Estratégia Concursos

Aula 00

****NÃO ATIVAR***Desenvolvimento p/ Prefeitura de Lucas do Rio Verde-MT (Analista em TI) Com Videoaulas - Pós-Edital*

Autor:
Equipe Informática e TI, Pedro Henrique Chagas Freitas

16 de Março de 2020

Sumário

Lógica de Programação 7

1 - Introdução ao desenvolvimento de software 7

2 - Compreendendo Constantes e Variáveis na programação 13

3 - Estruturas de controle: Decisão e Repetição 19

4 – Recursividade 33

Questões Comentadas 37

3 – Sistemas de numeração e aritmética computacional 101

Questões Comentadas 108

Lista de Questões - Lógica de programação 117

Lógica de programação 117

Lista de Questões - Sistemas de numeração 149

Gabarito - lógica de programação 153

Gabarito - sistema de numeração 154

APRESENTAÇÃO DO CURSO

Recuar não é uma escolha, quando se decide construir um sonho

Bem-vindo caro aluno (a)! Iniciamos nosso **curso com o objetivo de trazer o melhor material com** teoria e questões comentadas, voltado para provas **objetivas e discursivas** de concurso público. Para iniciar vamos partir para algumas reflexões:

Já fez **Check-in** rumo ao seu sonho? Já sentou na cadeira e embarcou rumo ao seu destino? Já escolheu o melhor caminho para chegar lá? Já pensou quantas escalas fazer e por qual companhia aérea voar?

A nossa vida parece muito com um saguão de aeroporto. Todo dia várias pessoas estão passando por nós, indo para vários destinos, esses destinos por sua vez, se originam de escolhas e nós fazemos escolhas todos os dias.



Quando temos o sonho de ingressar em **um bom concurso público** e provavelmente se você esta lendo este material em qualquer localidade do território nacional, deve ter esse sonho! Você precisa fazer check-in diariamente, rumo ao seu destino. Eu, enquanto seu professor e a Equipe de TI do **Estratégia Concursos**, estamos aqui para te ajudar nesse check-in e embarcar nessa viagem com você, tirando suas dúvidas e te preparando para a aprovação.

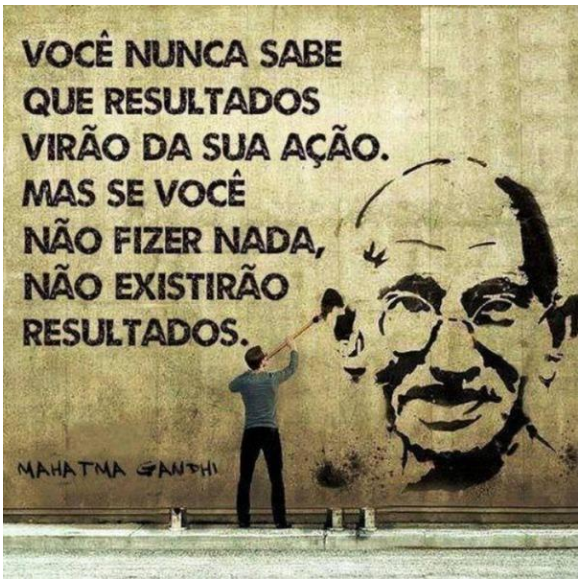
Todas as pessoas que chegaram a algum lugar, começaram de onde estavam. O que quero dizer com isso, é que de fato trabalhamos duro porque acreditamos no seu sonho de ingressar em uma boa carreira pública e estamos dispostos a te mostrar o caminho do sucesso, para alcançar a carreira que você tanto sonha!

Para isso quero te apresentar a nossa **aula demonstrativa**, você embarcando conosco nessa aula demonstrativa, vai poder desfrutar de uma viagem rumo a sua aprovação. Nesse caminho, quero te apresentar alguns conselhos que eu sempre gosto de dar.



Algumas pessoas me perguntam: *Quanto tempo leva até a aprovação?*

Eu já tenho alguns anos nessa estrada e já vi de tudo, já vi amigos meus ingressando no MPU com 6 meses de estudo, e também já vi outros amigos ingressando com 3 anos de estudo. A verdade é que não existe uma verdade sobre isso, o que existe são pessoas diferentes utilizando seu tempo, esforço, disciplina e fé de formas diferentes.



Conheço por exemplo pessoas excepcionais que não acreditam em si mesmas e aqui temos o **grande pulo do gato!** Você precisa ter tempo, esforço, disciplina e fé, mas antes de tudo isso **precisa acreditar em si mesmo!** Digo isso, porque muitos dos que param, não param pela dificuldade, mas por deixar de acreditar.

Tempo: Assim sendo, o tempo até sua aprovação vai depender do equilíbrio entre esses fatores: Tempo de qualidade nos estudos (*Pode ser 2 horas por dia? Sim. Pode ser 10 horas por dia? Sim. Desde que você absorva a matéria, mesmo que sejam 20 minutos por dia, precisa ser* ***tempo de qualidade****).*

Você já deve ter se deparado com aquele amigo seu, que estuda a 5 anos, 25 horas por dia e de fato existem pessoas assim, mas sinceramente eu não conheço ninguém que consiga realizar mais de 6 horas (de **qualidade nos estudos**), por isso gosto de sempre focar nisso, **você precisa ter tempo de qualidade nos seus estudos e não muito tempo para estudar.**



Esforço: **Esforço é a sua determinação em movimento**. Acredite não tem como chegar no lugar da vitória sem se esforçar muito, a propósito se você está começando nesse mundo dos concursos vai perceber que

tem muito conteúdo para você aprender, se já está nessa estrada vai lembrar que ainda não se tornou a melhor versão de você mesmo.

Mas nunca se esqueça: seu esforço vai até o dia da aprovação, às vezes pode ser difícil, mas quero garantir a você caro aluno (a), **vale a pena à luta!** Cada dia acordando cedo, cada resumo e principalmente cada noite de batalha ao lado do conteúdo para prova, resolvendo questões e se preparando! Tudo isso vai te levar ao lugar da aprovação, então mãos a obra, seu esforço está construindo o destino para onde você está indo! Se continuar nessa estrada dia após dia, eu te garanto uma coisa:

Você vai conseguir chegar a sua aprovação muito antes do que imagina.

Disciplina e Fé: Aprendi uma coisa estudando para concursos, a sua disciplina é o que te diferencia, qualquer pessoa pode se dedicar, mas nem todos serão **constantes** (**disciplinados**), sua memória deve sempre ser lembrada do conteúdo. Quem nunca se deparou com alguma questão e pensou:

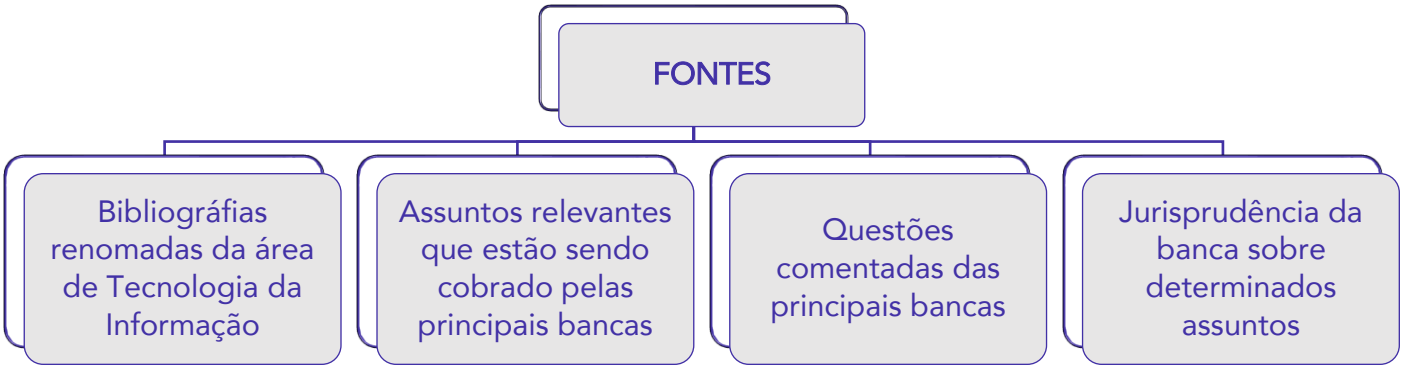
Nossa! Eu já vi isso antes. Você tenta se lembrar, então percebe que não guardou a informação que deveria ter guardado. Por isso, precisamos da **disciplina**, **manter a constância no objetivo traz o objetivo para perto de você**.



Sua Fé vai te ajudar nesse processo, eu pessoalmente sempre gosto de olhar as coisas com um propósito maior do que aquilo que estou vendo naquele momento. Quero te convidar a fazer a mesma coisa, toda vez que o cansaço aparecer ou qualquer outro fator, lembre-se **seu objetivo é a aprovação** e é para lá que você está indo, tenha fé e bom ânimo! **Você vai chegar lá!**

Algumas constatações sobre a metodologia são importantes!

Podemos afirmar que as aulas levarão em consideração as seguintes "fontes".



Para tornar o nosso estudo mais completo, é muito importante resolver questões anteriores para nos situarmos diante das possibilidades de cobrança. Traremos questões de todos os níveis.

Essas observações são importantes pois permitirão que possamos organizar o curso de modo focado, voltado para acertar questões objetivas e discursivas.

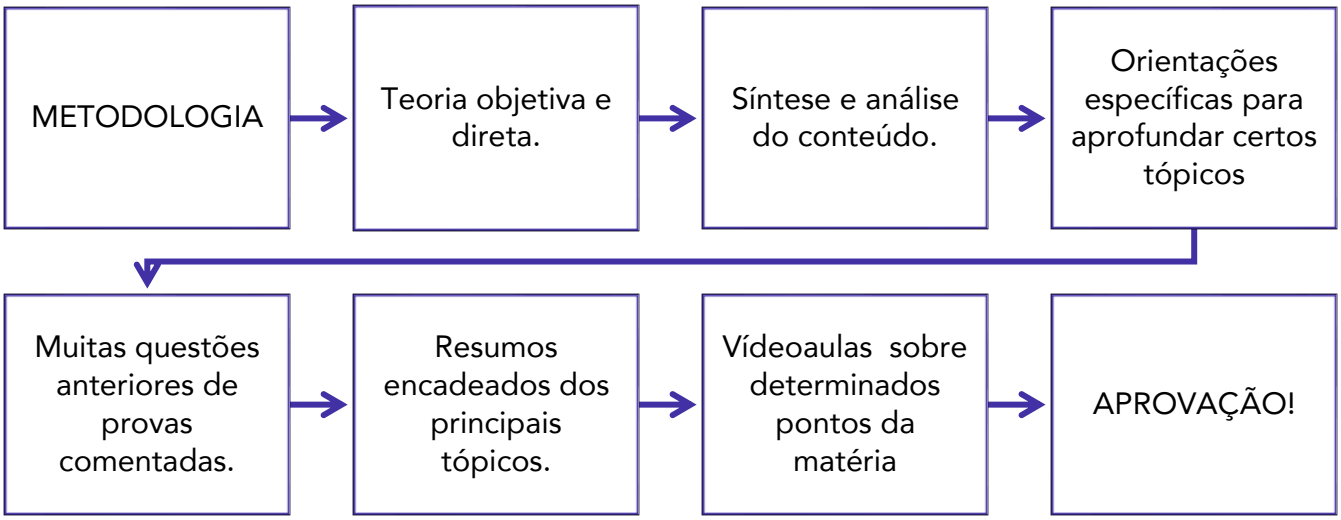
As aulas em *.pdf* tem por característica essencial a **didática**. Logo, o curso todo se desenvolverá com uma leitura de fácil compreensão e assimilação. Isso, contudo, não significa superficialidade. Pelo contrário, sempre que necessário e importante os assuntos serão aprofundados. A didática, entretanto, será fundamental para que diante do contingente de disciplinas, do trabalho, dos problemas e questões pessoais de cada aluno, possamos extrair o máximo de informações para hora da prova.

Para tanto, o material será permeado de **esquemas**, **figuras informativas**, **resumos**, **etc**, tudo com a pretensão de "chamar atenção" para as informações que realmente importam. Com essa estrutura e proposta pretendemos conferir segurança e tranquilidade para uma **preparação completa**, **sem necessidade de recurso a outros materiais didáticos**.

Finalmente, destaco que um dos instrumentos mais relevantes para o estudo em .PDF é o **contato direto e pessoal com o Professor, através do fórum de dúvidas**. Aluno nosso não vai para a prova com dúvida! Por vezes, ao ler o material surgem incompreensões, dúvidas, curiosidades, nesses casos basta acessar o computador e nos escrever. Assim que possível respondemos a todas as dúvidas

Além disso, teremos **videoaulas!** Que serão conduzidas por outro Professor, diga-se de passagem: O cara é TOP e possui um didática incrível! Essas aulas destinam-se a complementar a preparação. Quando estiver cansado do estudo ativo (leitura e resolução de questões) ou até mesmo para a revisão, abordaremos alguns pontos da matéria por intermédio dos vídeos. Com outra didática, você disporá de um conteúdo complementar para a sua preparação. Ao contrário do PDF, evidentemente, **AS VIDEOAULAS NÃO ATENDEM A TODOS OS PONTOS QUE VAMOS ANALISAR NOS PDFS**. **Por vezes, haverá aulas com vários vídeos; outras que terão videoaulas apenas em parte do conteúdo; e outras, ainda, que não conterão vídeos. Nosso foco é, sempre, o estudo ativo!**

Assim, cada aula será estruturada do seguinte modo:



APRESENTAÇÃO PESSOAL

Por fim, resta uma breve apresentação pessoal. Meu nome é Pedro Henrique Chagas Freitas! Sou graduado em Engenharia de Computação, pós-graduado em Gestão e Desenvolvimento de Sistemas e Mestre em Gestão do Conhecimento e Tecnologia da Informação.

Estou envolvido com concurso público há 8 anos, aproximadamente, quando ainda na faculdade. Trabalhei no Ministério da Economia, Ministério da Agricultura, Ministério da Educação, Ministério da Cidadania, Ministério dos Direitos Humanos e no Instituto Nacional de Tecnologia da Informação no cargo de ATI. Fui aprovado para o cargo de Analista de Tecnologia da Informação do Ministério do Planejamento, que veio a se tornar Ministério da Economia e também fui aprovado nos concursos de Técnico em Telecomunicações da ANATEL, Agente Administrativo do MTE, Analista de Sistemas do MEC, Analista de Tecnologia da Informação da FUB e 1º para o cargo de Professor de Informática da Secretaria de Educação do Distrito Federal.

Quanto à atividade de professor, leciono exclusivamente para concursos, com foco na elaboração de materiais em *pdf*, mas também já lecionei disciplinas para graduação de Engenharia Elétrica e Sistemas de Potência. Além disso tenho um grande orgulho de fazer parte do Time de Professores de TI do Estratégia Concursos, que sem dúvida é hoje o melhor site de material para concursos públicos.

Deixarei abaixo meus contatos para quaisquer dúvidas ou sugestões. Terei o prazer em orientá-los da melhor forma possível nesta caminhada que estamos iniciando.

E-mail: professorpedrofreitas@gmail.com

Cursos Estratégia: https://www.estrategiaconcursos.com.br/cursosPorProfessor/pedro-henrique-chagas-freitas-4000/

LÓGICA DE PROGRAMAÇÃO

1 - Introdução ao desenvolvimento de software

Vamos lá! Antes de entrarmos nesse mundo do desenvolvimento de software, precisamos entender onde foi o grande big bang do desenvolvimento de software, afinal de contas o que é o desenvolvimento de software e por que precisamos desenvolver software?

The Big Bang Theory do Desenvolvimento de Software



Há muito tempo.. em uma galáxia muito, muito distante o ser humano tentava de diversas **formas automatizar processos fundamentais**, como por exemplo: o cálculo matemático. Governos e Universidades de todo o mundo, se empenhavam em **produzir componentes lógicos capazes** de implementar **rotinas em circuitos integrados**.

