 3 – 4 + 5 – ... (+ ou –) n
(b) resp = 0 + 1 – 22 + 34 – 46 + 5n
(c) resp = 0 – 1 – 2 – 3 – 4 – 5 – ... – n
(d) resp = 0 + 1! – 2! + 3! – 4! + 5! – n!
(e) resp = 0 – 1 + 2 – 3 + 4 – 5 + ... – n

RESOLUÇÃO:

Esse código já está com cara de que vai dar trabalho! Então, logo para início de conversa, vamos fazer a nossa tabelinha com as variáveis do código:

N	?					
CONT	1					
RESP	0					
S	-1					

N, pelo visto, será o parâmetro inserido pelo usuário. A função **leia(n)** nos mostra isso. Logo, toda vez que esse programa for executado, **leia(n)** trará o valor de **N** para o programa e continuará a sua execução.

O algoritmo começa pra valer na estrutura de repetição "enquanto...faça". Esta, por sua vez, já inicia com uma estrutura condicional: **se** (cont mod 2 = 0).

mod, como o próprio enunciado da questão informa, é o operador que calcula o resto da divisão entre dois números inteiros. Saber se X mod 2 = 0, por exemplo, significa saber se o resto da divisão de X por 2 é zero ou não. Na prática, serve para analisar se o número é par ou ímpar, percebeu? É só analisar:

1 mod 2 = 0 -> falso;

2 mod 2 = 0 -> verdadeiro;

3 mod 2 = 0 -> falso;

4 mod 2 = 0 -> verdadeiro;

Programação é isso, pequenos raciocínios o tempo todo. Continuemos:

Na sequência, temos um **então**, executado quando o contador for par, e um **senão**, executado quando o contador for ímpar.

Quando o contador é **par**, a **resp** é adicionado o valor do contador com sinal negativo (resp <- resp + cont*s);

Por sua vez, quando o contador é **ímpar**, a **resp** é adicionado o valor do próprio contador, sem modificações (resp <- resp + cont);

Já conseguiu enxergar isso? Se você estiver enxergando, já será um excelente sinal. Veja como a tabela evolui ao longo de três contagens:

N	?	?	?	?		
CONT	1	1	2	3		
RESP	0	1 (0+1)	-1 (0+1-2)	2 (0+1-2+3)		
S	-1	-1	-1	-1		

O procedimento do algoritmo é esse. Quando contador (**CONT**) ultrapassa o valor de **N**, a estrutura "enquanto...faça" é abortada e o valor de **RESP** é impresso.

Assim sendo, percebemos que **resp** representa a solução da equação 0 + 1 - 2 + 3 - 4 + 5 ... (+ ou -) n.

Resposta: A

31.(CESPE – 2008 - SERPRO – Técnico – Programação)

A respeito da lógica de programação, que é fundamental para o desenvolvimento de códigos por meio de linguagens de programação, julgue os itens subsequentes.

Um algoritmo pode ser definido como uma sequência finita de passos que levam à execução de determinada tarefa ou conjunto de tarefas.

RESOLUÇÃO:

Definição clássica de algoritmo. É isso aí.

Resposta: Certo

32.(CESPE – 2008 - SERPRO – Técnico – Programação)

Na lógica de programação, a instrução é o comando principal que indica a um programa uma condição estrutural a repetir.

RESOLUÇÃO:

Faltou definir qual seria o tipo da instrução. Se não, fica muito genérico. **While**, **for** e **repeat until** são instruções que definem uma estrutura a ser repetida dentro de um programa.

Resposta: Errado

33.(CESPE – 2008 - SERPRO – Técnico – Programação)

Um diagrama de blocos é uma forma padronizada para se representar os passos lógicos de determinado processamento. Por meio do diagrama, pode ser utilizada uma sequência de símbolos, com significado bem definido, para auxiliar a representação dos passos de um processamento.

RESOLUÇÃO:

Correta. Os diagramas são ferramentas úteis para representar visualmente a execução de um algoritmo.

Resposta: Certo

34. (CESPE – 2008 - SERPRO – Técnico – Programação)

Os operadores relacionais são utilizados para comparar números. Para se comparar sequências de caracteres (strings), são utilizados os operadores lógicos, que retornam valores verdadeiro e falso.

RESOLUÇÃO:

Na verdade, é possível comparar strings (texto) utilizando operadores relacionais, que também retornam verdadeiro ou falso. Maior ou menor comparam tamanho, enquanto igual ou diferente comparam conteúdo.

Resposta: Errado

35. (CESPE – 2008 - SERPRO – Técnico – Programação)

Em lógica de programação, uma constante é um valor fixo que não se modifica ao longo do tempo durante a execução de um programa. Essa constante pode ser numérica, lógica ou literal.

RESOLUÇÃO:

A constante não pode ser modificada, apenas a variável.

Resposta: Certo

---

36. (CESPE – 2013 - SERPRO – Analista – Suporte Técnico)

Julgue os itens seguintes, relativos a programação e lógica de programação.

Parâmetros são pontos de comunicação entre módulos de um programa. A passagem de parâmetros, que consiste na substituição do parâmetro formal pelo parâmetro real, pode ser realizada por valor ou por referência.

RESOLUÇÃO:

Exato. Quando você chama uma função passando parâmetros, esses parâmetros são uma forma de comunicação, isto é, de transferir dados entre o módulo que chamou a função e a função que foi chamada.

Resposta: Certo

---

37. (CESPE – 2013 - SERPRO – Analista – Suporte Técnico)

Segundo o pseudocódigo abaixo, um vetor de 100 números é lido e, em seguida, é montado um segundo vetor a partir dos valores do primeiro vetor multiplicados por 3.

```
início
      VET1,VET2 : vetor [1..100] numérico
      CONTADOR : numérico
      para CONTADOR de 1 até 100 faça
            leia "Digite um número: ",VET1[CONTADOR]
            VET2[CONTADOR] ← (VET1[CONTADOR] * 3)
      fim-para
fim
```

RESOLUÇÃO:

A montagem do segundo vetor ocorre paralelamente à montagem do primeiro vetor (não em seguida, como sugere a assertiva), já que a leitura dos valores do vetor ocorre **dentro do laço**. Faça o teste de mesa se tiver dúvidas!

Resposta: Errado

---

38. (CESPE – 2011 - STM – Analista Judiciário – Análise de Sistemas – adaptada)

Com relação a algoritmos e lógica de programação, julgue os itens a seguir.

Na passagem de parâmetros por referência, o valor do parâmetro real é copiado para o parâmetro formal do módulo, preservando, assim, o valor original do parâmetro. Na passagem de parâmetros por valor, toda alteração feita nos parâmetros formais reflete-se nos parâmetros reais.