Começaram a surgir então **sequências numéricas** que foram chamadas de **instruções** que tinham por atribuição **manipular dados,** até que esses dados se tornassem **informações**. Essas instruções por sua vez reunidas formavam **funções,** que basicamente representavam uma **entrada e uma saída**.

Os estudiosos da época se concentravam principalmente nas universidades e chegaram a uma conclusão bastante interessante:

*Precisamos separar os **conceitos de hardware** (meio físico) **do conceito de software** (meio lógico). Tivemos então na época dos anos 40 a primeira definição de programa olhando exclusivamente para o **conceito lógico** por trás de um conjunto de instruções.*

Mas a pergunta então foi: *Como vamos executar essas instruções?*

Nasceu então o conceito de **linguagem de máquina**, a fim de definir como os processadores da época compreenderiam um **conjunto lógico de instruções**. Conforme os anos foram passando, essa interpretação da linguagem de máquina foi evoluindo, passando inclusive pelo que hoje conhecemos como

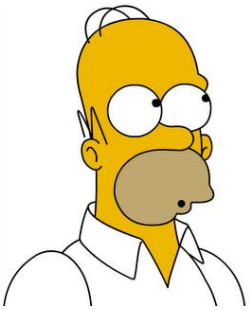
interpretadores, exemplo: JVM (Java Virtual Machine), que é responsável por converter bytecodes em código interpretável e executável.

Quando os computadores já estavam compreendendo bem e processando a **linguagem de máquina (bits: 0 e 1),** a interpretação dessas instruções começou a crescer para um nível mais alto de compreenção, veio então o que compreendemos por **linguagem de programação**.

As linguagens de programação surgiram com a intenção de se aproximar da linguagem humana. *Por que Professor?* Porque a linguagem de máquina é muito pouco intuitiva, o que dificulta ela ser utilizada para desenvolver um **conjunto de instruções lógicas (programas).**

Mas Professor como podemos desenvolver um conjunto de instruções lógicas?

Através da **sintaxe** que vem evoluindo junto com as linguagens. Quando desejamos criar uma linguagem de programação definimos uma sintaxe, que nada mais é que um **conjunto de regras para definir expressões que serão válidas dentro da linguagem**, ou seja expressões por exemplo que são válidas em Java podem não funcionar em COBOL. Ok?



Lembre-se então que Software é um **conjunto de instruções**, ordenadas de **forma lógica** com o objetivo de **executar funções específicas** (tarefas) com a finalidade de criar um software **executável**.

Note então, que um software **não é só** um conjunto de instruções, porque estas instruções **precisam estar ordenadas,** mas não basta somente existir ordenação, essa ordenação precisa **refletir uma lógica** que tenha o objetivo de executar **funções específicas, isto é "tarefas".**

Vamos então compreender como a lógica de programação, reflete funções específicas (tarefas) dentro de um software.

Quase todo problema computacional reflete um problema do mundo real. Isso significa que quando resolvemos problemas no mundo real, expressamos alguma lógica para resolvê-los. Dessa forma, quando nos deparamos com esses problemas do mundo real, dentro do mundo computacional, também precisamos empregar uma lógica para resolver esses problemas.

Essa lógica é baseada no que conhecemos por instruções, logo para resolver um problema no mundo computacional eu vou precisar designar instruções específicas em conjunto.

Essas instruções específicas em conjunto vão formar o que nós chamamos de **algoritmos**, que nada mais é que um **conjunto definido de instruções**.

Vamos ver um exemplo:

Vamos elaborar um algoritmo que faça a soma de três números inteiros, vou começar declarando um algoritmo que vou chamar aqui de somatresnumeros. Ok?

```
Algoritmo somatresnumeros

Variavel n1, n2, n3, resultado: inteiro

Inicio

Ler n1

Ler n2

Ler n3

Resultado <- n1 + n2 + n3

Escrever Resultado

fim
```

```
Programa somatresnumeros;

Var n1, n2, n3, result: integer;

Begin

        Read (n1);

        Read (n2);

        Read (n3);

        Result = n1 + n2 + n3;

        Write (result);

end
```

Na esquerda em azul temos um algoritmo que representa a soma de três números e na direita em roxo temos o programa (linguagem de programação) que representa o algoritmo em azul. Certinho?

Perceba que quando eu crio um algoritmo eu estou dizendo de maneira lógica, como o meu código vai ser descrito, é como uma receita de como fazer. De forma estruturada temos no algoritmo uma sequência de instruções, que objetivam entregar um resultado. **Logo, um software é a representação codificada de um algoritmo.**

A necessidade então de representar algoritmos através de uma linguagem de programação fez surgir diversos tipos de linguagens, como: **COBOL, Java, C++, C#, etc**.

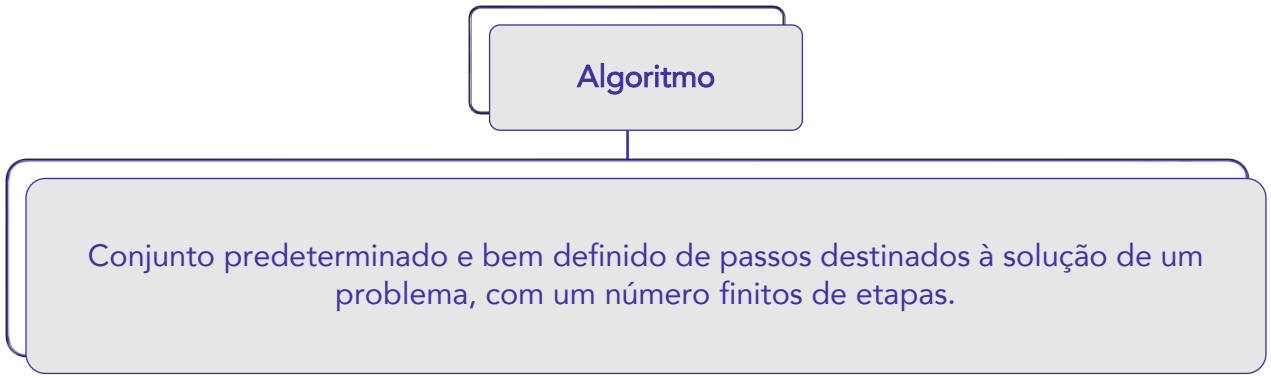
Quando estamos desenvolvendo costumamos atribuir o termo código fonte, ao código que desenvolvemos a fim de representar um algoritmo, da mesma forma, o algoritmo muitas das vezes é chamado de **pseudocódigo**! Um pseudocódigo (algoritmo) tem por sua vez a finalidade de descrever a lógica de forma simples, sem a necessidade da sintaxe de uma linguagem de programação.

Alguns autores entendem o pseudocódigo como um intermediário entre a linguagem natural (linguagem humana) e a linguagem de programação. Eu pessoalmente, não entendo o pseudocódigo como uma linguagem, mas como a representação de uma linguagem. Sigamos em frente!

RESUMO

Primeiro lugar, por que chamamos de lógica? Porque é necessário utilizar a lógica para resolver um problema computacional. *Como assim?* **Precisamos de um encadeamento ou uma sequência de pensamentos para alcançar um determinado objetivo**. Nós podemos descrever esses pensamentos como uma sequência de instruções ou passos.

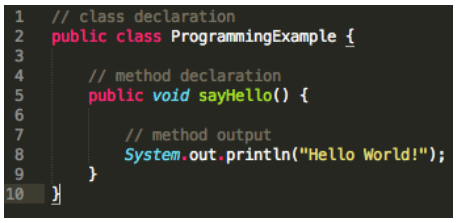
Professor, o que seria uma instrução? **É um conjunto de regras ou normas definidas para a realização ou emprego de algo, indicando ao computador uma ação elementar a ser executada**. Galera, um computador não pensa, ele é burro, ele recebe ordens e executa! O programador de computador é o camarada que vai dar ar ordens. Quando temos um conjunto de instruções, elas formam um algoritmo:



Professor, você pode dar um exemplo? Sim, o exemplo mais comum da bibliografia é uma receita de bolo. Observem que para fazer um bolo (solucionar um problema), é necessário seguir uma sequência de passos finitos e predeterminados. **No fim das contas, grosso modo, um software nada mais é do que a representação de um algoritmo**.

Professor, todos os programas que eu utilizo no meu computador são representações de algoritmos? Sim! *Inclusive o joguinho de Paciência que eu curto?* Sim! *Mas até mesmo os apps que eu utilizo no celular?* Eles também! **Todos os softwares (de desktop, notebook, smartphone, tablet, geladeira, relógio, foguete da NASA, entre outros) são representações de algoritmos**.

Então, basta que eu escreva um conjunto de passos em qualquer língua que o meu computador realiza a tarefa que eu quiser? Claro que não! Computadores não entendem, por exemplo, português – eles entendem o e 1 (na verdade meeeeesmo, eles entendem presença ou ausência de tensão elétrica), **portanto é necessário representar esses algoritmos por meio de uma linguagem de programação**.



Como assim, professor? Pessoal, um computador é uma grande calculadora. No entanto, ele é "burro", ele só calcula o que o mandam calcular. **As linguagens de programação surgem como uma solução para abstrair a comunicação entre seres humanos e computadores**. Na imagem ao lado, a ordem do programador era: "*Computador, escreva na tela: Hello World!*".

Bem, acho que todo mundo já ouviu falar alguma vez na vida em Código-Fonte. Se você não sabe o que é um Código-Fonte, abra o Internet Explorer ou Google Chrome ou Mozilla Firefox, entre em qualquer site que você queira e pressione a Tecla F12. Pronto, você verá o Código-Fonte por trás do site bonitinho que você está vendo...

Todo software ou site possui um código-fonte, que é um conjunto de palavras organizadas de acordo com regras específicas, formando um ou mais algoritmos. Essas palavras que formam o algoritmo são escritas utilizando uma linguagem de programação. Esse código-fonte é traduzido e posteriormente executado pelo usuário.

Pessoal... se eu não souber uma linguagem de programação, eu posso escrever um algoritmo utilizando um pseudocódigo ou pseudolinguagem! *O que é isso?* É uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. **Trata-se de um pseudocódigo, logo não pode ser executado em um sistema real**.

Um tipo de pseudocódigo é o Portugol (ou Português Estruturado)! Trata-se de uma simplificação extrema da língua portuguesa, limitada a pouquíssimas palavras e estruturas que têm significado pré-definido, na medida em que deve seguir um padrão. **Emprega uma linguagem intermediária entre a linguagem natural e a linguagem de programação para descrever os algoritmos**.

Embora o Portugol seja uma linguagem bastante simplificada, possui todos os elementos básicos e uma estrutura semelhante à de uma linguagem de programação de computadores. Portanto **resolver problemas com português estruturado pode ser uma tarefa tão complexa quanto a de escrever um programa em uma linguagem de programação qualquer**. Olha um exemplo:

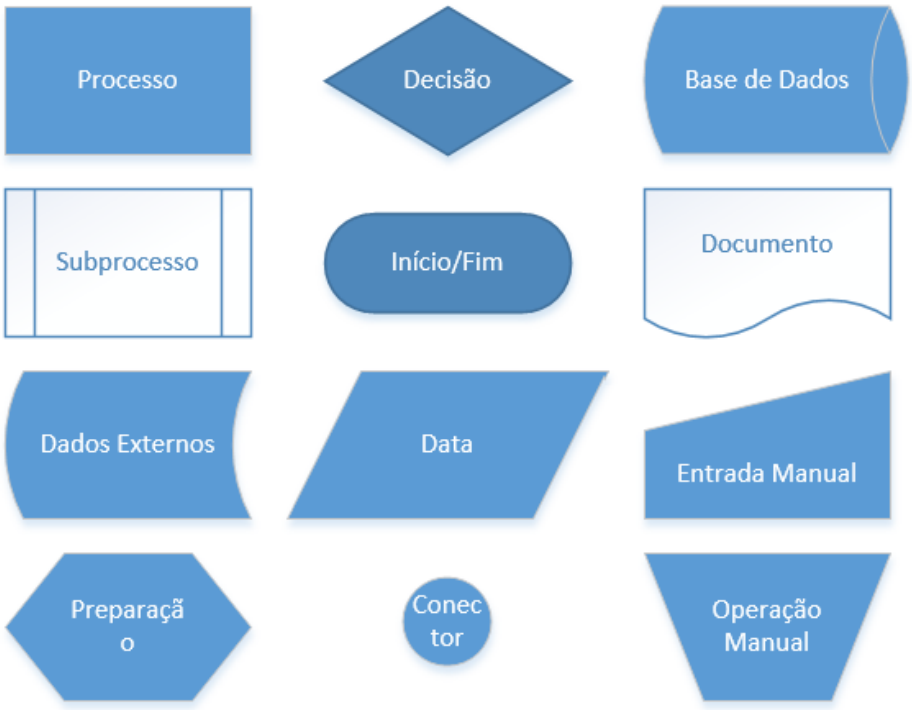
```
início
    <instruções>

    se <teste> então
        <instruções>
```

```
    senão
        <instruções>
    fim_se
fim
```

Além do português estruturado, é possível representar um algoritmo também por meio de um Fluxograma! *O que é isso, professor?* É uma espécie de diagrama utilizado para documentar processos, ajudando o leitor a visualizá-los, compreendê-los mais facilmente e encontrar falhas ou problemas de eficiência.

Os principais símbolos de um Fluxograma são:



Bem, nós vimos então que podemos representar algoritmos por meio de Linguagens de Programação, Pseudocódigos ou Fluxogramas! Em geral, os fluxogramas são mais utilizados para leigos; pseudocódigo para usuários um pouco mais avançados; e linguagens de programação para os avançados. **Agora vamos falar de conceitos mais específicos: constantes, variáveis e atribuições**!

2 - Compreendendo Constantes e Variáveis na programação

Constantes caro aluno (a) são dados que **não vão variar no tempo**, ou seja, as constantes sempre **terão um valor fixo.** A **variável** por sua vez de forma técnica é um espaço na memória que pode guardar **dados variáveis**.

Por exemplo: eu desejo armazenar uma variável X na memória RAM do meu computador.

Primeiro uma **posição para essa variável será estabelecida**, após isso o dado que eu desejo armazenar será **atribuído para aquela posição** na memória RAM. Ok?

A variável por sua vez vai ter um nome, assim como a constante, que será responsável por **abstrair o endereço de memorial no qual aquela variável ou constante estar localizada**.

Quando alteramos o conteúdo de uma variável, a atribuição desse conteúdo que estava associado a variável é apagado e um novo conteúdo é armazenado no endereço de memória anteriormente ocupado.

Quando eu desejo atribuir então um conteúdo (valor) para uma variável, eu apresento **a notação de uma seta para a esquerda**. Observe:

Nome ← Pedro

Aula ← Lógica de Programação

Dia ← 23

Nos deparamos então com dois tipos de dados**: Os dados estruturados e os dados elementares**. Os ***dados estruturados*** são também conhecidos como **compostos** e os ***dados elementares*** são conhecidos como **simples ou primitivos.**

Dados Estruturados: Tem por finalidade representar o conceito de decomposição, por exemplo: Pedro, pode ser decomposto em 'P' 'e' 'd' 'r' 'o'.

Aqui temos 5 caracteres, que unidos representam um nome. Dentro da classificação de dados estruturados, temos a **String** ou Cadeia de Caracteres, que são representações de sequências de caracteres. Logo aqui temos: frases, palavras, nomes, etc.

Dados Elementares: É o oposto dos dados estruturados, são chamados elementares porque **não podem ser decompostos**. Exemplos de dados elementares são o **Integer** (Inteiro) que são números inteiros que não podem ser fracionados, o **Float** que são números com parte fracionária, o **Char** que representa letras, dígitos e símbolos, o **Boolean** que é uma representação de um estado lógico: ligado/desligado, verdadeiro/falso, sim/não, etc.

Dentro desse contexto, temos também os **operadores aritméticos** (*, /, +, -, ^) e os **operadores relacionais** (=, !=, >, <, >=, <=). Os operadores aritméticos são utilizados para obter resultados

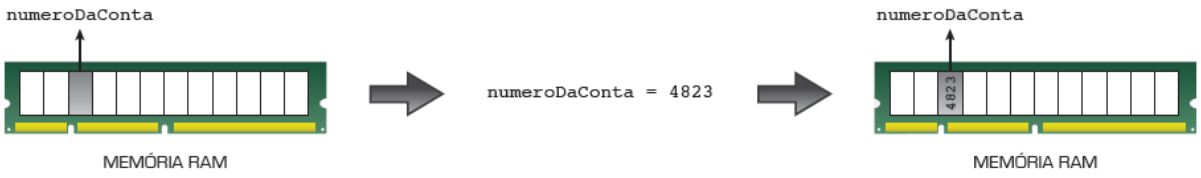
numéricos e os operadores relacionais são utilizados para comparar números e valores literais, a fim de retornar valores lógicos.

Temos também os **operadores lógicos** que são utilizados para combinar resultados de expressões, a fim de retornar valores lógicos (AND, OR, NOT).

INDO MAIS FUNDO

Constantes são dados que simplesmente não variam com o tempo, i.e., possuem sempre um valor fixo invariável. Por exemplo: a constante matemática π é (sempre foi e sempre será) igual a 3.141592 – esse valor não mudará! *Professor, e o que seria uma variável?* São espaços na memória do computador reservados para guardar informações ou dados que podem variar.

Em geral, variáveis são associadas a posições na Memória RAM, armazenando diversos tipos de dados que veremos com detalhes à frente! Ela possui um nome identificador que abstrai o nome do endereço na memória que ela ocupa. Observem a imagem abaixo: existe uma variável (espaço em memória) chamada numeroDaConta em que se armazena o valor 4823.



O conteúdo de uma variável pode ser alterado, consultado ou apagado diversas vezes durante a execução de um algoritmo, porém o valor apagado é perdido. *Bacana, mas e a atribuição?* Bem, **trata-se de uma notação para associar um valor a uma variável, i.e., armazenar o conteúdo no endereço de memória específico**. Cada linguagem de programação adotará uma maneira de representá-la.

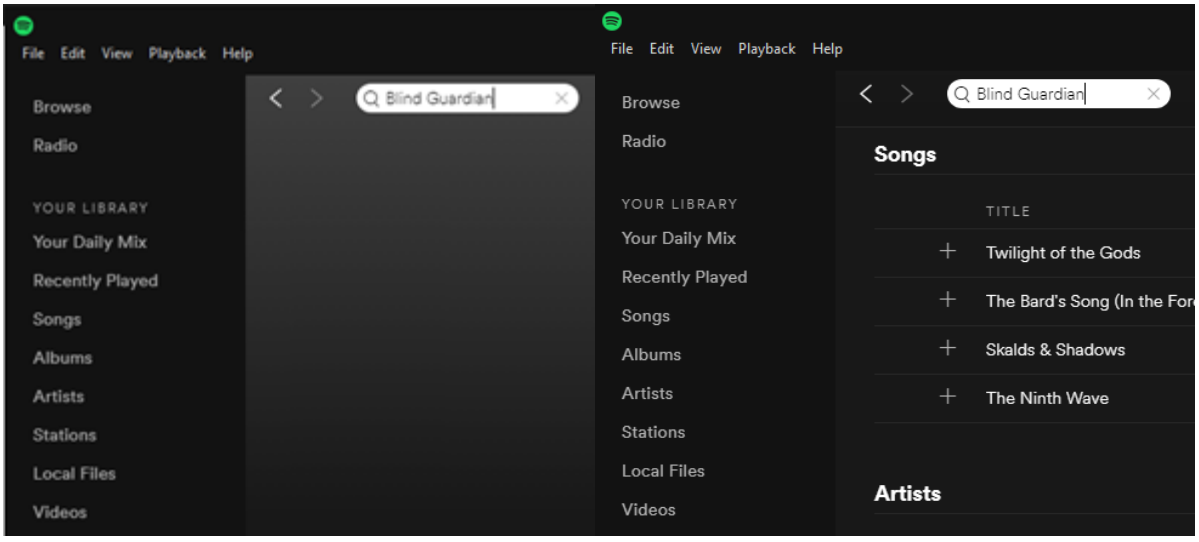
Galera, nós não vamos nos prender a uma linguagem de programação específica, vamos adotar o Português Estruturado, também conhecido – como dito anteriormente – Portugol. *Como é a notação de atribuição?* A notação de atribuição é representada por uma seta apontando para a esquerda do valor para o identificador, podendo ser:

```
Constante:     dataDeNascimento ← 1988
```

```
Variável:       endereço ← cidade

Expressão:      idade ← (anoAtual – anoDeNascimento)
```

E a Entrada/Saída de dados? **Galera, um modelo computacional é baseado em uma ENTRADA → PROCESSAMENTO → SAÍDA**. Entradas e saídas fazem parte da interação do programa com o mundo real, i.e., a forma com que o programa recebe os dados a serem processados do mundo real e devolve uma resposta. Esse é um conceito básico da primeira aula de um curso de computação.



Vamos falar agora sobre tipos de dados! Em geral, eles se dividem **em dois grupos: Dados Elementares e Dados Estruturados**. Só uma informação antes de começar: os dados elementares também podem ser chamados de simples, básicos, nativos ou primitivos. Já os dados estruturados também podem ser chamados de compostos. *Bacana?* Vamos para as definições!

- **Tipos Elementares:** são aqueles que não podem ser decompostos. *Ora, se eu disser que o Pedrinho tem 10 anos, é possível decompor esse valor de idade?* Não, logo é um tipo elementar. Há diversos tipos elementares, dependendo da linguagem de programação utilizada. No entanto, os principais são:

    - **Inteiro:** também conhecido como Integer, são similares aos números inteiros da matemática, i.e., sem parte fracionária. Podem ser positivos, negativos ou nulos. Ex: -2% de crescimento do PIB; 174 km de distância; 0 ºC de Temperatura; etc.

    - **Real:** também conhecido como Float (Ponto Flutuante), são similares aos números reais da matemática, i.e., possuem parte fracionária. Ex: 3,141592 é a constante de PI; 9,81 m/s² de Aceleração Gravitacional; raiz quadrada de 7; etc.

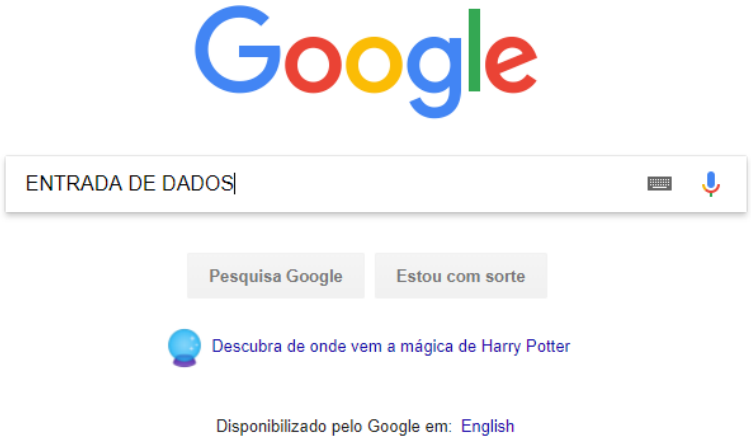
- **Caractere:** também conhecido como Literal ou Char, são representações de letras, dígitos e símbolos. Quando colocadas em conjunto, formam um tipo estruturado chamado String ou Cadeia de Caracteres. Ex: 'a', '$', '5', 'D', etc.

- **Lógico:** também conhecido como Boolean, são representações de valores lógicos – verdadeiro/falso, ligado/desligado, sim/não, etc. São extremamente importantes na programação, principalmente na verificação de condições.

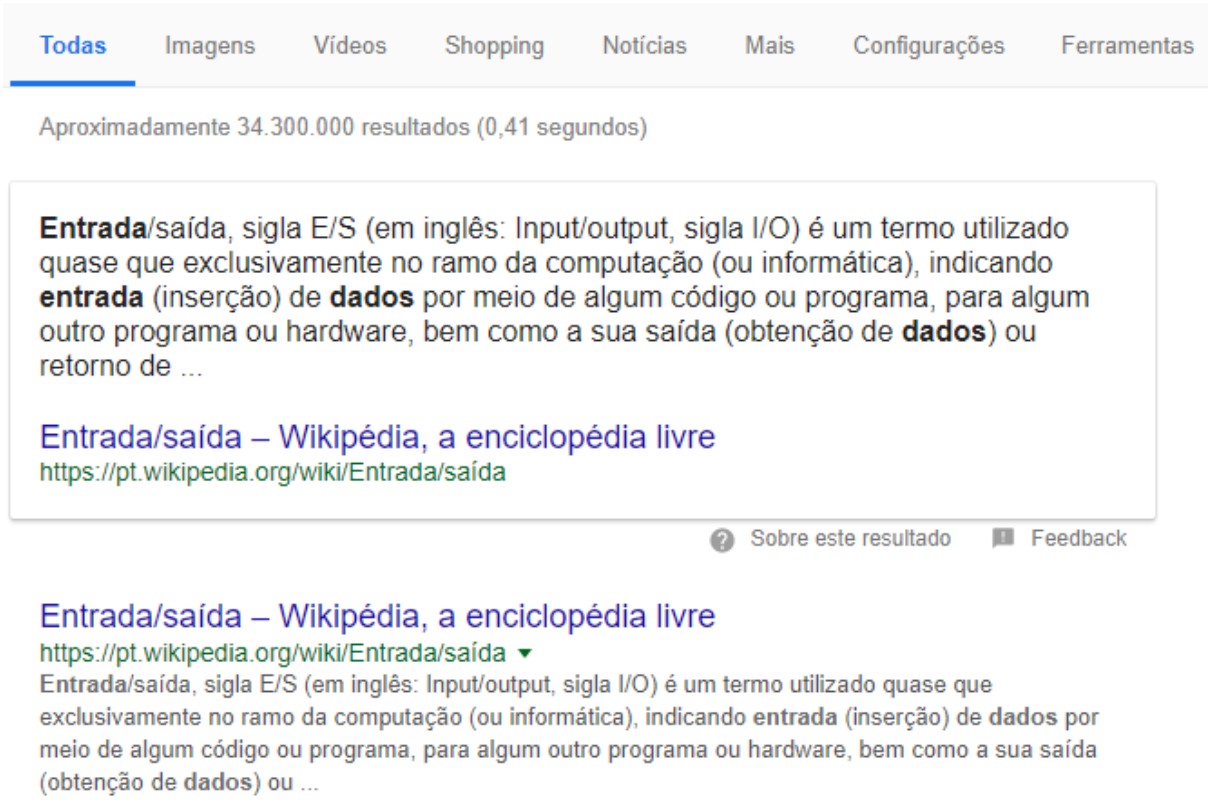
- **Tipos Estruturados:** são aqueles que podem ser decompostos. *Ora, se eu disser que o nome da bola da copa é Brazuca, é possível decompor esse nome?* Sim, basta dividi-lo em caracteres: 'B', 'r', 'a', 'z', 'u', 'c', 'a'. Há infinitos tipos estruturados[1], pois eles são a combinação de vários outros, o mais comum é:

  - **Cadeia de Caracteres:** também conhecido como String, são representações de sequências de caracteres, incluindo ou não símbolos. Pode ser uma palavra, frase, código, etc, por exemplo: "O rato roeu a roupa do rei de Roma".

Pessoal, quase todas as linguagens de programação possuem instruções para Leitura e Escrita de dados. A Leitura é uma Entrada de Dados (Input) que busca ler dados do usuário por meio teclado geralmente (Ex: busca do Google). A Escrita é uma Saída de Dados (Output) que busca mostrar informações ao usuário na tela do computador (Ex: resultado da busca do Google).



[1] Uma música em .mp3, um texto em .pdf, uma imagem em jpg são todos tipos estruturados.



Professor, e como eu faço para manipular dados? **Temos alguns operadores:**

Operadores Aritméticos: são utilizados para obter resultados numéricos, preocupando-se com a priorização[2].

Operador	Símbolo	Prioridade
Multiplicação	*	2º
Divisão	/	2º
Adição	+	3º
Subtração	-	3º
Exponenciação	^	1º

Operadores Relacionais: são utilizados para comparar números e literais, retornando valores lógicos.

[2] Em operadores que possuem a mesma prioridade, o que aparecer primeiro deve ser priorizado! Além disso, parênteses possuem sempre a maior prioridade!

Operador	Símbolo
Igual a	=
Diferente de	<> ou !=
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Operadores Lógicos: servem para combinar resultados de expressões, retornando valores lógicos (verdadeiro ou falso).

Operador/Símbolo
E/And
Ou/Or
Não/Not

Por fim, antes de passarmos para as estruturas de controle, vamos entender o que é um Bloco de Comandos. Um Bloco de Comandos é um conjunto de comandos limitados por dois delimitadores que marcam o início e o fim do bloco. O bloco pode um ou mais comandos (quando tem apenas um, é opcional utilizar os dois delimitadores) – geralmente é: {...} ou **begin** ... **end**.

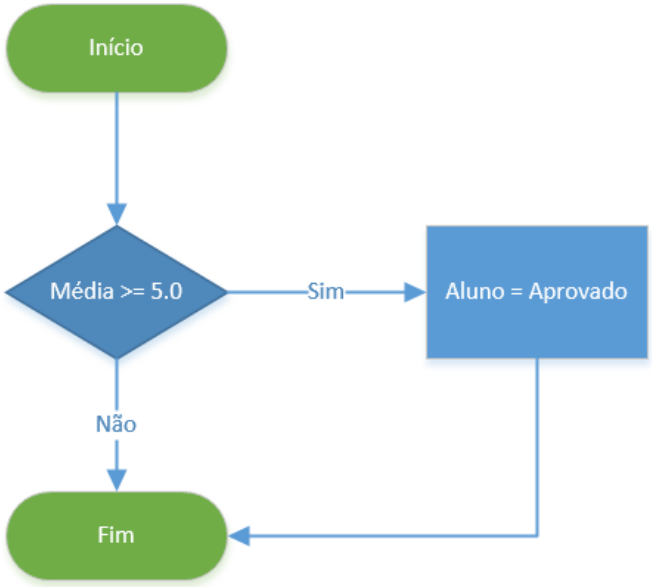
3 - Estruturas de controle: Decisão e Repetição

Bacana! Com isso, já podemos falar sobre Estruturas de Decisão e Repetição! Galera, as Estruturas de Decisão (também chamadas de Estruturas de Seleção, inclusive no edital) permitem interferir na sequência de instruções executadas dependendo de uma condição de teste, i.e., o algoritmo terá caminhos diferentes para decisões diferentes. *Bacana?*

O contrário dessa afirmação é a execução sequencial das instruções, sem loops ou desvios. Quando terminarmos, todos devem saber cantar "Hey Jude" na ordem certinha, seguindo os comandos que nós estamos aprendendo no fim da aula! *Beleza?* A melhor maneira de se visualizar uma estrutura de decisão é com o uso de fluxogramas. Então, vamos ver...