RESOLUÇÃO:

Inversão de conceitos! "Na passagem de parâmetros por **valor**, o valor do parâmetro real é copiado para o parâmetro formal do módulo, preservando, assim, o valor original do parâmetro. Na passagem de parâmetros por **referência**, toda alteração feita nos parâmetros formais reflete-se nos parâmetros reais.".

Resposta: Errado

---

39. (CESPE – 2011 - STM – Analista Judiciário – Análise de Sistemas – adaptada)

Nas estruturas de controle, tais como as estruturas de seleção simples, compostas ou encadeadas, é necessário verificar as condições para a realização de uma instrução ou sequência de instruções.

RESOLUÇÃO:

Isso aí. A verificação das condições ocorre no comando **if**.

Resposta: Certo

---

40. (CESPE – 2012 - Banco da Amazônia – Técnico Científico – Análise de Sistemas)

O comando while utilizado em algoritmos implementa laços com teste antecipado de condições, testando a condição e, sendo ela verdadeira, executando o bloco de comandos.

RESOLUÇÃO:

Correta. Já o **repeat until** verifica a condição depois da execução da estrutura.

Resposta: Certo

---

41. (CESPE – 2012 - Banco da Amazônia – Técnico Científico – Análise de Sistemas)

Quando um break é encontrado dentro de um laço for, a execução do código é interrompida e o programa é finalizado.

RESOLUÇÃO:

Não é bem assim. O break interrompe o laço **for**, mas a execução prossegue normalmente nas instruções subsequentes ao laço. O programa não é necessariamente finalizado.

Resposta: Errado

---

42. (CESPE – 2009 - UNIPAMPA – Analista de Tecnologia da Informação – Rede e Suporte)

Em relação aos conceitos de lógica de programação utilizados para a construção de algoritmos, julgue os próximos itens.

Valores que sejam armazenados em variáveis locais de determinado procedimento ou função não podem ser utilizados em outros procedimentos ou funções.

RESOLUÇÃO:

Não há problema algum. É possível passar uma variável local como parâmetro para outro procedimento ou função.

Resposta: Errado

---

43. (CESPE – 2009 - UNIPAMPA – Analista de Tecnologia da Informação – Rede e Suporte)

Estruturas de repetição permitem que uma sequência de comandos seja executada repetidamente até que determinada condição de interrupção seja satisfeita. É possível que, em determinada execução do algoritmo, a sequência de comandos não seja executada nenhuma vez.

RESOLUÇÃO:

Correta. Se um comando **while** verificar uma condição que seja falsa logo na primeira verificação, a estrutura de repetição é pulada sem nenhuma execução.

Resposta: Certo

---

44. (CESPE – 2009 - UNIPAMPA – Analista de Tecnologia da Informação – Rede e Suporte)

Registros são estruturas consideradas heterogêneas porque são compostos de dados que, apesar de serem logicamente relacionados, não têm necessariamente o mesmo tipo.

RESOLUÇÃO:

Correta. Registros são utilizados em programação estruturada. Seu equivalente, na programação orientada a objetos, é o objeto.

Resposta: Certo

---

45. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

Julgue os itens seguintes, relativos à lógica de programação e construção de algoritmos.

Na definição de uma função, a passagem de parâmetros por referência possibilita que o valor de uma variável passado como argumento seja alterado na função, e sua alteração mantenha-se mesmo após a execução da função.

RESOLUÇÃO:

É isso mesmo.

Resposta: Certo

---

46. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

É possível implementar procedimentos cujos valores gerados podem ser armazenados em variáveis que garantem sua existência mesmo após o término da execução de tais procedimentos.

RESOLUÇÃO:

Sim. Basta armazenar esses valores em **variáveis globais**, que deverão estar fora de todas as funções ou procedimentos do programa.

Resposta: Certo

---

47. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

Estruturas de repetição são usadas para que determinado bloco de comandos seja executado diversas vezes. A garantia de parada da repetição ocorre por meio de uma condição que é verificada a cada nova iteração. Dependendo do tipo de estrutura de repetição utilizado, o bloco de comandos é executado pelo menos uma vez.

RESOLUÇÃO:

Quando se utiliza o **repeat until**, o bloco de comandos será executado pelo menos uma vez.

Resposta: Certo

---

48. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

Variáveis declaradas dentro de funções ou procedimentos são chamadas de variáveis locais e não são visíveis por outras funções. Por esse motivo, não é possível declarar variáveis que possam ser utilizadas por qualquer função de um programa.

RESOLUÇÃO:

É possível por meio de **variáveis globais**.

Resposta: Errado

---

49. (FCC - 2012 - ARCE - Analista de Regulação - Analista de Sistemas)

Há duas maneiras de se passar argumentos ou parâmetros para funções: por valor e por referência. Todas as afirmativas sobre passagem de parâmetros estão corretas, EXCETO:

(a) Na passagem por referência, o que é passado como argumento no parâmetro formal é o endereço da variável.
(b) Na passagem por valor, o valor é copiado do argumento para o parâmetro formal da função.
(c) Por exemplo, quando duas variáveis inteiras i1 e i2 são passadas por valor à função troca() chamada pelo programa principal, elas também são alteradas no programa principal.
(d) Na passagem por referência, dentro da função, o argumento real utilizado na chamada é acessado através do seu endereço, sendo assim alterado.
(e) Na passagem por valor, quaisquer alterações feitas nestes parâmetros dentro da função não irão afetar as variáveis usadas como argumentos para chamá-la.

RESOLUÇÃO:

(a) Na passagem por referência, o que é passado como argumento no parâmetro formal é o endereço da variável.

Certo.

(b) Na passagem por valor, o valor é copiado do argumento para o parâmetro formal da função.

Certo.

(c) Por exemplo, quando duas variáveis inteiras i1 e i2 são passadas por valor à função troca() chamada pelo programa principal, elas também são alteradas no programa principal.

Errado. Na passagem por valor, são criadas cópias das variáveis i1 e i2 visíveis somente dentro da função troca(). O que for alterado nessas cópias não afeta as variáveis originais i1 e i2 no programa principal.

(d) Na passagem por referência, dentro da função, o argumento real utilizado na chamada é acessado através do seu endereço, sendo assim alterado.

Certo.

(e) Na passagem por valor, quaisquer alterações feitas nestes parâmetros dentro da função não irão afetar as variáveis usadas como argumentos para chamá-la.

Certo.