CASO 1:

```
Se (Média >= 5.0) Então
    Aluno = Aprovado
```

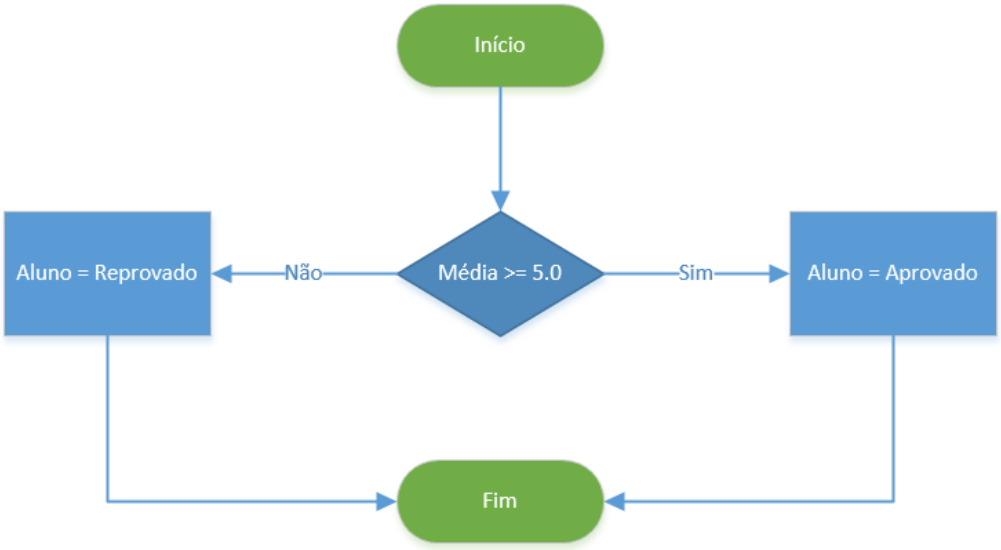


O primeiro caso, e mais simples, é uma estrutura de decisão com um teste (Média >= 5.0). O programa avalia se a variável Média tem o valor maior ou igual a 5.0, no caso de o teste ser verdadeiro a instrução "Aluno = Aprovado" é executada. Em caso negativo, o programa pula a instrução "Aluno = Aprovado" e continua logo após o final do bloco de decisão.

Quando uma estrutura de decisão (seleção) possui somente o bloco "Se Então", é chamada de simples. Isso já foi cobrado em provas da FCC, galera! Prestem bastante atenção nesse quesito! Algumas provas também cobram os nomes das estruturas em inglês, ou seja, ao invés de "Se Então", eles também podem cobrar como "If Then". *Tudo tranquilo?*

CASO 2:

```
Se (Média >= 5.0) Então
     Aluno = Aprovado
Senão
     Aluno = Reprovado
```

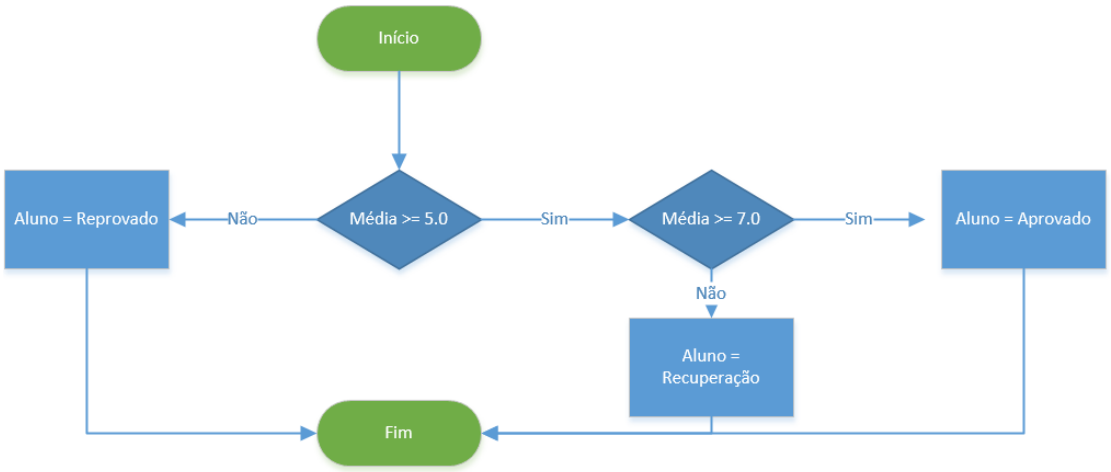


O segundo caso é só um pouquinho mais complexo, onde temos dois fluxos possíveis, um que passa pelo comando "Aluno = Aprovado" e outro que passa pelo "Aluno = Reprovado". **Importante ressaltar que, nesse caso, sempre alguma dessas instruções serão executadas**, pois, ou Média é maior ou igual a 5.0 ou é menor, sempre. *Bacana?*

Esse exemplo de estrutura de decisão possui os blocos Se-Então-Senão. **Quando a estrutura de decisão possui todos os blocos (Se-Então-Senão), ela é chamada de composta.** Em inglês a estrutura composta é conhecida como "If-Then-Else". Então, nós vimos já o Se-Então, agora vimos o Se-Então-Senão e agora vamos ver o Se-Então-Senão dentro de outro Se-Então-Senão.

CASO 3:

```
Se (Média >= 5.0) Então
        Se (Média >= 7.0) Então
        Aluno = Aprovado
        Senão
                Aluno = Recuperação
Senão
        Aluno = Reprovado
```



Vejam o código anterior! Nesse caso, o programa realiza um primeiro teste (Média >= 5.0). No caso de a Média ser menor que 5.0, o comando "Aluno = Reprovado" será executado e o fluxo termina. **Caso contrário, ou seja, a Média sendo maior ou igual a 5.0 executamos os comandos do bloco "Então", que, nesse caso possui mais uma estrutura de decisão**.

O segundo teste avalia a expressão "Média >= 7.0". **Chamamos esse tipo de utilização em diferentes níveis de estrutura de decisão (seleção) aninhada.** Antes de aprendermos a nossa última estrutura de decisão (seleção), "Caso Selecione", temos que nos atentar para o fato de que a condição testada nas estruturas de decisão não precisa ter uma expressão somente.

Qualquer uma que retorne um valor verdadeiro ou falso é válida como condição de teste. Expressões como (Média > 10 ou Média < 5) ou ainda (Aluno == Reprovado e Média < 3.0 ^ NumeroFaltas > 5) poderiam ser utilizadas. *Sabe uma utilizada muitas vezes em programação?* (1 = 1). **Isso mesmo, é óbvio, mas é muito utilizado em programação para algumas coisas específicas – é óbvio 1 = 1 é verdadeiro.**

CASO 4:

```
Selecione (Número)
     Caso 13: Presidente = "Dilma"
     Caso 20: Presidente = "Pastor Everaldo"
     Caso 28: Presidente = "Levy Fidelix"
     Caso 43: Presidente = "Eduardo Jorge"
     Caso 45: Presidente = "Aécio Neves"
     Caso 50: Presidente = "Luciana Genro"
     Caso Outro: Presidente = "Nulo"
Fim Selecione
```

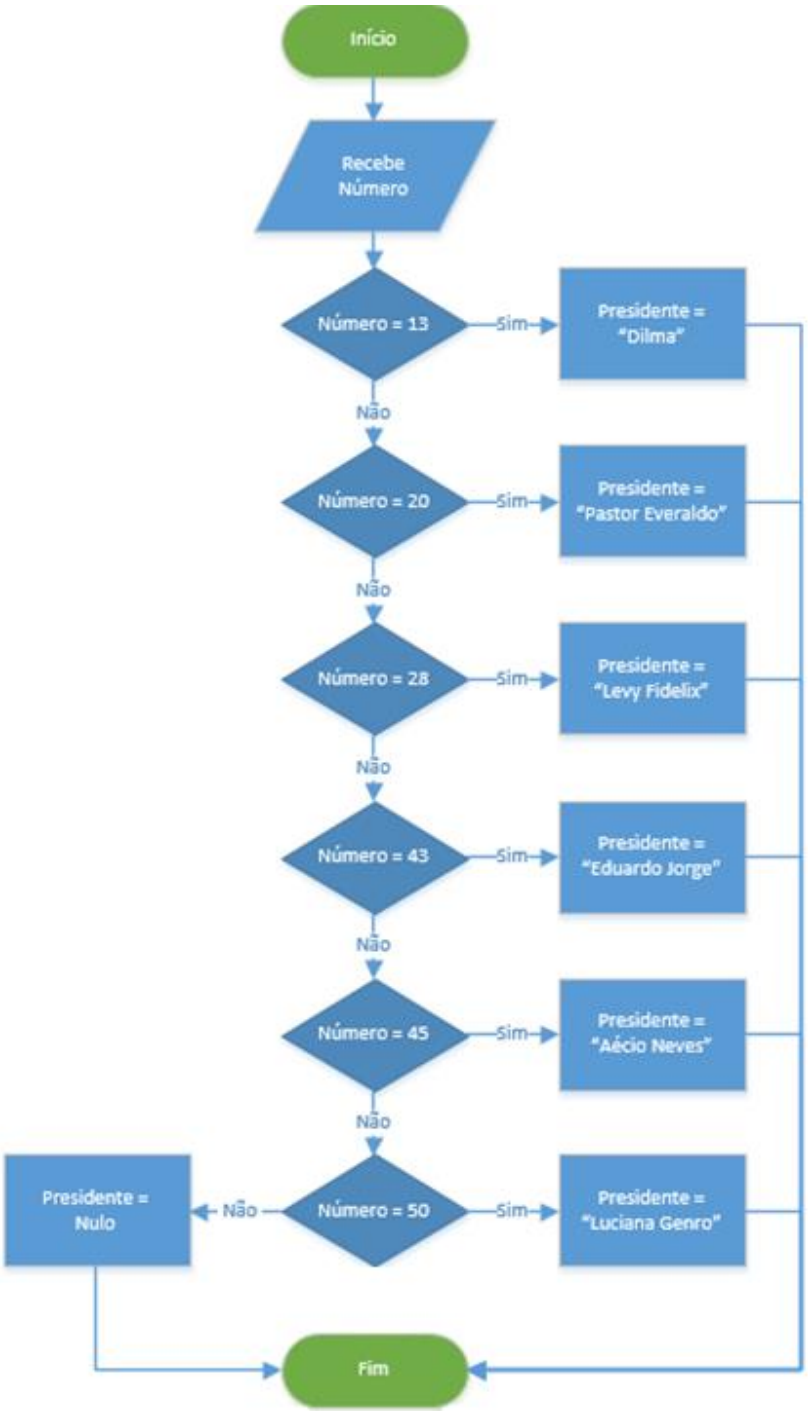
A estrutura "Caso-Selecione" também é conhecida como de decisão múltipla escolha. Diferente das outras que vimos, não parte de um teste de condição para definir para onde o fluxo deve ser desviado, mas sim avalia o valor de uma variável e dependendo de qual seja o conteúdo, encaminha para o rótulo indicado e executa as instruções ali apresentadas.



Temos, então, quatro Estruturas de Decisão (Seleção): Se-Então, Se-Então-Senão, Se-Então (Aninhados) e Caso-Selecione.[3]

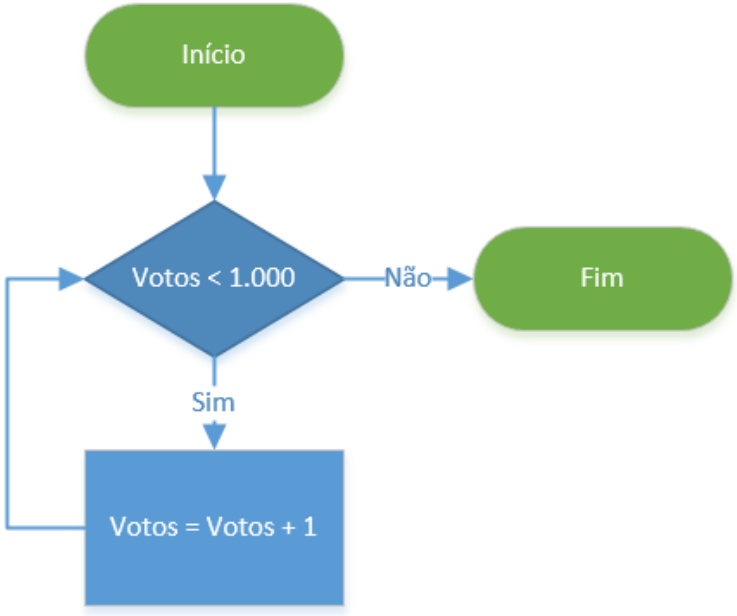
Agora veremos as famosas Estruturas de Repetição! **Essas estruturas são utilizadas quando se deseja que um determinado conjunto de instruções ou comandos sejam executados por mais de uma vez.** A quantidade pode ser definida, indefinida, ou enquanto um estado se mantenha ou seja alcançado. Parece confuso, mas não é! Vamos ver, pessoal...

[3] Em inglês: If-Then, If-Then-Else, If-Then (nested) e Switch-Case.



CASO 1: REPETIÇÃO PRÉ-TESTADA (TESTA-SE ANTES DE PROCESSAR!)

```
Enquanto (Votos < 1.000) Faça
     Votos = Votos + 1
Fim-Enquanto
```



Esse primeiro tipo de estrutura de repetição realiza um teste antes de executar qualquer instrução dentro de seu bloco. Desse modo, as instruções podem nem ser executadas, caso o teste da condição inicial falhe. No exemplo acima, se a variável "Votos" for maior ou igual a 1.000, a instrução "Votos = Votos + 1" não é executada uma vez sequer.

A variável sendo testada deve sofrer alguma alteração dentro do bloco da estrutura de repetição **sob pena de a execução não ter fim**. Exemplo:

```
Votos = 500

Enquanto (Votos < 1.000) Faça
```

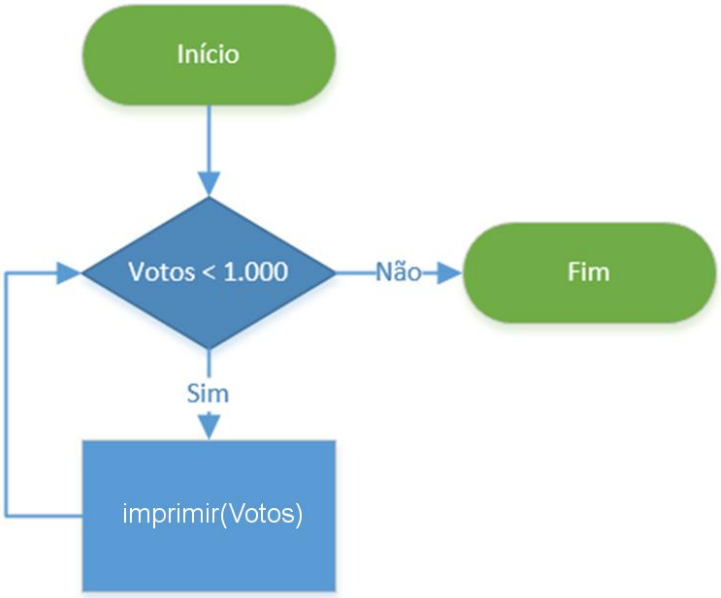
```
    imprimir("Eu nunca mais vou sair daqui! Muahahahaha")
Fim-Enquanto
```

A variável votos tem o valor estipulado em 500, ou seja, é menor que 1.000. Quando é executado o primeiro teste "Votos < 1000" o resultado é verdadeiro, ou seja, o fluxo é direcionado para dentro do bloco e a instrução "imprimir" é executada. Na segunda repetição, como não houve alteração da variável "Voto" vai executar novamente a instrução "imprimir" e assim por diante.

Daí a obrigatoriedade de se alterar o valor da variável utilizada no teste de condição de entrada e saída da estrutura. Em inglês, a estrutura é "While Do".