Resposta: C

---

50. (FCC - 2011 - TRT - 1ª REGIÃO (RJ) - Analista Judiciário - Tecnologia da Informação)

Considere:

(I) Sequência, decisão e iteração são as estruturas necessárias e suficientes para o desenvolvimento da programação de computadores.
(II) Uma sequência de passos, incluindo a forma como os dados serão armazenados no computador, permitindo que o problema possa ser resolvido de maneira automática e repetitiva.
(III) Cada instrução é traduzida para uma representação interna e interpretada pela simulação de funcionamento do processador, o que torna mais rápido o ciclo escrita-execução-modificação.
(IV) A sua aplicação divide e estrutura o algoritmo em partes fechadas e coerentes para evitar a repetição de uma sequência de comandos que é utilizada em várias partes do programa.

Em relação à programação de computadores, as definições contidas nos itens I, II, III e IV correspondem, respectivamente, à programação

(a) linear, ao fluxograma, à compilação e ao procedimento.
(b) modular, ao algoritmo, à constante e à interpretação.
(c) orientada a objetos, ao projeto lógico, ao fluxograma e à compilação.
(d) linear, ao algoritmo, à compilação e à função.
(e) estruturada, ao algoritmo, à interpretação e ao procedimento.

RESOLUÇÃO:

Vamos analisar as definições I, II, III e IV.

(I) Sequência, decisão e iteração são as estruturas necessárias e suficientes para o desenvolvimento da programação de computadores.

Estruturas de sequência são o fluxo normal de execução das instruções na ordem em que aparecem, sequencial. Estruturas de decisão são o se-então, se-então-senão e o caso-selecione. Estruturas de iteração são as repetições que estudamos. Essas são as estruturas necessárias e suficientes para a programação **estruturada**.

(II) Uma sequência de passos, incluindo a forma como os dados serão armazenados no computador, permitindo que o problema possa ser resolvido de maneira automática e repetitiva.

Definição de **algoritmo**.

(III) Cada instrução é traduzida para uma representação interna e interpretada pela simulação de funcionamento do processador, o que torna mais rápido o ciclo escrita-execução-modificação.

Característica de linguagens de programação **interpretadas**.

(IV) A sua aplicação divide e estrutura o algoritmo em partes fechadas e coerentes para evitar a repetição de uma sequência de comandos que é utilizada em várias partes do programa.

Está falando de modularização de código por meio de **procedimentos**.

Resposta: E

Lista de questões comentadas

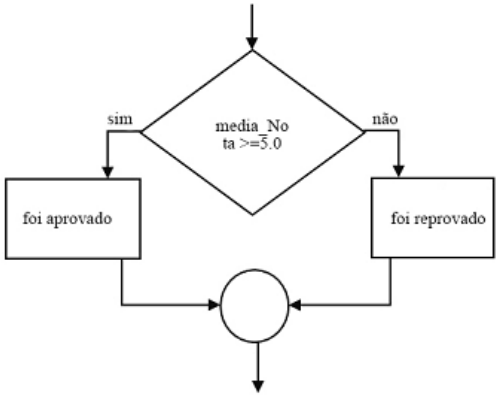
1. (CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o próximo item, concernente ao conceito relacionado a algoritmos e linguagens de programação.

Em um algoritmo, uma constante é um espaço físico na memória, e é identificada por um nome que não sofre alteração durante a execução do programa.

---

2. (CESPE - 2017 - TRT - 7ª Região (CE) - Analista Judiciário - Tecnologia da Informação)



A estrutura lógica presente no diagrama apresentado é do tipo

(a) SE ENTÃO.
(b) CASO SELECIONE.
(c) CASO REPITA
(d) SE ENTÃO SENÃO.

---

3. (FCC - 2018 - SABESP - Analista de Gestão - Sistemas)

De acordo com dados da SABESP, um pequeno buraco de 2 milímetros no encanamento desperdiça 3,2 mil litros de água em um dia. Um Analista escreveu o algoritmo em pseudocódigo abaixo para calcular o desperdício de água em função de buracos em encanamentos.

```
Var largburaco, desperdicio: real
       dias: inteiro
Início
         imprima("Digite a largura do buraco em milímetros: ")
         leia(largburaco)
         imprima("Digite o número de dias do vazamento: ")
         leia(dias)
         se  I
                  então
                       II
                       imprima("Em ", dias, " dias foram desperdiçados ", desperdicio,
                       " mil litros de água ")
                  senão   imprima("Dado(s) inválido(s)")
         fimse
Fim
```

O comando que preenche corretamente a lacuna

(a) **I** é: (largburaco > 0 ou dias > 0)
(b) **I** é: (largburaco > 0 e desperdicio > 0)
(c) **II** é: desperdicio ← (largburaco/2.0) * 3.2 * dias
(d) **II** é: desperdicio ← desperdicio + (largburaco/2.0) * 3.2
(e) **II** é: desperdicio ← (largburaco/3.2) * 2.0 * dias

---

4. (CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 8)

Julgue o item subsequente, relativo à lógica de programação.

Na passagem de parâmetro por referência, é possível alterar o valor da variável que é apontada por referência.

---

5. (VUNESP - 2016 - MPE-SP - Analista Técnico Científico - Engenheiro de Computação)

No contexto de passagem de parâmetros para uma sub-rotina, existe a denominada passagem de parâmetro por valor. Nesse caso,

(a) o parâmetro pode ser passado para a sub-rotina, desde que ela seja uma sub-rotina de tratamento de interrupção.
(b) o endereço onde se encontra o valor a ser passado como parâmetro é fornecido para a sub-rotina.
(c) um ponteiro para o endereço onde se encontra o valor a ser passado como parâmetro é fornecido para a sub-rotina.
(d) um registrador que aponta para o valor a ser passado como parâmetro é fornecido para a sub-rotina.
(e) uma cópia do valor do parâmetro é fornecida para a sub-rotina.

---

6. (FUMARC - 2013 - TJM-MG - Oficial Judiciário - Assistente Técnico de Manutenção de Informática)

Em relação aos conceitos de lógica de programação, analise as seguintes afirmativas:

(I) Uma constante é um determinado valor fixo que não se modifica durante a execução de um programa.
(II) Uma variável está associada a uma posição de memória, cujo conteúdo pode ser modificado durante a execução do programa.
(III) Toda variável é identificada por uma constante

Estão CORRETAS as afirmativas:

(a) I e II, apenas.
(b) I e III, apenas.
(c) II e III, apenas.
(d) I, II e III.

---

7. (CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o próximo item, concernentes aos conceitos relacionados a algoritmos e linguagens de programação.

A resposta do algoritmo seguinte é 8.

```
função pfactor(num) {
    if (num <= 1) return 1;
    return pfactor(num-1) + pfactor(num-2);
}
x=5;
factors = pfactor(x);
escreva(factors);
```

---

8. (CONSULPAM - 2018 - Câmara de Juiz de Fora - MG - Assistente Legislativo – Técnico em Informática)

Analise os itens abaixo que versam sobre Lógica de Programação e depois responda:

(I) Lógica de programação é o modo como se escreve um programa de computador, um algoritmo. Um algoritmo é uma sequência de passos para se executar uma função.