CASO 2: REPETIÇÃO COM VARIÁVEL DE CONTROLE

```
Para Votos de 1 até 1000 Faça
    imprimir(Votos)
Fim-Enquanto
```



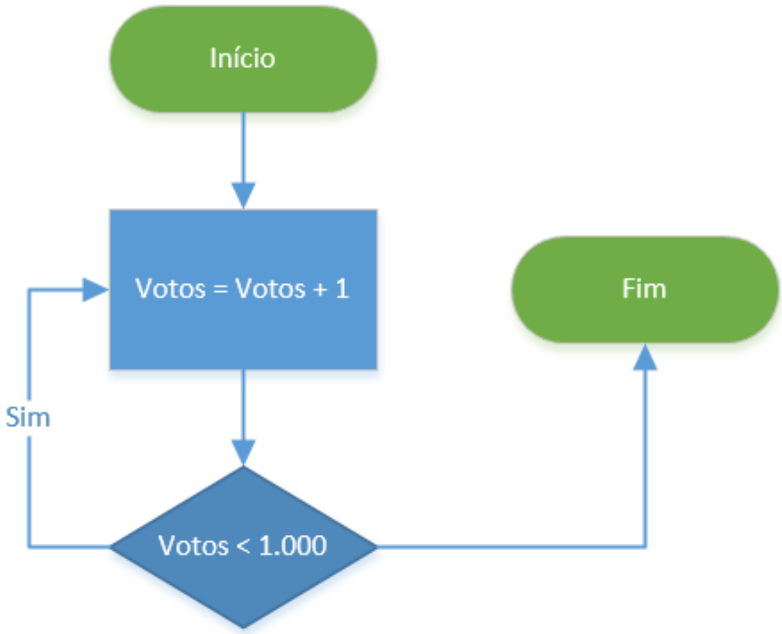
Esse tipo de estrutura é um pouco diferente da primeira, já que define de antemão quantas vezes será repetida as instruções dentro do bloco. No nosso exemplo, a instrução "imprimir (Votos)" será executada 1000 vezes, pois assim determina a expressão "Para Votos de 1 até 1000 Faça". Essa expressão é simplificada, mas realiza várias instruções internas:

- Testa se Votos é igual a 1000;
- Se é igual, encerra a repetição;
- Se é menor, soma 1 à variável de controle "Votos";
- Mais uma repetição

Perceba que, como não há outros testes de condição de saída a não ser a contagem da variável de controle, **as instruções que fazem parte do bloco de execução da estrutura de repetição não precisam alterar a variável de controle,** pois a própria estrutura o faz. É uma grande diferença para os casos de estrutura de repetição pré-testada e pós-testada, que necessitam de tal alteração.

CASO 3: REPETIÇÃO PÓS-TESTADA (TESTA-SE DEPOIS DE PROCESSAR!)

```
Faça
      Votos = Votos + 1
Enquanto (Votos < 1.000)
```



O último caso é o pós-testado. **Como sugere o nome, nessa estrutura a instrução do bloco é executada sempre ao menos uma vez!** Isso porque o primeiro teste somente é feito ao final. Em inglês essa estrutura é conhecida como "Do While". Agora que sabemos tudo sobre estruturas de seleção (ou decisão) e de repetição, que tal tentar um exemplo de código que imprime a letra de "Hey Jude" dos Beatles.

Fomos inspirados pelo fluxograma abaixo, **que destaca muito bem as repetições e decisões para saber que parte da letra cantar**. *Vamos nessa?*

```
01 estrofe = 0
02
03 Enquanto (estrofe < 3) Repetir
04   estrofe == estrofe + 1
05   imprimir("Hey Jude, don't ")
06
07   Se (estrofe == 1) Então
08     imprimir("make it bad")
09     imprimir("take a sad song and make it better")
10   Senão Se (estrofe == 2) Então
11     imprimir("be afraid")
12     imprimir("you were made to go on and get her ")
13   Senão Se (estrofe == 3) Então
14     imprimir("let me down")
15     imprimir("you have found her, now go and get her")
16
17   imprimir("remember to let her ")
```



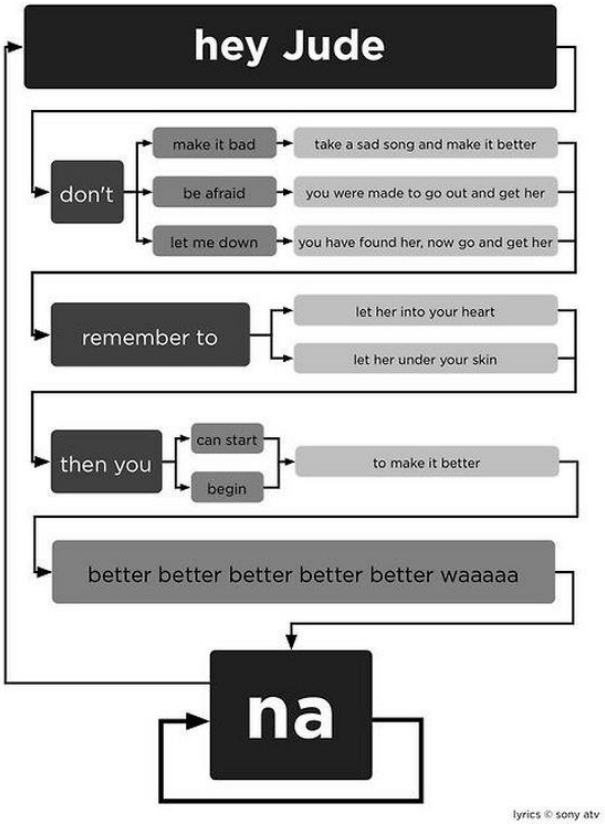

```
18
19   // Teste se estrofe é par ou ímpar
20   Se (estrofe % 2 == 1) Então
21     imprimir("into your heart")
22   Senão
23     imprimir("under your skin")
24
25   imprimir("then you ")
26
27   // Teste se estrofe é par ou ímpar
28   Se (estrofe % 2 == 1) Então
29     imprimir("can start")
30   Senão
31     imprimir("begin")
32
33   imprimir("to make it better")
34
35   Se (estrofe == 3) Então
36     imprimir("better better better better BETTER WAAAAAAAAAAA")
37     Repetir
38        imprimir("NA NA NA NANANA NAAAAAAA")
39        imprimir("NANANA NAAAAAAA")
40        imprimir("Hey Jude")
41     Enquanto(Verdadeiro)
```




A música começa com uma estrutura que conhecemos, a repetição pré-testada (linha 03). **A condição de execução é se estrofe é menor que 3, isso porque somente temos três estrofes na música.** A seguir vemos uma estrutura de decisão composta (linha 07), todas avaliando a variável "estrofe". Para cada um dos valores de estrofe (1, 2 e 3) um bloco em separado é executado.

Após o primeiro "Se Então Senão", temos outro, que testa se estrofe é par ou ímpar (linha 28). Essa estrutura também é composta, pois possui um bloco "Senão". Por fim, temos uma estrutura de seleção (decisão) simples (linha 35), ou seja, somente com bloco "Se Então" e, dentro desse, encontramos uma estrutura de repetição pós-testada (linha 37).

No exemplo lúdico essa repetição nunca termina (*pois ela termina em fade e parece continuar, lembram? :)*), pois a condição "Verdadeiro" sempre está satisfeita. Nos programas reais temos que lançar mão de algum escape. *Todo mundo cantou bem alto para o vizinho ouvir? Ficou difícil?* **Tirei do diagrama abaixo, assim acho mais fácil de visualizar. Abraços e até a próxima.**



RESUMO

Vamos agora trabalhar com duas estruturas muito importantes dentro do desenvolvimento de software. **As estruturas de repetição e decisão** são responsáveis por interferências nas instruções implantando condições durante a execução do código.

Teremos então decisões diferentes a depender do tipo de estrutura que é montada, ou seja, são estabelecidos **desvios ou loops nas instruções do código**. Vamos ver como funcionam essas estruturas de repetição:

```
Se (idade>= 18) então
        Pessoa = Maior de idade
Se não
        Pessoa = Menor de idade
```

Note que temos aqui uma estrutura de decisão ou seleção, onde temos dois fluxos possíveis que é a pessoa ser maior de idade ou ser menor de idade, com a condição da idade >= 18, logo uma das condições será executada, porque temos se ou se não.

No primeiro Se (**IF**) temos um teste de repetição, onde a instrução avalia a condição >= 18 para então validar a idade (Maior, Menor).

Temos também a condição de retorno onde temos múltiplas opções. Exemplo:

```
Selecione (número):
        Caso 1: ''primeiro colocado''
        Caso 2: ''segundo colocado''
        Caso 3: ''terceiro colocado''
```




Aqui não temos um teste de condição para a definição do fluxo, temos na verdade uma **variável** que a depender do **conteúdo**, atribui uma colocação (primeiro, segundo ou terceiro colocado). Esse tipo de estrutura é conhecida como: **Switch-Case**.

Estruturas de repetição por sua vez, são utilizadas quando se deseja que instruções determinadas sejam executadas diversas vezes de maneira **indefinida ou de maneira definida**. Exemplo:

```
Enquanto (pontuacao< 100):

    Pontuacao = pontuacao + 1
```

Note que aqui temos um teste antes do início da instrução, a fim de que a condição inicial não falhe, se falhar a instrução não é executada.

No caso, se a variável "pontuacao" for maior ou igual a 100, a instrução entra em somatório "pontuacao = pontuação+1, até que "pontuação" se torne maior que 100, acontecendo isso a instrução encerra sua execução.

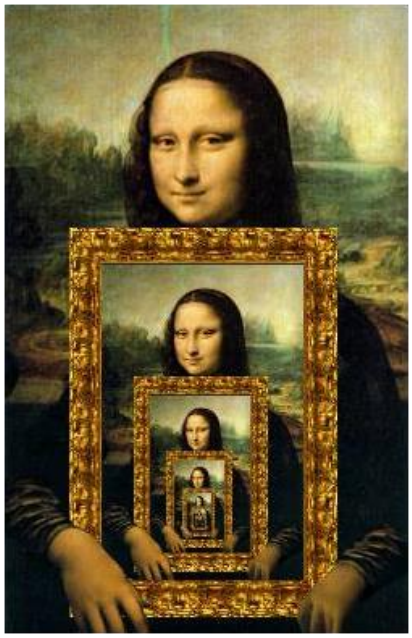
Esse tipo de estrutura é conhecida como "**while**".

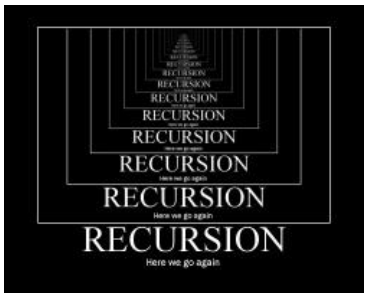
4 – Recursividade

Por fim, vamos falar sobre recursividade! *O que é isso, professor?* Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, **a recursão é uma técnica em que uma função chama a si mesma**. Eu gosto de pensar em recursividade por meio de imagens como as apresentadas abaixo.





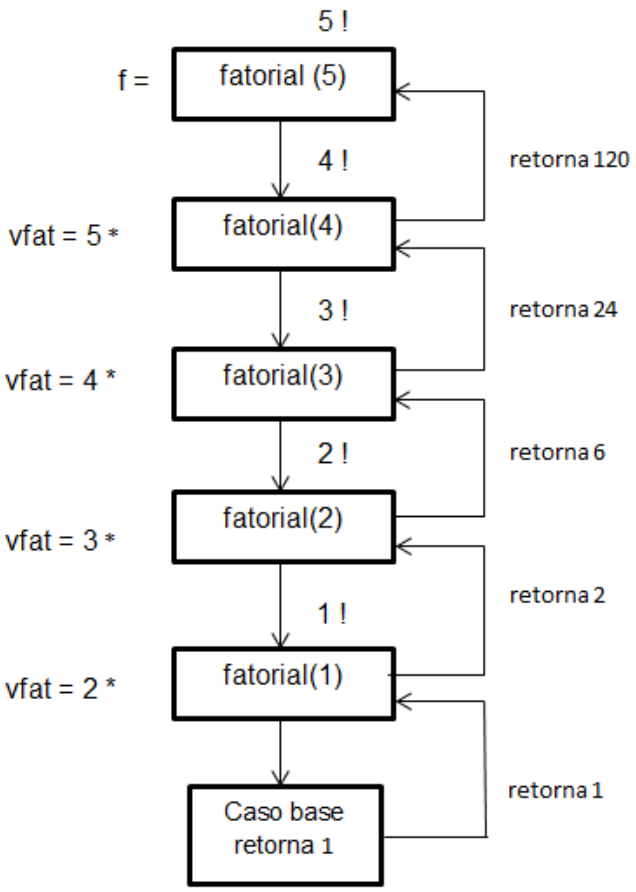




Nós dissemos que se trata de uma função que chama a si mesma e, daí, temos o nosso primeiro problema. *Se ela chama a si mesma sempre, não entra em um loop infinito?* **Sim, por isso é necessário um caso-base (solução trivial), i.e., um caso mais simples em que é utilizada uma condição de parada, que resolve o problema**. Toda recursividade é composta por um caso base e pelas chamadas recursivas.

Vantagens da recursividade: torna a escrita do código mais simples, elegante e, muitas vezes, menor. Desvantagens da recursividade: quando o loop recursivo é muito grande, é consumida muita memória nas chamadas a diversos níveis de recursão, **tendo em vista que cada chamada recursiva aloca memória para os parâmetros, variáveis locais e de controle**.

Em muitos casos, uma solução iterativa gasta menos memória e torna-se mais eficiente em termos de performance do que usar recursão. *Galera, vocês se lembram que estudaram fatorial na escola?* Pois é, fatorial é uma operação matemática em que um número natural (representado por n!) é o produto de todos os inteiros positivos menores ou iguais a n.



Professor, facilita aí. Já saí da escola há muito tempo. Ok! Fatorial(5) = 5x4x3x2x1; uma maneira diferente de escrever isso é: Fatorial(5) = 5 x Fatorial(4), que é o mesmo que dizer 5 x (4 x Fatorial(3)), que é o mesmo que 5 x (4 x (3 x Fatorial(2))), que é o mesmo que 5 x (4 x (3 x (2 x Fatorial(1))))). *E agora, professor?* Agora é a vez do caso-base! No caso do fatorial, o caso-base é: Fatorial(1) = 1. **Todas essas instâncias ficaram em memória aguardando a chegada do caso-base e agora retornamos com os resultados**. Dado o caso-base, temos que: 5 x (4 x (3 x (2 x Fatorial(1)))), logo 5 x (4 x (3 x (2 x 1))) = 120. Vejam ao lado a imagem da a execução de um código representando o fatorial.

Por fim, gostaria de mencionar que existem dois tipos de recursividade: direta e indireta. De modo bem simples, a recursividade direta é aquela tradicional em que uma função chama a si mesma (Ex: Função A chama a Função A); a recursividade indireta é aquela alternativa em que uma função chama outra função que chama a primeira (Ex: Função A chama a Função B, que chama a Função A).

RESUMO

Recursividade é uma técnica em que uma **função chama a si mesma**. A vantagem é que o código quando recursivo é mais **clean** (limpo), mais simples e menor, todavia quando executamos um loop através de recursividade, a tendência é que tenhamos um **consumo maior de memória**, porque várias chamadas são executadas, ou seja, eu acabo alocando **muita memória para uma variável** por exemplo.

Existem dois tipos de recursividade: **direta** e **indireta**. A direta é a recursividade na qual uma **função chama a si mesma** e a indireta é quanto uma **função chama outra função que chama a primeira**.

Recursividade direta: Função X chama Função X.

NÃO ATIVARDesenvolvimento p/ Prefeitura de Lucas do Rio Verde-MT (Analista em TI) Com Videoaulas - Pós-Edital
www.estrategiaconcursos.com.br

Recursividade indireta: Função X chama Função Y, que chama Função X.

Na nossa aula de estrutura de dados aprofundaremos mais no conceito de recursividade, exemplo dela é a função quicksort que é uma função que chama a si mesma. Note que através da recursividade eu posso resolver problemas por meio da divisão dos problemas, em problemas menores similares.