(II) A linguagem de programação é como uma língua normal, um grupo de palavras com significados. No caso da programação, a maioria das linguagens é escrita em Inglês. Estas linguagens fazem o computador assimilar cada comando e função de um algoritmo, depois executar cada função.

(III) Na hora de programar alguns passos são indispensáveis, como Declarar Variáveis. Variáveis são escritas exclusivamente por letras, que representam um valor que pode ser mudado a qualquer momento.

(IV) Saber lógica de programação é saber o melhor jeito de escrever um código, para o computador interpretar corretamente. É saber se comunicar com a máquina a partir de uma linguagem seja lá qual for.

Analisados os itens é CORRETO afirmar que:

(a) Todos os itens estão corretos.
(b) Apenas o item IV está incorreto.
(c) Apenas o item III está incorreto.
(d) Conjunto finito de passos.

---

9. (CONSULPAM - 2018 - Câmara de Juiz de Fora - MG - Assistente Legislativo – Técnico em Informática)

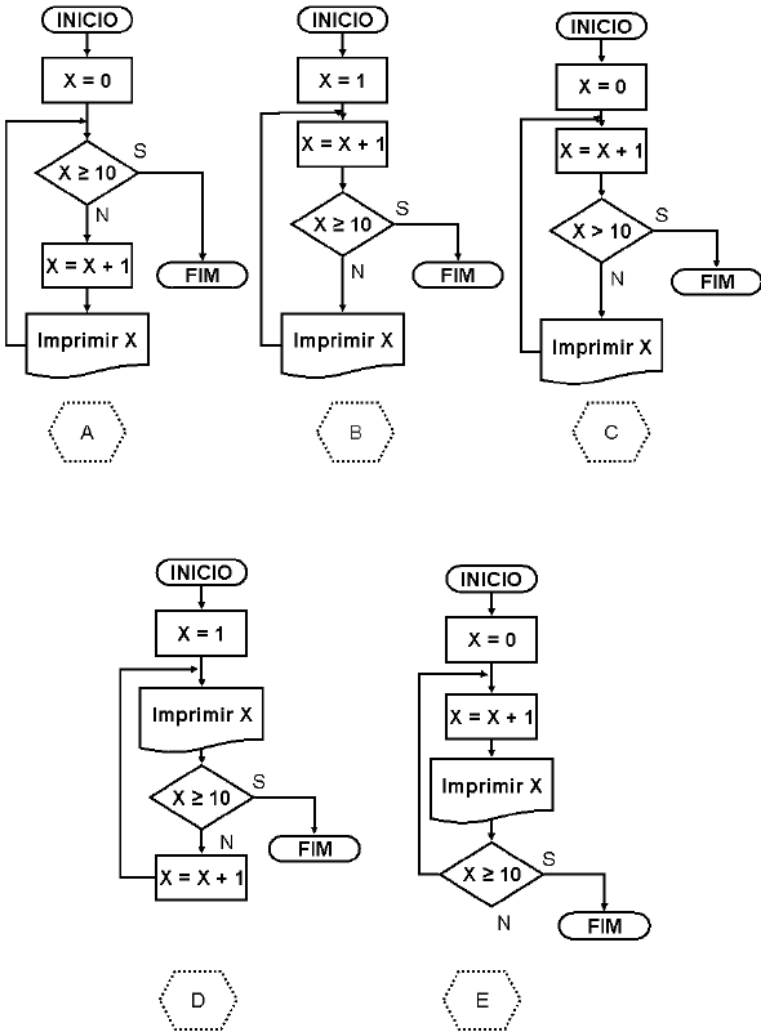
Algoritmo é uma sequência finita e bem definida de passos que, quando executados, realizam uma tarefa específica ou resolvem um problema. NÃO é uma das propriedades do algoritmo:

(a) Composto por ações complexas e por vezes indefinidas.
(b) Composto por ações simples e bem definidas (não pode haver ambiguidade, ou seja, cada instrução representa uma ação que deve ser entendida e realizada).
(c) Sequência ordenada de ações.
(d) Conjunto finito de passos.

---

10. **(FAURGS - 2018 - TJ-RS - Programador)**

Considere os cinco fluxogramas abaixo, identificados pelas letras A, B, C, D e E, que geram valores da variável X e imprimem esses valores dentro de uma faixa controlada.



O objetivo dos fluxogramas é imprimir valores de X, na faixa de 1 a 10 (incluindo os limites). Porém um dos fluxogramas imprime valor(es) fora desta faixa. Qual é este fluxograma?

11.(CESPE - 2018 - BNB - Especialista Técnico - Analista de Sistema)

Julgue o item a seguir, relativo ao conceito de construção de algoritmos.

O algoritmo a seguir apresenta um exemplo de busca sequencial.

```
var a:vetor[1..5] de inteiro;
    temp:inteiro;
    i,k:inteiro;
inicio
    a[1]:=57;
    a[2]:=58;
    a[3]:=60;
    a[4]:=56;
    a[5]:=59;
    para i:=1 até 4 faça inicio
        para k:=i+1 até 5 faça inicio
            se a[i] > a[k] então inicio
               temp:=a[i];
               a[i]:=a[k];
               a[k]:=temp;
            fim se;
        fim para;
    fim para;
    escreva('Resultado: ');
    para i:=1 até 5 faça inicio
        escreva(a[i],',');
    fim para;
fim;
```

12. (IF-MT - 2018 - IF-MT - Informática)

Analise o trecho do algoritmo abaixo representado em português estruturado:

```
COMECO < - 1
        FINAL < - 10
        ACHA <- falso
        enquanto (COMECO <= FINAL) e (ACHA = falso) faca
                MEIO <- (COMECO + FINAL) div 2
                se (PESQ = NOME[MEIO]) entao
                        ACHA < - verdadeiro
                senao
                        se (PESQ < NOME[MEIO]) entao
                                FINAL <- MEIO - 1
                        senao
                                COMECO <- MEIO + 1
                        fimse
                fimse
        fimenquanto
```

É correto afirmar que:

(a) Trata-se do trecho de um algoritmo de pesquisa binária que só é possível ser empregada se os dados estiverem ordenados no sistema FIFO.
(b) Trata-se do trecho de um algoritmo de pesquisa sequencial que deve ter ordenado o vetor anteriormente para que pudesse ser empregada.
(c) Trata-se de um algoritmo de ordenação do tipo bubble sort onde é enviado ao fim da fila o maior elemento da lista.
(d) Trata-se de um trecho de algoritmo de pesquisa binária em um vetor que deve estar previamente ordenado em ordem crescente para seu emprego.
(e) Trata-se de um algoritmo de ordenação sequencial para que posteriormente seja possível pesquisar no vetor Nome.