Logo, o mesmo método utilizado para reduzir os problemas menores, pode ser utilizado para resolver os problemas maiores e vice-versa, até chegarmos a solução do problema.

Por exemplo: Vamos estabelecer um processo recursivo para calcular o fatorial de um inteiro n.

Temos então que:

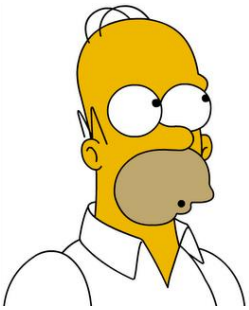
n! = 1, quando n = 0

n! = n x (n-1)!, quando n > 0

Temos então dois casos aqui, no primeiro n=0, então n!=1 e no segundo n! = n x (n-1)!. Usando recursividade podemos montar o seguinte algoritmo, para calcular, por exemplo: 4!

```
Int fatorial (int n) {
        If (n ==0)
                Return1;
        else
                return n*fatorial(n-1);
}
```

Para implementar essa função utilizamos o mecanismo chamado Pilha, mas isso é assunto para nossa aula de estrutura de dados. Por enquanto vamos executar o valor 4!



Professor espere 1 minuto, eu não me lembro o que é fatorial?

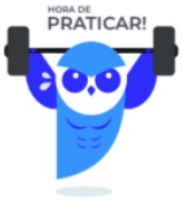
Fatorial é uma operação matemática em que um número natural (n!) é o produto de todos os inteiros positivos menores ou iguais a n. Vamos exemplificar para clarear a nossa visão sobre fatorial.

Fatorial (4) = 4x3x2x1. Ok? Concorda comigo que isso é o mesmo que 4x3! ou que 4x3x2!? Perceba então que chegaremos aqui, a um caso base que é: Fatorial (1) = 1.

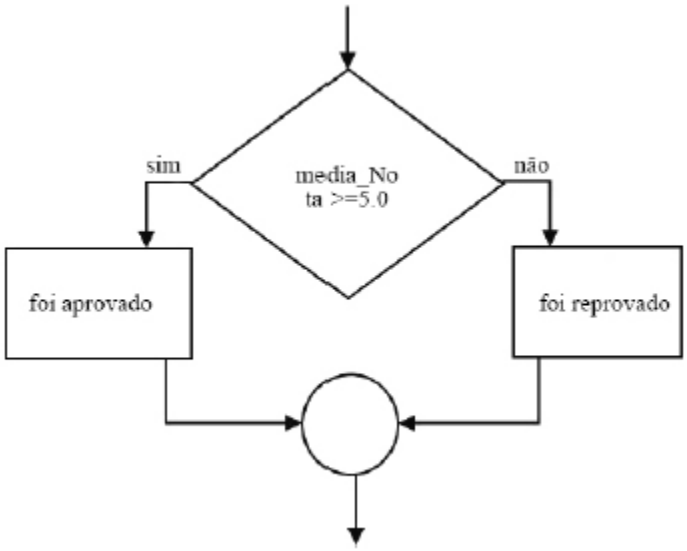
Quando executamos uma instrução em memória, as instâncias recursivas aguardam até a chegada do caso base: Fatorial (1) = 1. Teríamos então:

```
4! = 4*3!
     =4* (3*2!)
     =4* (3*(2*1!))
     =4* (3*(2*(1*0!)))
     =4*(3*(2*(1*1)))
     =4*(3*(2*1))
     =4*(3*2)
     =4*6
     =24
```

QUESTÕES COMENTADAS



1. (CESPE / Analista Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)



A estrutura lógica presente no diagrama apresentado é do tipo

a) SE ENTÃO.

b) CASO SELECIONE.

c) CASO REPITA.

d) SE ENTÃO SENÃO.

Gabarito: **Letra D**.

Analisando as alternativas:

a) SE ENTÃO.

ERRADO. A estrutura SE ENTÃO tem um bloco de comandos, e a outra saída do teste (losango) vai direto para confluência (o círculo)

b) CASO SELECIONE.

ERRADO. A estrutura CASO SELECIONE tem um teste (losango) para cada caso, e cada caso com somente um bloco de comandos. Na figura, há um teste com dois blocos de comando.

c) CASO REPITA.

ERRADO. A estrutura CASO REPITA tem um teste (losango) com somente um bloco de comandos, e a saída do bloco de comandos deve retornar para o teste.

d) SE ENTÃO SENÃO.

CERTO. A estrutura SE ENTÃO SENÃO tem dois blocos de comando, um para o caso verdadeiro, outro para o caso falso, e ambos têm as saídas direcionadas para a confluência (círculo).

Gabarito: **Letra D**.

2. (FGV / Analista Legislativo Municipal (CM Salvador) / 2018 / / Tecnologia da Informação)

Expressões lógicas são frequentemente utilizadas em linguagens de programação. Por exemplo, um comando if com a expressão

if not (A and B)

pode ser reescrito, para quaisquer valores lógicos de A e B, com a expressão:

a) A or B

b) not A or not B

c) not A or B

d) not (not A or not B)

e) A and B

Gabarito: **Letra B**

Essa questão é basicamente a aplicação do *Teorema de De Morgan*, onde você tem uma expressão equivalente negando o resultado, negando os operadores e trocando as operações.

Assim, not (A and B) é o mesmo que not not (not A or not B) que é o mesmo que (not A or not B).

Gabarito: **Letra B**

3. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 6)

As estruturas de controle de fluxo WHILE e DO...WHILE possuem a mesma finalidade e seus respectivos blocos de comandos são executados pelo menos uma vez em cada uma delas.

Gabarito: **ERRADO**

Ambas estruturas possuem a mesma finalidade - iterar a partir de uma condição. Contudo, o teste do WHILE é feito antes da execução do bloco e, caso seja falso, o bloco não será executado nem a 1a. vez. O DO...WHILE, por sua vez, executa o bloco antes de realizar o teste, garantindo que o bloco será executado pelo menos uma vez.

Corrigindo o item:

As estruturas de controle de fluxo WHILE e DO...WHILE possuem a mesma finalidade, **sendo que o bloco de comandos do DO...WHILE é executado pelo menos uma vez**.

Gabarito: **ERRADO**

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

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

Gabarito: **CERTO**.

Na passagem **por referência**, a variável interna e externa apontam para o mesmo endereço. Portanto, alterações no valor da variável interna serão refletidas na variável externa.

Na passagem **por valor**, uma cópia do valor da variável externa é feita para outra posição de memória, que será referenciada pela variável interna. Portanto, no caso de passagem por valor, se a variável interna é alterada, essa alteração não é refletida na variável externa.

Gabarito: **CERTO**.

5. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

Uma variável com capacidade de armazenar um baite pode representar valores no intervalo de !512 a 512.

Gabarito: **ERRADO**

Um byte tem 8 bits, ou seja, 2^8 = 256 possibilidades. No caso de ter um bit de sinal, terá valores entre -128 até +127.

6. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

Para a determinação da parte decimal de um número real, pode-se utilizar a função INT(x), como no exemplo a seguir, onde INT(x) retorna a parte inteira de x.

x = 3.1415926;

escreva x-INT(x)

Gabarito: **CERTO**

Se **INT(X)** retorna a parte decimal do número, então **INT(3.1415926)** vai retornar **3**. Logo, 3.1415926 - 3 = 0.1415926, que é a parte decimal do número. O método funciona para qualquer número.

7. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

Para o seu correto funcionamento, os algoritmos devem ser implementados como um conjunto de métodos e mensagens.

Gabarito: **ERRADO**

Algoritmos podem ser implementados de qualquer forma que permita instruções e controle de fluxo. Não existe necessidade de utilização de métodos e mensagens - característicos da Orientação a Objetos.

8. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

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.

Gabarito: **ERRADO**.

Existem diversas formas de criação de blocos de comando, dependendo da linguagem de programação. Contudo, **nenhuma delas utiliza INPUT e OUTPUT.**

Os modos de criação de blocos mais comuns são:

- Utilização de chaves {} - Utilizada em Java, C, JavaScript.
- Utilização das palavras-chave BEGIN e END - Utilizada em Pascal, ALGOL, Ruby.
- Utilização das palavras-chave DO e DONE - Utilizada em Bash, Visual Basic.
- Utilização de indentação - Utilizada por Python.

9. (CESPE / Oficial Técnico de Inteligência / 2018 / / 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.

O laço de repetição na estrutura de repetição para será executado pelo menos uma vez.

Gabarito da Banca: **CERTO**.

Questão polêmica e, para mim, deveria ser **errada** ou **anulada**. A banca não acatou os recursos contra essa questão.

O único laço que garante pelo menos uma execução do bloco não é o **PARA**, mas o **FAÇA ... ENQUANTO**.

O laço para **PARA** tem um comportamento diferente em cada linguagem mas, normalmente, ele define uma *inicialização*, um *incremento* e um *teste*. É assim, por exemplo, em C, C++, Java, JavaScript, e outras. Nessas linguagens, caso a *inicialização* faça com que o *teste* retorne "falso", então nem a primeira iteração será executada.

10. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

A estrutura de controle seleção não pode ser utilizada nas situações em que duas alternativas dependam de uma mesma condição — uma de a condição ser verdadeira e outra de a condição ser falsa.

Gabarito: **ERRADO**

Nada impede a criação que a lógica:

```
SE (meuTeste) ENTAO
  // codigo no caso true
SENAO
  // codigo no caso false
FIM-SE
```

Seja transformada em:

```
SELECIONE (meuTeste)
  CASO true:
    //codigo se true
  CASO false:
    //codigo se false
FIM-SELECIONE
```




Não é nem necessário usar o CASO false, podemos também usar a opção padrão:

```
SELECIONE (meuTeste)
 CASO true:
  //codigo se true
 DEFAULT:
  //codigo se false
FIM-SELECIONE
```

Não é uma opção muito elegante, mas não está errado do ponto de vista de algoritmo.

11. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

Uma das vantagens de se construir um algoritmo por meio do pseudocódigo é o fato de que a passagem do algoritmo para uma linguagem de programação qualquer se torna uma atividade quase que instantânea.

Gabarito: **CERTO**.

A transformação do pseudocódigo para uma linguagem de programação é uma atividade de **pura tradução**. No pseudo-código o algoritmo de resolução do problema já está pronto, portanto, falta apenas a conversão dos comandos textuais abstratos nos comandos da linguagem de programação.

12. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

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.

Gabarito: **CERTO**.

Uma **variável** de um programa é, basicamente, um **espaço de armazenamento na memória** do computador. Dessa forma, pode-se armazenar nesse espaço diversos valores diferentes, mas como a área é a mesma, a atribuição de um novo valor vai sobrescrever o valor antigo. Portanto, a **variável só armazena um valor por vez**.

13. (FUNRIO / Analista Legislativo (CM SJM) / 2018 / / Analista em Tecnologia)

A programação de computadores necessita das estruturas de controle abaixo referenciadas para que possa ser utilizada com eficiência. Neste contexto, relacione as estruturas de controle a seguir com as características correspondentes.

(EE) ENQUANTO ... FAÇA ... FIM ENQUANTO

(RR) REPITA ... ATÉ ... FIM REPITA

( ) A condição de teste da estrutura é inserida no fim da estrutura de controle.

( ) A condição de teste da estrutura é inserida no início da estrutura de controle.

( ) Se o resultado do teste for FALSO, a execução do programa permanece no loop.

( ) Se o resultado do teste for VERDADEIRO, a execução do programa permanece no loop.

( ) A saída do loop ocorre quando o teste da condição de controle retorna valor FALSO.

( ) A saída do loop ocorre quando o teste da condição de controle retorna valor VERDADEIRO.

A relação correta, de cima para baixo, é:

a) (RR), (EE), (EE), (RR), (RR) e (EE).

b) (EE), (EE), (RR), (RR), (EE) e (RR).

c) (RR), (RR), (EE), (RR), (EE) e (EE).

d) (RR), (EE), (RR), (EE), (EE) e (RR).

e) (EE), (RR), (EE), (EE), (RR) e (RR).

Gabarito: **Letra D**.

O **(EE) ENQUANTO ... FAÇA ... FIM ENQUANTO** tem a seguinte estrutura:

ENQUANTO (teste verdadeiro) FAÇA

// bloco de comandos

FIM ENQUANTO

- O *bloco de comandos* é executado depois do teste, enquanto o teste for verdadeiro.
- O teste é feito antes da execução do bloco.
- A condição for falsa já no início, o bloco nunca será executado.
- No momento em que o teste se torna falso, a repetição encerra (saída do loop).

O **(RR) REPITA ... ATÉ ... FIM REPITA** tem a seguinte estrutura:

REPITA

// bloco de comandos

ATÉ (teste falso) FIM REPITA

- O bloco de comandos é executado antes do teste, enquanto o teste for falso.
- O teste é feito somente no final da iteração.
- O bloco de comandos será executado pelo menos uma vez.
- No momento que a condição de saída é alcançada, o teste é verdadeiro, e o loop encerra.

Portanto, a resposta dos itens é:

(**RR**) A condição de teste da estrutura é inserida no fim da estrutura de controle.

(**EE**) A condição de teste da estrutura é inserida no início da estrutura de controle.

(**RR**) Se o resultado do teste for FALSO, a execução do programa permanece no loop.

(**EE**) Se o resultado do teste for VERDADEIRO, a execução do programa permanece no loop.

(**EE**) A saída do loop ocorre quando o teste da condição de controle retorna valor FALSO.

(**RR**) A saída do loop ocorre quando o teste da condição de controle retorna valor VERDADEIRO

Portanto, resposta **d) (RR), (EE), (RR), (EE), (EE) e (RR)**.

14. (FCC / Estagiário (SABESP) / 2018 / / Ensino Médio Regular)

Considere, por hipótese, que a SABESP utiliza diferentes preços de tarifas para os serviços de abastecimento de água e/ou coleta de esgoto para o município de São Paulo. Para a categoria Residencial/Favela as tarifas são:

Consumo	Valor da Tarifa
0 a 10	6,25/mês
11 a 20	0,71/m³
21 a 30	2,36/m³
31 a 50	7,14/m³
acima de 50	7,89/m³

Foi solicitado a um estagiário propor a lógica de programação para a solução do seguinte problema: ler o valor do consumo de um usuário da categoria Residencial/Favela (variável consumo) e calcular o valor a pagar com base nas tarifas (variável valor).

O Estagiário sugeriu utilizar

a) o tipo básico inteiro para ambas as variáveis.

b) o tipo básico caracter para consumo e o tipo básico inteiro para valor.

c) a instrução leia para ler o valor que o usuário deverá pagar.

d) as instruções se então senão de forma aninhada para avaliar as diferentes faixas de consumo.

e) a instrução escolha caso para avaliar os diferentes tipos de valor que poderão ser pagos pelo usuário.

Gabarito: **Letra D**.

Questão bem interessante, que requer entendimento teórico e elaboração. Analisando item a item:

a) o tipo básico inteiro para ambas as variáveis.

ERRADO. Ele não especifica no enunciado se o consumo é um número inteiro, então poderíamos ter dúvidas com relação ao tipo da variável. Mas o valor da tarifa é um número decimal e, portanto, o valor final da conta também deve ser.

b) o tipo básico caracter para consumo e o tipo básico inteiro para valor.

ERRADO. Consumo é a quantidade de energia consumida por mês. É um número, e não um caractere.

c) a instrução leia para ler o valor que o usuário deverá pagar.

ERRADO. O valor a ser pago será calculado pela multiplicação do consumo pelo valor da tarifa.

d) as instruções se então senão de forma aninhada para avaliar as diferentes faixas de consumo.

CERTO. Instruções SE-ENTÃO-SENÃO aninhadas são ótimas alternativas para o tratamento de intervalos, sendo ideais para o tratamento das faixas de tarifa a partir do consumo da questão.

e) a instrução escolha caso para avaliar os diferentes tipos de valor que poderão ser pagos pelo usuário.