13.(CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 8)

Considere o pseudo-código abaixo:

```
F0 = 0
F1 = 1
I = 0
PARA I DE 1 ATÉ 10, FAÇA
      T = F1
      F1 = F1 + F0
      F0 = T
FIM PARA
MOSTRE F1
```

O valor da variável F1 exibido é

(a) 54
(b) 33
(c) 89
(d) 93
(e) 141

---

14. (SUGEP - UFRPE - 2018 - UFRPE - Técnico de Tecnologia da Informação - Sistemas)

Considere a função recursiva 'func' definida por

```
func(1) = 1
func(n) = (n - 1) * func(n - 1)
```

Quais são os valores de func(4) e func(5), respectivamente?

(a) 24 e 120
(b) 12 e 24
(c) 6 e 24
(d) 1 e 2
(e) 2 e 6

15.(SUGEP - UFRPE - 2018 - UFRPE - Técnico de Tecnologia da Informação - Sistemas)

Considere o algoritmo a seguir.

```
Inteiro x1 =2, x2 = -1, x3 = 4
Enquanto (x1 > 0) faça
    x2 = x3/3-x2*4
    x1 = x3 % x1
Fim enquanto
Imprime(x2)
```

O que será impresso ao final do programa?

(a) 0
(b) 5
(c) 10
(d) -1
(e) 4

---

16. **(Quadrix - 2018 - CRM-PR - Técnico em Tecnologia da Informação)**

A respeito de análise e desenvolvimento de sistemas, julgue o item subsequente.

Os algoritmos são sequências finitas de instruções que, quando corretamente executadas, levam à solução de um problema.

---

17. **(Quadrix - 2018 - CRM-PR - Técnico em Tecnologia da Informação)**

A respeito de análise e desenvolvimento de sistemas, julgue o item subsequente.

Em um fluxograma, as caixas de decisão são como "caixas pretas", uma vez que não se tem clareza da ação que será executada.

---

18. **(FAURGS - 2018 - TJ-RS - Programador)**

Instrução: A questão refere-se ao algoritmo abaixo, escrito em uma pseudolinguagem. Considere X um arranjo; length, uma função que devolve o tamanho do arranjo passado como parâmetro. A endentação demarca blocos de comandos.

```
1  for j=2 to length(X)
2      do      valor = X[ j ]
3              i = j-1
4              while i > 0 e X[ i ] > valor
5                      do      X[i+1] = X[ i ]
6                              i = i-1
7              X[i+1] = valor
```

Qual é a característica principal desse algoritmo?

(a) É baseado na utilização de recursividade.
(b) É controlado por comandos de desvio incondicional.
(c) Utiliza comandos de repetição.
(d) Pode permanecer em laço infinito.
(e) É baseado na abordagem dividir e conquistar.

---

19. **(FCC - 2016 - TRF - 3ª REGIÃO - Técnico Judiciário - Informática)**

Em uma linguagem de programação, os tipos primitivos de dados:

(a) são sempre verificados pelo compilador. Caso se extrapole a capacidade do tipo, um erro ocorre e o programa é abortado.
(b) mais comuns e mais utilizados são as matrizes e os registros.
(c) são associados a um descritor. Um descritor é uma estrutura de dados, que não ocupa espaço na memória, que armazena os atributos do tipo de dados.
(d) na forma de caracteres geralmente são armazenados como codificações numéricas, como o padrão UTF.
(e) inteiros são sempre representados como uma cadeia de caracteres. O caracter mais à esquerda representa o sinal positivo ou negativo.

---

20. **(FGV - 2015 - PGE-RO - Técnico da Procuradoria - Tecnologia da Informação)**

Analise o pseudocódigo mostrado a seguir.

```
var i: inteiro
var j: inteiro
para i:= 1 até 2
begin
     if i < 2
     then k=i*2
     else k=i

     para j:= i até k
     begin
         print (i+j)
     end
end
```

Sabendo-se que nesse código cada ocorrência do comando print produz uma linha na saída, está correto afirmar que o número de linhas produzidas é:

(a) 3
(b) 5
(c) 7
(d) 9
(e) 11

---

21. (FCC - 2015 - DPE-SP - Programador)

Considere o algoritmo em pseudocódigo no qual DIV calcula o quociente da divisão inteira e MOD o resto da divisão inteira:

```
Var  taxa, cinco, tres, quociente, resto: inteiro

Início
    imprima ("Digite um numero inteiro maior ou igual a 8: ")
    leia(taxa)
    quociente ← taxa DIV 5
    resto ← taxa MOD 5
    tres ← 0
    cinco ← 0
    caso resto seja
         0: cinco ← quociente
            tres ← 0
         1: cinco ← quociente -1
            tres ← 2
         2: cinco ← quociente -2
            tres ← 4
         3: cinco ← quociente
            tres ← 1
         4: cinco ← quociente -1
            tres ← 3
   fim caso
   imprima(taxa, cinco, tres)

Fim.
```

O algoritmo em pseudocódigo acima

(a) garante que o valor de entrada seja maior ou igual a 8 para que seja possível dividir a taxa por 5 e por 3.
(b) para o valor inicial da taxa = 22 finaliza com cinco= 2 e tres=4.
(c) determina o maior número de 5 e de 3 unidades cuja soma dá o valor da taxa.
(d) para o valor inicial da taxa = 17 finaliza com cinco= 3 e tres=2.
(e) sempre finaliza com valores da variável cinco maiores ou igual a 1, mas a variável tres pode ter valor 0.

22. **(FCC - 2015 - DPE-SP - Programador)**

Considere a função Divide apresentada em pseudocódigo.

```
Função Divide (N1, N2: inteiro): real
Var result: real

Início
   result  ←  N1/N2

   Retorne result
Fim
```

Em relação aos conceitos de função e à função Divide acima, é correto afirmar:

(a) Quando são passados valores para os parâmetros da função Divide, os valores são copiados para a função. Este tipo de chamada em que se faz apenas a cópia dos valores é denominado passagem de parâmetro por valor.
(b) Pode-se, no programa principal, usar o comando: imprima (Divide(5,0)) e este comando exibirá 0.
(c) Para chamar a função Divide no programa principal é necessário que sejam declaradas 2 variáveis globais do mesmo tipo e com os mesmos identificadores utilizados na função.
(d) Para chamar a função Divide no programa principal é necessário que seja declarada uma variável real para receber o resultado retornado pela função.
(e) Quando são passados valores para os parâmetros da função Divide, são passados os endereços das variáveis. Este tipo de chamada em que utilizam-se endereços é denominado passagem de parâmetro por valor.

---

23. **(CESGRANRIO - 2014 - Petrobras - Técnico(a) de Informática Júnior)**

Considere a situação abaixo.

```
Algoritmo faca_contas
inicio
numero = 3
para i de 1 ate 5, faca
      leia (X)
      se X > 4, entao faca numero ← numero + X
                caso contrario, faca numero ← numero - X
      fimse

fimpara
escreva (numero)
fimalgoritmo
```

Qual é a saída do algoritmo faca_contas para a entrada 7, 3, 5, 2, 3?

(a) 3
(b) 6
(c) 7
(d) 10
(e) 12

24. **(CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 9)**

Julgue o item seguinte a respeito da construção de algoritmos, dos conceitos de variáveis e de bloco de comandos e das estruturas de controle.

Durante a execução de um programa, o conteúdo de uma variável pode mudar ao longo do tempo, no entanto ela só pode armazenar um valor por vez.

25. **(CESPE - 2018 - ABIN - Oficial Técnico de Inteligência - Área 9)**

Julgue o item seguinte a respeito da construção de algoritmos, dos conceitos de variáveis e de bloco de comandos e das estruturas de controle.

Na lógica de programação, um bloco de comando é definido como um conjunto de ações para determinada função e tem como delimitadores as palavras reservadas INPUT e OUTPUT.

26. **(FCC – 2012 - TRE/SP – Técnico Judiciário - Programação de Sistemas)**

Analise o algoritmo a seguir:

```
Algoritmo Cálculo
var n, r, cont: inteiro
início
        r ← 1
        cont ← 1
        enquanto (cont<=5) faça
            leia(n)
            r ← r * n
            cont ← cont + 1
        fim_enquanto
        imprima (r)
fim
```

Se forem lidos os valores 2, 5, 7, 3 e 4,

(a) a saída será 840.
(b) haverá um erro, pois o resultado de um cálculo envolvendo a variável r não pode ser armazenado na própria variável r.
(c) a saída será 210.
(d) haverá um erro, pois o valor gerado será maior do que uma variável do tipo inteiro pode suportar.
(e) a saída será 0.

27.(UEL – 2011 - CMTU – Analista Administrativo – Tecnologia da Informação)

Observe o trecho do algoritmo a seguir

```
atribuir 50 a I
atribuir 0 a TOTAL
atribuir 0 a K

enquanto K < I faca

    inıcio
      somar 10 a K;
      atribuir TOTAL+K a TOTAL
      imprimir(K);
    fim;

fim-enquanto;

imprimir(TOTAL);
```

Ao final do processamento, a variável TOTAL e o número de vezes que a K será impressa são, respectivamente:

(a) 100 e 4
(b) 150 e 5
(c) 150 e 8
(d) 150 e 9
(e) 210 e 6.

28. **(FCC – 2012 - ARCE – Analista de Regulação - Analista de Sistemas)**

Considere o algoritmo em pseudocódigo:

```
Var A, B, C: lógico
Var X, Y: real
Var R1, R2, R3: inteiro
Início
    A ← verdadeiro
    B ← falso
    C ← verdadeiro
    X ← 2.5
    Y ← 3.5
    se ( C ou (X-Y > 0) )
      então
        R1 ← 1
      senão
        R1 ← 0

    se ( ( (A e B) ou C ) e ( (X > Y) ou não(A) ) )
      então
        R2 ← 1
      senão
       R2 ← 0

   se ( (B ou (X-Y <0)) e ( (B e não(A)) ou (não (C) ) )
     então
       R3 ← 1
     senão
     R3 ← 0
Fim
```

Os valores de R1, R2 e R3, após a execução do algoritmo são:

(a) R1=0, R2=1, R3=1
(b) R1=1, R2=1, R3=0
(c) R1=1, R2=0, R3=0
(d) R1=0, R2=0, R3=1
(e) R1=0, R2=0, R3=0

29. **(FCC – 2010 - TRF/4ª Região – Analista Judiciário – Informática)**

Considere:

```
algoritmo PROVA
var N,X,CONTA,CONTB,EXP,RESP: inteiro
inicio
    N ← 4
     X ← 2
    RESP← X
    para  CONTA←2 até N passo 2 faça
                EXP ← X
                CONTB ← 1
                para CONTB=1 até CONTA-1 passo 1 faça
                            EXP ← EXP * X
                fim_para
                RESP ← RESP + EXP
    fim_para
    imprima (RESP)
fim
```

Dado o algoritmo representado na forma de português estruturado, o valor de saída contido na variável RESP será

(a) 6.
(b) 22.
(c) 86.
(d) 0.
(e) 342

30. (FCC – 2012 - METRÔ/SP – Analista Desenvolvimento Gestão Júnior – Ciências da Computação)

Analise a estrutura abaixo.

```
Algoritmo Cálculo
var n, cont, resp, s: inteiro
início
    leia(n)
    resp ← 0
    cont ←1
    s ← -1
    enquanto (cont<=n) faça
        se (cont mod 2 = 0)
            então resp ← resp + cont * s
            senão resp ← resp + cont
        fim_se
        cont ← cont+1
    fim_enquanto
    imprima(resp)
fim
```

Considere mod o operador que calcula o resto da divisão entre dois números inteiros.

Por meio de um teste de mesa é possível constatar que o portugol mostrado representa a resolução da equação:

(a) resp = 0 + 1 – 2 + 3 – 4 + 5 – ... (+ ou –) n
(b) resp = 0 + 1 – 22 + 34 – 46 + 5n
(c) resp = 0 – 1 – 2 – 3 – 4 – 5 – ... – n
(d) resp = 0 + 1! – 2! + 3! – 4! + 5! – n!
(e) resp = 0 – 1 + 2 – 3 + 4 – 5 + ... – n

31.(CESPE – 2008 - SERPRO – Técnico – Programação)

A respeito da lógica de programação, que é fundamental para o desenvolvimento de códigos por meio de linguagens de programação, julgue os itens subsequentes.

Um algoritmo pode ser definido como uma sequência finita de passos que levam à execução de determinada tarefa ou conjunto de tarefas.

32.(CESPE – 2008 - SERPRO – Técnico – Programação)

A respeito da lógica de programação, que é fundamental para o desenvolvimento de códigos por meio de linguagens de programação, julgue os itens subsequentes.

Na lógica de programação, a instrução é o comando principal que indica a um programa uma condição estrutural a repetir.

33.(CESPE – 2008 - SERPRO – Técnico – Programação)

A respeito da lógica de programação, que é fundamental para o desenvolvimento de códigos por meio de linguagens de programação, julgue os itens subsequentes.