ERRADO. A instrução escolha caso não analisa intervalos. Ele direciona o fluxo para o rótulo cujo valor seja idêntico ao valor da variável selecionada.

15. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

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.

Gabarito: **CERTO** (mas podia ser mais cuidadoso...)

Uma constante é um endereço fixo em memória (portanto, gera um espaço físico), e esse endereço não pode ter seu conteúdo alterado. Todos as linguagens de programação permitem criar um nome para esse endereço - o nome da constante. Perceba que, o que não sofre alteração durante a execução do programa é o conteúdo da constante, e não necessariamente o seu nome (a maior parte das linguagens permite criar outras referências para o mesmo endereço de memória com outro nome).

Na minha opinião, faltou zelo do examinador. Mas, sem procurar "cifre em cabeça de cavalo", o item está **CERTO**.

16. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

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";
```

Gabarito: **CERTO**.

Pela prioridade dos operadores, devemos resolver primeiramente:

1. Parêntesis
2. Exponenciação (^)
3. Multiplicação (*), Divisão (/) e Resto (%)
4. Soma (+) e Subtração (-)

Portanto, analisando os 2 operandos do teste ((-(-2-6*12/3-1)) > (3+3-3*3-3^3+3)):

Operando 1:

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

(-(-2-**24**-1))

(-(-27))

(27)

Operando 2:

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

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

(3+3-**9**-**27**+3)

(-27)

Portanto, o teste é 27 > -27, que implica que será escrito o texto "verdadeiro". Portanto, afirmação **CORRETA**.

17. (FGV / Analista em Tecnologia da Informação e Comunicação (SEPOG RO) / 2017)

Considere o algoritmo em pseudocódigo descrito a seguir.

```
para i=0 até n
inicio
  j = 1
  enquanto j<n
  inicio
    j = 2 * j
    para k = 0 até j
    inicio
      execute f
    fim
  fim
fim
```

Assinale a opção que indica o número de vezes em que o código irá executar a função f para n igual a 8.

a) 25
b) 153
c) 278
d) 481
e) 587

Gabarito: **Letra B**.

Considerando n = 8, o loop externo (para i = 0 até n) será executado de 0 até 8 (9 vezes). Perceba que a variável "i" não é utilizada no algoritmo, ou seja, não existe nenhuma modificação referente o número da iteração.

No loop (enquanto j<n), a variável j começa em 1, e é dobrada a cada iteração. O loop será executado para j=1, j = 2, j = 4. (internamente ao loop passa a ser j=2, j=4 e j=8, porque "j" muda na primeira linha).

Como o loop mais interno (para k = 0 até j), a função "f" será executada j+1 vezes (de 0 até j). Portanto, para j=2, 4 e 8, a função f será executada 3, 5 e 9 vezes, respectivamente. São 17 vezes por iteração.

São 17 execuções de f por cada loop de (enquanto j<n), sendo que esse loop será executado 9 vezes. 17 * 9 = 153, portanto, Letra B.

18. (FCC / Técnico (DPE RS) / 2017 / / Informática)

Considere o seguinte algoritmo em pseudocódigo:

```
Algoritmo Valida
tipo V = vetor [1..4] de inteiro
var vet: V
indice, numero: inteiro

Inicio
indice←1
enquanto (indice<=4)faça
  leia(numero);
enquanto(..... I ) faça
    imprima("Valor invalido. Digite um valor dentro do limite.")
    leia(numero)
  fim_enquanto
  vet[indice]←numero
```

indice←indice + 1

fim_enquanto

para (indice de 1 até 4 passo 1) faça

..... II

fim_para

Fim

Para que o algoritmo acima leia quatro valores de anos de 1900 até 2017 e os apresente na tela, a lacuna

a) I deve ser preenchida com numero>=1900 e numero<=2017

b) II deve ser preenchida com leia(vet[indice])

c) I deve ser preenchida com numero<1900 ou numero>2017

d) II deve ser preenchida com imprima ("Valor valido = ", vetor[indice])

e) I deve ser preenchida com numero>=1900 ou numero<=2017

Gabarito: **Letra C**.

a) I deve ser preenchida com numero>=1900 e numero<=2017

ERRADO. O teste em I deve verificar se numero é inválido, e o teste proposto verifica se ele é válido.

b) II deve ser preenchida com leia(vet[indice])

ERRADO. O comando em II deve imprimir o valor, e não ler.

c) I deve ser preenchida com numero<1900 ou numero>2017

CERTO. O teste em I deve verificar se numero é inválido. Números menores que 1900 ou maiores que 2017 são inválidos.

d) II deve ser preenchida com imprima ("Valor valido = ", vetor[indice])

ERRADO. O nome da variável é *vet*, e não *vetor*.

e) I deve ser preenchida com numero>=1900 ou numero<=2017

ERRADO. O teste proposto sempre será verdadeiro.

19. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Texto 10A1AAA

O algoritmo a seguir realiza o cálculo do dígito verificador (DV) do código de barras (COD) a ser usado em um sistema de controle de processos.

```
1 inicio
2   tipo VET = vetor [1..11] de inteiros;
3   VET: VCOD;
4   inteiro: I, J, K, COD, SOMA1, SOMA2,
        FATOR, DV;
5   SOMA1, SOMA2, FATOR, DV ← 0;
6   leia COD;
7   para I de 1 até 11 passo 1 faça
8     J ← 11 – I;
9     K ← pot (10,J)
10     VCOD [I] ← COD div K;
11     COD ← COD mod K;
12  fim para;
13  para I de 1 até 11 passo 2 faça
14    SOMA1 ← SOMA1 + VCOD[I];
15  fim para;
16  SOMA1 ← SOMA1 * 3;
17  para I de 2 até 10 passo 2 faça
```

```
18    SOMA2 ← SOMA2 + VCOD[I];
19  fim para;
20  SOMA2 ← SOMA2 + SOMA1;
21  FATOR ← SOMA2 div 10;
22  FATOR ← FATOR + 1;
23  FATOR ← FATOR * 10;
24  DV ← FATOR – SOMA2;
25  escreva DV;
26 fim.
```

O algoritmo apresentado no texto **10A1AAA** realiza, entre as linhas 7 e 12, o
a) somatório dos valores dos dígitos parciais, a partir dos valores de cada uma das posições de um vetor.
b) preenchimento de um vetor com os valores de cada umdos dígitos do código de barras lido.
c) preenchimento de um vetor com os somatórios das parcelas das posições ímpares do código de barras lido.
d) cálculo do dígito verificador a partir das parcelas somatórias extraídas do código de barras lido.

Gabarito: **Letra B.**

Para entender o loop entre as linhas 7 e 12, vamos antes pegar um exemplo simples: imagine que eu tenha o número 987, e queira separar em 2 variáveis - a variável X com o valor do dígito mais significativo (9), e Y com o restante (87).

Como posso fazer isso?

Simples! X vai ser a parte inteira da divisão de 987 por 100 (987/100 = 9) e Y vai ser o resto da divisão 987 por 100 (987%100 = 87).

É exatamente isso que o loop de 7 e 12 faz. Vai tirando dígito por dígito de COD e armazenando no vetor VCOD. Vejamos 2 iterações como exemplo:

Para i = 1
j = 10
k = 10^10



vcod[i] = cod / 10^10 (o resultado é o dígito 11)
cod = cod % 10^10 (o resultado é COD sem o dígito 11)
(termina com o dígito 11 em vcod[1],
e cod perde esse dígito, e passa a ter 10)

Para i = 2
j = 9
k = 10^9
vcod[i] = cod / 10^9 (o resultado é o dígito 10)
cod = cod % 10^9 (o resultado é COD sem o dígito 10)
(termina com o dígito 10 em vcod[2],
e cod perde esse dígito, e passa a ter 9)

Portanto, o que está acontecendo nesse trecho de código é **Letra b)** preenchimento de um vetor com os valores de cada um dos dígitos do código de barras lido.

20. (... mesma prova da anterior, mesmo código da anterior..)

Considere que, na execução do algoritmo apresentado no texto **10A1AAA,** o valor do código lido na linha 6 seja 12345678901. Nesse caso, o valor da variável SOMA1, imediatamente após a linha 16, será igual a
a) 138.
b) 60.
c) 78.
d) 46.

Gabarito: **Letra C**.

Se você resolveu a prova, a questão anterior mostrava que o trecho de código entre as linhas 7-12 toma o valor lido na linha 6 e coloca dígito a dígito no vetor VCOD. Logo, para a entrada 12345678901, VCOD[1] = 1, VCOD[2] = 2, ... VCOD[11] = 1.

O loop nas linhas 13-15 vai de 1 até 11 **de 2 em 2 (perceba o *passo 2*)**, agregando o valor daquela posição m SOMA1. Portanto, SOMA1 nesse trecho vai acumular VCOD[1] + VCOD[3] + VCOD[5] + VCOD[7] + VCOD[9] + VCOD[11] = 1 + 3 + 5 + 7 + 9 + 1 = 26.

A linha 16 triplica esse valor. Portanto, 26*3 = 78, **Letra C**.

21. (... mesma prova da anterior, mesmo código da anterior..)

Na execução do algoritmo apresentado no texto 10A1AAA, se o valor de K for 0, então, na linha 10, VCOD[I]
a) receberá o valor de COD.
b) receberá o valor 0.
c) não será atualizado, porque um erro causará a interrupção do algoritmo.
d) receberá o valor 1.

Gabarito: **Letra C**.

Analisando as linhas 9 e 10:

9 K ← pot (10,J)
10 VCOD [I] ← COD div K;

Se K for 0, na linha 10 acontecerá uma divisão por 0 e, como sabemos desde o 10. grau, não podemos dividir por 0. Portanto, teremos um erro na execução do algoritmo, **Letra C**.

22. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

10 A ← 5;

11 B ← A * -2;

12 C ← A – 1;

13 D ← A – 2;

14 H ← (((4*A)div D)-B) – pot(A,2)mod C;

Considerando a execução do trecho de algoritmo precedente, assinale a opção que apresenta o valor atribuído a H na linha 14.

a) 16
b) –9
c) 15
d) –5

Gabarito: **Letra C**.

Analisando linha a linha:

10 A ← 5;

11 B ← A * -2; **# B = -10**

12 C ← A – 1; **# C = 4**

13 D ← A – 2; **# D = 3**

14 H ← (((4*A)div D)-B) – pot(A,2)mod C;

Analisando a linha 14:

(((4*5)div 3)-(-10)) – pot(5,2)mod 4;

(((20)div 3)+10)) - 25 mod 4; # 20 / 3 = 6 (Divisão de inteiros, desconsidera a parte decimal).

(6 + 10) - 1 = 15 # Letra C.

23. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

21 VET[1] ← 1;

22 VET[2] ← 1;

23 para I de 3 até 27 passo 1 faça
24 VET[I] ← VET[I–1] + VET [I–2];

25 fim para;

Considerando a execução completa do trecho de algoritmo precedente, assinale a opção que apresenta o valor armazenado em VET[12].

a) 233
b) 192
c) 144
d) 89

Gabarito: **Letra C**.

Cada iteração vai colocar na posição I do vetor a soma das 2 posições anteriores i-1 e i-2. Portanto:

vet[1] = 1

vet[2] = 1

vet[3] = 1 + 1 = 2

vet[4] = 1 + 2 = 3

vet[5] = 2 + 3 = 5

vet[6] = 8

vet[7] = 13

vet[8] = 21

vet[9] = 34

vet[10] = 55

vet[11] = 89

vet[12] = **144** - **Letra C**.

24. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Em determinada organização, existem cinco salas de reunião e, para cada uma delas, são destinados horários fixos para o agendamento de até seis reuniões por dia. Um especialista em tecnologia da informação criou uma matriz para armazenar os agendamentos solicitados. O sistema percorre a estrutura de dados em busca de uma sala com horário livre para hospedar uma reunião.

Considerando a situação apresentada, assinale a opção cujo trecho de algoritmo apresentado realiza a referida busca.

a)
```
1 leia (SOLICITANTE);
2   para I de 1 até 5 passo 1 faça
3     para J de 1 até 6 passo 1 faça
4       AGESALA[I,J] ← SOLICITANTE
5       escreva "Agendamento com sucesso";
6       escreva "Sala:", I, "Horário:", J;
7     fim para;
8   fim para;
9   se I = 5 então
10    escreva "Agendamento não realizado"
11    escreva "Não há salas disponíveis"
12  fim se;
```

b)
```
1 leia (SOLICITANTE);
2   para I de 1 até 5 passo 1 faça
3     para J de 1 até 6 passo 1 faça
4       se AGESALA[I,J] = " "
5         AGESALA[I,J] ← SOLICITANTE
6         escreva "Agendamento com sucesso";
7         escreva "Sala:", I, "Horário:", J;
8         J ← 100;
9         I ← 100;
10      fim se;
11    fim para;
12  fim para;
13 se I <> 100 então
14   escreva "Agendamento não realizado"
15   escreva "Não há salas disponíveis"
16 fim se;
```

c)
```
1 leia (SOLICITANTE);
2   se SOLICITANTE <> " " então
3     I ← SOLICITANTE;
4     para J de 1 até 6 passo 1 faça
```

```
5         se AGESALA[I,J] = " "
6            AGESALA[I,J] ← SOLICITANTE
7            escreva "Agendamento com sucesso";
8            escreva "Sala:", I, "Horário:", J;
9         fim se;
10      fim para;
11   fim se;
12   se AGESALA[I,J] = 30 então
13      escreva "Agendamento não realizado"
14       escreva "Não há salas disponíveis"
15   fim se;
```

d)
```
1 leia (SOLICITANTE);
2    se SOLICITANTE <> " " então
3       para I de 1 até 6 passo 1 faça
4          para J de 1 até 5 passo 1 faça
5             AGESALA[I,J] ← 0
6          fim para;
7       fim para;
8       leia SALA;
9       leia HORARIO;
10      I ← SALA;
11      J ← HORARIO;
12      AGESALA[I,J] ← SOLICITANTE
13      se AGESALA[I,J] <> 0 então
14         escreva "Agendamento com sucesso";
15         escreva "Sala:", I, "Horário:", J;
16      fim se;
17      se AGESALA[I,J] = 0 então
18         escreva "Agendamento não realizado"
19          escreva "Não há salas disponíveis"
20       fim se;
```

Gabarito: **Letra B.**

Analisando item por item:

Letra A) **ERRADO**. Itera sobre as salas e horários, mas atribui o solicitante para todas as salas. No final ainda vai escrever que não fez o agendamento (apesar de ter feito em todos os horários).

Letra B) **CERTO**. Itera sobre as salas e horários, e atribui o solicitante somente quando não houver nada agendado na sala. Ao fazer o agendamento, modifica I e J para 100 (o que vai fazer com que o LOOP interrompa). No final, se I for diferente de 100, significa que o agendamento não foi feito, e escreve "Agendamento não realizado". Perfeito!

Letra C) **ERRADO**. Ao invés de iterar sobre as salas de 1 a 5, atribui o solicitante para a posição I. Isso causará um erro.

Letra D) **ERRADO**. Começa limpando todos os agendamentos! Veja que a linha 5 atribui o para tudo, caso o SOLICITANTE seja informado. Depois pede para informar sala e horário... isso é diferente do que foi pedido no enunciado da questão.

Portanto, **Letra B**.

25. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)

```
algoritmo
  var numero: inteiro
inicio
 funcao abc(numero)
  se(numero < = 1)
   retorne numero
  senao
   retorne numero * abc(numero - 1)
 fim-se
fim
mostre abc(4)
```

fim

Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

a) 24

b) 0

c) 12

d) 4

e) 15

Gabarito: **Letra A**.

Trata-se de uma implementação recursiva da função fatorial.

Analisando a execução de abc(4), temos:

abc(4) = 4*abc(3)

mas abc(3) = 3*abc(2), ou seja,

abc(4) = 4*3*abc(2)

mas abc(2) = 2*abc(1), ou seja,

abc(4) = 4*3*2*abc(1)

mas abc(1) = 1, ou seja,

abc(4) = 4*3*2*1 = 24, **Letra A**.

26. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)