Um diagrama de blocos é uma forma padronizada para se representar os passos lógicos de determinado processamento. Por meio do diagrama, pode ser utilizada uma sequência de símbolos, com significado bem definido, para auxiliar a representação dos passos de um processamento.

---

34. (CESPE – 2008 - SERPRO – Técnico – Programação)

A respeito da lógica de programação, que é fundamental para o desenvolvimento de códigos por meio de linguagens de programação, julgue os itens subsequentes.

Os operadores relacionais são utilizados para comparar números. Para se comparar sequências de caracteres (strings), são utilizados os operadores lógicos, que retornam valores verdadeiro e falso.

35.(CESPE – 2008 - SERPRO – Técnico – Programação)

A respeito da lógica de programação, que é fundamental para o desenvolvimento de códigos por meio de linguagens de programação, julgue os itens subsequentes.

Em lógica de programação, uma constante é um valor fixo que não se modifica ao longo do tempo durante a execução de um programa. Essa constante pode ser numérica, lógica ou literal.

---

36. (CESPE – 2013 - SERPRO – Analista – Suporte Técnico)

Julgue os itens seguintes, relativos a programação e lógica de programação.

Parâmetros são pontos de comunicação entre módulos de um programa. A passagem de parâmetros, que consiste na substituição do parâmetro formal pelo parâmetro real, pode ser realizada por valor ou por referência.

37.(CESPE – 2013 - SERPRO – Analista – Suporte Técnico)

Julgue os itens seguintes, relativos a programação e lógica de programação.

Segundo o pseudocódigo abaixo, um vetor de 100 números é lido e, em seguida, é montado um segundo vetor a partir dos valores do primeiro vetor multiplicados por 3.

```
início
      VET1,VET2 : vetor [1..100] numérico
      CONTADOR : numérico
      para CONTADOR de 1 até 100 faça
            leia "Digite um número: ",VET1[CONTADOR]
            VET2[CONTADOR] ← (VET1[CONTADOR] * 3)
      fim-para
fim
```

---

38. (CESPE – 2011 - STM – Analista Judiciário – Análise de Sistemas – adaptada)

Com relação a algoritmos e lógica de programação, julgue os itens a seguir.

Na passagem de parâmetros por referência, o valor do parâmetro real é copiado para o parâmetro formal do módulo, preservando, assim, o valor original do parâmetro. Na passagem de parâmetros por valor, toda alteração feita nos parâmetros formais reflete-se nos parâmetros reais.

---

39. (CESPE – 2011 - STM – Analista Judiciário – Análise de Sistemas – adaptada)

Nas estruturas de controle, tais como as estruturas de seleção simples, compostas ou encadeadas, é necessário verificar as condições para a realização de uma instrução ou sequência de instruções.

---

40. (CESPE – 2012 - Banco da Amazônia – Técnico Científico – Análise de Sistemas)

O comando while utilizado em algoritmos implementa laços com teste antecipado de condições, testando a condição e, sendo ela verdadeira, executando o bloco de comandos.

---

41. (CESPE – 2012 - Banco da Amazônia – Técnico Científico – Análise de Sistemas)

Quando um break é encontrado dentro de um laço for, a execução do código é interrompida e o programa é finalizado.

---

42. (CESPE – 2009 - UNIPAMPA – Analista de Tecnologia da Informação – Rede e Suporte)

Em relação aos conceitos de lógica de programação utilizados para a construção de algoritmos, julgue os próximos itens.

Valores que sejam armazenados em variáveis locais de determinado procedimento ou função não podem ser utilizados em outros procedimentos ou funções.

---

43. (CESPE – 2009 - UNIPAMPA – Analista de Tecnologia da Informação – Rede e Suporte)

Estruturas de repetição permitem que uma sequência de comandos seja executada repetidamente até que determinada condição de interrupção seja satisfeita. É possível que, em determinada execução do algoritmo, a sequência de comandos não seja executada nenhuma vez.

---

44. (CESPE – 2009 - UNIPAMPA – Analista de Tecnologia da Informação – Rede e Suporte)

Registros são estruturas consideradas heterogêneas porque são compostos de dados que, apesar de serem logicamente relacionados, não têm necessariamente o mesmo tipo.

---

45. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

Julgue os itens seguintes, relativos à lógica de programação e construção de algoritmos.

Na definição de uma função, a passagem de parâmetros por referência possibilita que o valor de uma variável passado como argumento seja alterado na função, e sua alteração mantenha-se mesmo após a execução da função.

---

46. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

É possível implementar procedimentos cujos valores gerados podem ser armazenados em variáveis que garantem sua existência mesmo após o término da execução de tais procedimentos.

---

47. (CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)

Estruturas de repetição são usadas para que determinado bloco de comandos seja executado diversas vezes. A garantia de parada da repetição ocorre por meio de uma condição que é verificada a cada nova iteração. Dependendo do tipo de estrutura de repetição utilizado, o bloco de comandos é executado pelo menos uma vez.

---

48. **(CESPE – 2010 - Banco da Amazônia – Técnico Científico – Redes e Telecomunicações)**

Variáveis declaradas dentro de funções ou procedimentos são chamadas de variáveis locais e não são visíveis por outras funções. Por esse motivo, não é possível declarar variáveis que possam ser utilizadas por qualquer função de um programa.

---

49. **(FCC - 2012 - ARCE - Analista de Regulação - Analista de Sistemas)**

Há duas maneiras de se passar argumentos ou parâmetros para funções: por valor e por referência. Todas as afirmativas sobre passagem de parâmetros estão corretas, EXCETO:

(a) Na passagem por referência, o que é passado como argumento no parâmetro formal é o endereço da variável.
(b) Na passagem por valor, o valor é copiado do argumento para o parâmetro formal da função.
(c) Por exemplo, quando duas variáveis inteiras i1 e i2 são passadas por valor à função troca() chamada pelo programa principal, elas também são alteradas no programa principal.
(d) Na passagem por referência, dentro da função, o argumento real utilizado na chamada é acessado através do seu endereço, sendo assim alterado.
(e) Na passagem por valor, quaisquer alterações feitas nestes parâmetros dentro da função não irão afetar as variáveis usadas como argumentos para chamá-la.

---

50. **(FCC - 2011 - TRT - 1ª REGIÃO (RJ) - Analista Judiciário - Tecnologia da Informação)**

Considere:

(I) Sequência, decisão e iteração são as estruturas necessárias e suficientes para o desenvolvimento da programação de computadores.
(II) Uma sequência de passos, incluindo a forma como os dados serão armazenados no computador, permitindo que o problema possa ser resolvido de maneira automática e repetitiva.
(III) Cada instrução é traduzida para uma representação interna e interpretada pela simulação de funcionamento do processador, o que torna mais rápido o ciclo escrita-execução-modificação.
(IV) A sua aplicação divide e estrutura o algoritmo em partes fechadas e coerentes para evitar a repetição de uma sequência de comandos que é utilizada em várias partes do programa.

Em relação à programação de computadores, as definições contidas nos itens I, II, III e IV correspondem, respectivamente, à programação

(a) linear, ao fluxograma, à compilação e ao procedimento.
(b) modular, ao algoritmo, à constante e à interpretação.
(c) orientada a objetos, ao projeto lógico, ao fluxograma e à compilação.
(d) linear, ao algoritmo, à compilação e à função.
(e) estruturada, ao algoritmo, à interpretação e ao procedimento.

Gabarito

1. Certo
2. D
3. C
4. Certo
5. E
6. A
7. Certo
8. C
9. A
10. B
11. Errado
12. D
13. C
14. C
15. B
16. Certo
17. Errado
18. C
19. D
20. A
21. B
22. A
23. C
24. Certo
25. Errado
26. A
27. B
28. C
29. B
30. A
31. Certo
32. Errado
33. Certo
34. Errado
35. Certo
36. Certo
37. Errado
38. Errado
39. Certo
40. Certo
41. Errado
42. Errado
43. Certo
44. Certo
45. Certo
46. Certo
47. Certo
48. Errado
49. C
50. E

Resumo Direcionado

Características do **Algoritmo**:

Finito	Não pode ficar executando indefinidamente. Tem que ter um escape.
Bem definido	Não pode haver ambiguidade nem subjetividade nas instruções
Entrada	Parâmetros a partir dos quais é feito o processamento
Saída	Resultado do algoritmo. Pode ser a solução de um problema ou a execução de uma tarefa

Pseudocódigo é uma forma de escrever algoritmos utilizando palavras intuitivas da língua portuguesa para representar as instruções computacionais.

Constantes	Não mudam ao longo do algoritmo	Ambas são regiões de memória para armazenar valores
Variáveis	Mudam ao longo do algoritmo	

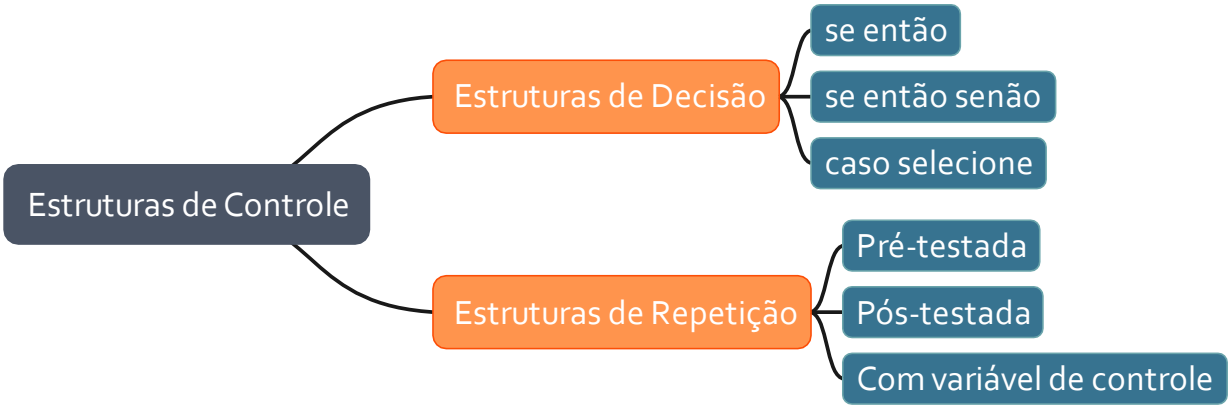
Tipos de dados Elementares	Descrição	Exemplos
Lógico	Possui 1 dentre 2 possíveis valores	VERDADEIRO/FALSO TRUE/FALSE 1/0, SIM/NÃO
Inteiro	Número sem parte fracionária	5 12
Real	Número com a parte fracionária	3,1415 10,50
Caractere	Dígito	'a', 'b', 'c', '$', '?', '4'

Tipos de dados Estruturados	Descrição	Exemplos
Vetor	Sequência de tamanho fixo de dados elementares do mesmo tipo	[1, 7, 4, 8, 23] [25.50, 46.30, 99.99]
String	Vetor de caracteres	"Polícia Federal"
Lista	Sequência de tamanho variável de dados elementares do mesmo tipo	(igual aos vetores, só que podendo aumentar ou diminuir de tamanho)

Operadores Aritméticos	Símbolo	Prioridade	Exemplo
Exponenciação	^	1ª	2 ^ 3 = 8
Multiplicação	*	2ª	3 * 4 = 12
Divisão	/	2ª	8 / 4 = 2
Módulo	%	2ª	10 % 3 = 1
Adição	+	3ª	2 + 7 = 9
Subtração	-	3ª	5 – 1 = 4

Operadores relacionais	Símbolo	Uso
Igual	=	x = y
Diferente	<> ou !=	x <> y ou x != y
Maior	>	x > y
Menor	<	x < y
Maior ou igual	>=	x >= y
Menor ou igual	<=	x <= y

Operadores lógicos	Significado
E / And	Combina 2 operandos booleanos retornando VERDADEIRO somente quando os 2 operandos são VERDADEIROS.
Ou / Or	Combina 2 operandos booleanos retornando FALSO somente quando os 2 operandos são FALSOS.
Não / Not	Operador unário (é aplicado sobre somente 1 operando) que inverte um valor booleano. Se o operando for VERDADEIRO o retorno é FALSO e vice-versa.



se-então

```
se (localDeNascimento = "Minas Gerais") então
        denominação = "Mineiro"
```

se-então-senão

```
se (condição = VERDADEIRO) então
        caminho = "cima"
senão
        caminho = "baixo"
```

caso-selecione

```
selecione (númeroMês)
        caso 1:
            mês = "Janeiro"
        caso 2:
            mês = "Fevereiro"
        caso outro:
            escreva("Você deve selecionar um número
de 1 a 12")
fim selecione
```

repetição pré-testada

```
enquanto (condição = VERDADEIRO) faça
        <bloco de comandos>
fim-enquanto
```

repetição pós-testada

```
faça
        <bloco de comandos>
enquanto (condição = VERDADEIRO)
```

repetição com variável de controle

```
para contador de 1 até n faça
        <bloco de comandos>
fim-enquanto
```

	Função	**Procedimento**
Recebe parâmetros?	Sim, mas não é obrigatório	Sim, mas não é obrigatório
Tem retorno?	Sim	Não

	Recursividade	**Iteratividade**
Vantagem	Facilidade da solução	Consome pouca memória
Desvantagem	Consome muita memória	Dificuldade da solução