```
algoritmo
var numero: inteiro
inicio
numero = 12

  se (numero mod 2 = 0) entao
escreva ("A")
  senao
escreva ("B")
fim-se

  se (numero > 12) entao
    escreva ("C")
fim-se

fim
```

Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

a) B

b) A

c) AC

d) C



e) BC

Gabarito: **Letra B**.

Pessoal, não fica mais fácil do que isso! Você só precisa saber que ***mod*** é o operador módulo, ou operador resto, que retorna o resto de uma divisão do primeiro número pelo segundo número. Portanto, 12 mod 2 é 0, porque 12 divido por 2 é 6, e deixa **resto 0**.

O primeiro teste **se(numero mod 2 = 0)** será verdadeiro, e será executado **escreva("A")**.

O segundo teste **se(numero > 12)** será falso, porque 12 não é maior que 12, e nada será feito.

Portanto, o resultado final será apenas a impressão de A, que é a opção dada na **Letra B**.

27. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)

```
inicio
 funcao abc(n : inteiro)
  inicio
 a = 0;
  b = 1;
  c = 1;
 para i = 1 ate i < n faca
     c = a + b
```

```
    a = b
    b = c
  fim-para
  retornar c
 fim
mostrar abc(5)
fim
```

Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

a) 4

b) 2345

c) 1235

d) 1234

e) 5

Gabarito: **Letra E.**

abc(5) vai executar a função abc com n=5, e o loop vai iterar de i=1 até 4. Acompanhando o Loop, temos:

iteração i=1, a=0, b=1, c=1, n=5
c = a + b = 0 + 1 = 1
a = b = 1
b = c = 1

iteração i=2, a=1, b=1, c=1, n=5
c = a + b = 1 + 1 = 2

a = b = 1
b = c = 2

iteração i=3, a=1, b=2, c=2, n=5
c = a + b = 1 + 2 = 3
a = b = 2
b = c = 3

iteração i=4, a=2, b=3, c=3, n=5
c = a + b = 2 + 3 = 5
a = b = 3
b = c = 5

Ao término, temos a=3, b=5, c=5, e **ao retornar c, retorna 5**, portanto, **Letra E**.

Observação: Esse algoritmo é a implementação da *Sequência de Fibonacci*, muito comum em provas.

28. (FGV / Auditor Fiscal de Tributos Estaduais (SEFIN RO) / 2018)

Analise o trecho de pseudocódigo a seguir.

```
a := 2;
b := a * 10;
while a < 10 and b > 14
begin
 if a <> b
  begin
    if a > 5
      print (a, b)
    else
      a := a + 3;
  end
 else
```

```
  begin
    a := b – 2;
    print (a);
  end;
 a := a + 3
end;
```

Assinale a opção que exibe o conteúdo integral do resultado que seria produzido numa hipotética execução desse código.

a)

```
2   20
5   20
8   20
```

b)

```
2   20
```

c)

```
8   20
```

d)

```
2
5
8
```

e)

```
2   20
  17
5   20
  14
8   20
  11
```

Gabarito: **Letra C**.

INICIO a=2, b=20

Iteração 1 - teste 2 < 10 and 20 > 14 - TRUE
se 2 <> 20 - TRUE **(operador <> é o operador "diferente)**
se 2 > 5 - FALSE
else
a := a + 3 = 2 + 3 = 5

a := a + 3 = 5 + 3 = 8

FINAL ITERAÇÃO: a=8, b=20

Iteração 2 - teste 8 < 10 and 20 > 14 - TRUE
se 8 <> 20 - TRUE
se 8 > 5 - TRUE
print(a, b) - 8, 20
a:= a + 3 = 8 + 3 = 11

FINAL ITERAÇÃO: a=11, b=20

Iteração 3 - teste 11 < 10 and 20 > 14 - FALSE

FINAL PROGRAMA: a=11, b=20

No final, só será exibido no console os números 8 e 20. Portanto, a **Letra C** está correta.

29. (FCC / Técnico Judiciário (TST) / 2017 / Programação / Apoio Especializado)

Atenção: Para responder à questão, considere o algoritmo adaptado na forma de pseudocódigo abaixo.

```
var real: r
   inteiro: n, aux
início
 repita
   leia (n)
enquanto (n <= 1)
```

```
r ← 1.0
para aux de 2 até n passo 1 faça
  r ← r + 1.0 / aux
fim_para
exiba(r)

    fim
```

O algoritmo apresentado
a) utiliza a condição enquanto incorretamente, pois ela deve vir antes da instrução leia.
b) resolve corretamente r = 1 + 1/2 + 1/3 + 1/4 +...+ 1/n, para n maior do que 1.
c) exibirá o valor 3.083 se for lido o valor 4 para n.
d) gerará um erro de buffer overflow para valores de n maiores do que 10.
e) tenta resolver a equação r = 1 + 1/2 + 1/4 +...+ 1/n, mas ocorrerá um erro se n for ímpar.

Gabarito: **Letra B**.

a) utiliza a condição enquanto incorretamente, pois ela deve vir antes da instrução leia.

ERRADO. Existem as duas formas de loop: ENQUANTO-REPITA e REPITA-ENQUANTO. Na primeira o teste é feita no início, na segunda o teste é feito no final. Não há problema.

b) resolve corretamente r = 1 + 1/2 + 1/3 + 1/4 +...+ 1/n, para n maior do que 1.

CERTO. O loop itera agregando "1/aux" em "r" em cada iteração, "aux" variando de 2 até n.

c) exibirá o valor 3.083 se for lido o valor 4 para n.

ERRADO. Para n=4, temos r= 1+ 1/2 + 1/3 + 1/4 = 25/12 = 2.083333...

d) gerará um erro de buffer overflow para valores de n maiores do que 10.

ERRADO. Não existe nenhum buffer para "estourar". A cada iteração é feita apenas uma soma decimal de números, e essa operação nunca vai estourar buffer.

e) tenta resolver a equação r = 1 + 1/2 + 1/4 +...+ 1/n, mas ocorrerá um erro se n for ímpar.



ERRADO. Não existe nenhum teste para verificar a paridade de "n", e nenhuma operação que falhe para n ímpar.

Portanto, **Letra B** está correta.

30. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Considere que há 3 categorias para pagantes de pensões alimentícias: a primeira engloba os que pagam até 1 valor base (R$ 900.00), a segunda os que pagam de 2 até 4 valores base e a terceira os que pagam acima de 4 valores base. Um programador apresentou o trecho em pseudocódigo abaixo como solução para identificar os pagantes destas 3 categorias.

```
Var valor: real
...
  imprima("Qual é o valor da pensão alimentícia? ")
 leia(valor)
 escolha (valor)
   caso 900.00: imprima("Categoria 1")
   caso 1800.00, 2700.00, 3600.00: imprima("Categoria 2")
   senão imprima("Categoria 3")
   fim_escolha
....
```

Um Técnico Programador, ao analisar o trecho acima, afirma corretamente que

a) não há erro de lógica nem de sintaxe.

b) o comando escolha deve ser substituído por um conjunto de comandos condicionais (se) aninhados que trate os valores da variável valor (do tipo real) como solicitado.

c) o comando escolha é o mais adequado para a solução, pois os valores das pensões são múltiplos de 900.00.

d) embora haja erro de sintaxe no comando escolha, a lógica da solução atende de forma correta o que foi solicitado no problema.

e) para que a lógica da solução fique correta, basta trocar o tipo da variável valor para inteiro e retirar os .00 dos valores de cada caso do comando escolha.

Gabarito: **Letra B**.

Considerando o código abaixo (apenas foram incluídas nr. de linhas para referência):

```
1: imprima("Qual é o valor da pensão alimentícia? ")
2: leia(valor)
3: escolha (valor)
4:    caso 900.00: imprima("Categoria 1")
5:    caso 1800.00, 2700.00, 3600.00: imprima("Categoria 2")
6:    senão imprima("Categoria 3")
7: fim_escolha
```

Há erro de lógica, pois a estrutura ESCOLHA-CASO seleciona valores exatos, e o requisito era de valores até os limites (que inclui qualquer valor entre os limites). A estrutura correta para testar se um valor está até 900.00 seria valor < 900.00, e esse tipo de teste só pode ser feito com um condicional SE (uma vez que, como dito, ESCOLHA-CASO só trabalha com valores exatos).

É difícil falar de erro de sintaxe em pesudo-código (pois esse tipo de linguagem não possui regras formais rígidas), mas as linguagens que oferecem a estrutura de controle ESCOLHA-CASO normalmente utilizam apenas 1 valor para cada caso, diferentemente do que acontece na linha 5. Também houve omissão do ":" depois de "senão" na linha 6. Por último, não é comum a utilização de uma variável real (variável com parte decimal) - a maior parte das linguagens só aceita essa estrutura com números inteiros.

a) não há erro de lógica nem de sintaxe.

ERRADO. Conforme foi dito, existem os 2 tipos de erro.

b) o comando escolha deve ser substituído por um conjunto de comandos condicionais (se) aninhados que trate os valores da variável valor (do tipo real) como solicitado.

CERTO. Condicionais aninhados são ótimos para realizar testes comparativos. Nesse caso, o código seria:

```
3: se (valor < 900.00) então
4:   imprima("Categoria 1")
5: senão
6:   se (valor < 3600.00) então
7:    imprima("Categoria 2")
8:   senão
9:    imprima("Categoria 3")
10:  fim_se
11: fim_se
```

c) o comando escolha é o mais adequado para a solução, pois os valores das pensões são múltiplos de 900.00.

ERRADO. Em nenhum momento foi dito que o valor das pensões são múltiplos de 900.00, e ESCOLHA-CASO não é uma solução para testar intervalos.

d) embora haja erro de sintaxe no comando escolha, a lógica da solução atende de forma correta o que foi solicitado no problema.

ERRADO. Conforme foi dito, existem os 2 tipos de erro.

e) para que a lógica da solução fique correta, basta trocar o tipo da variável valor para inteiro e retirar os .00 dos valores de cada caso do comando escolha.

ERRADO. Isso resolveria o problema de utilização de uma variável REAL, mas ainda haveriam problemas lógicos e de sintaxe.

31. (FGV / Analista Legislativo Municipal (CM Salvador) / 2018 / / Tecnologia da Informação)

Observe o trecho de pseudocódigo exibido a seguir.

```
a := 1;
b := 3;
c := 5;
while b <> a and c < 20
{
   if a > c {
     c := c – 2
   }
   else {
      c := c + 2;
      if a + b < c {
         a := b – a;
         b := b + 2
      }
   }
}
print a, b, c;
```

Numa hipotética execução desse código, os valores exibidos seriam:

a) 2, 5, 7;

b) 6, 13, 15;

c) 6, 13, 19;

d) 7, 15, 21;

e) 7, 17, 23.

Gabarito: Letra D.

Passagem 1: a=1, b=3, c=5

3 <> 1 (true) and 5 < 20 (true)

if 1 > 5 (false)

else

c = 5 + 2 = 7

if 1 + 3 < 7 (true)

a = 3 - 1 = 2

b = 3 + 2 = 5

Passagem 2: a=2, b=5, c=7

5 <> 2 (true) and 7 < 20 (true)

if 2 > 5 (false)

else

c = 7 + 2 = 9

if 2 + 5 < 7 (false)

Passagem 3: a=2, b=5, c=9

5 <> 2 (true) and 9 < 20 (true)

if 2 > 9 (false)



else

c = 9 + 2 = 11

if 2 + 5 < 11 (true)

a = 5 - 2 = 3

b = 5 + 2 = 7

Passagem 4: a=3, b=7, c=11

7 <> 3 (true) and 11 < 20 (true)

if 3 > 11 (false)

else

c = 9 + 2 = 13

if 3 + 7 < 11 (true)

a = 7 - 2 = 5

b = 7 + 2 = 9

Em uma análise parcial:

- Passagem 1: a=1, b=3, c=5
- Passagem 2: a=2, b=5, c=7
- Passagem 3: a=2, b=5, c=9
- Passagem 4: a=3, b=7, c=11

A única saída do loop é se A for igual a B (não existe nenhuma alternativa com A igual a B) ou C > 20. "C" começa em 5 e cresce de 2 em 2. Portanto, o loop é interrompido quando C = 21, ou seja, **letra D**.

32. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

A expressão a seguir especifica que: 1 será adicionado a x, se x for maior que 0; 1 será subtraído de x, se x for menor que 0; o valor de x será mantido, se x for igual a zero.

Se (x > 0) então x++; senão if (x < 0) x-- ;

Gabarito: **CERTO**

Em partes:

- 1 será adicionado a x, se x for maior que 0 --> Se x > 0 então x = x + 1;
- 1 será subtraído de x, se x for menor que 0; --> Senão Se x < 0 então x = x - 1;
- o valor de x será mantido, se x for igual a zero --> Senão ... (nada)

Portanto

Se (x > 0) então x++;

Senão

Se (x < 0) então x--;

Vale a crítica ao examinador que usou "Se" para a primeira condicional, e "if" para a segunda... é um ou outro!

33. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)



O pseudocódigo a seguir, após executado, apresentará como resultado 2.370.

```
inteiro contador = 1;
inteiro exp = 1;
real y = 0;
real aux = 1;
real n = 1;

faça {
  y = (1 + (1 / n));

  enquanto (exp <= contador) {
   aux = y * aux;
   exp++;
  }

  exp = 1;
  escreva(aux);

  contador++;
  aux = 1;
  n++;

} enquanto (contador <= 2);
```

Gabarito: **ERRADO**

Passagem 1: contador=1, exp=1, y=0, aux=1, n=1

y = (1 + (1 / n(1))) = 2

enquanto (exp(1) <= contador(1)) (true)

aux = y(2) * aux(1) = 2

exp = exp(1)++ = 2

enquanto (exp(2) <= contador(1)) (false)

exp=1

escreva(aux); //2

contador(1)++ = 2

aux=1;

n(1)++ = 2

enquanto(contador(2) <= 2);

Passagem 2: contador=2, exp=1, y=2, aux=1, n=2

y = (1 + (1 / n(2))) = 1.5

enquanto (exp(1) <= contador(2)) (true)

aux = y(1.5) * aux(1) = 1.5

exp = exp(1)++ = 2

enquanto (exp(2) <= contador(2)) (true)

aux = y(1.5) * aux(1.5) = 2.25

exp = exp(2)++ = 3

```
enquanto (exp(3) <= contador(2)) (true)
exp=1
escreva(aux); //2.25
contador(2)++ = 3
aux=1;
n(2)++ = 3
enquanto(contador(3) <= 2);
```

Portanto, o último número impresso é 2.25, Gabartito **ERRADO**.

34. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

O pseudocódigo a seguir, após executado, apresentará como resultado 13.

```
funcao X (n) {
  se (n == 1 ou n == 2) então
    retorne n;
  senão
    retorne X (n-1) + n * X (n-2);
}
escreva X(4);
```

Gabarito: **CERTO**.

X(4)

se(4 == 1 ou 4 == 2) (falso)

senão

retorne X(3) + 4 * x (2);

X(3)

se(3 == 1 ou 3 == 2) (falso)

senão

retorne X(2) + 3 * x(1)

X(2)

se(2 ==1 ou 2 == 2) (verdadeiro)

retorne 2

X(1)

se(1 ==1 ou 1 == 2)(verdadeiro)

retorne 1

Portanto:

X(4) = X(3) + 4 * x (2)

X(4) = (X(2) + 3 * x(1)) + 4 * 2 =

X(4) = (2 + 3*1) + 8 = 2 + 3 + 8 = **13**.



35. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

A expressão aritmética a seguir tem valor igual a 12.0.

2^3/2^{6/2+1}-5*2-3^{2-1}

Gabarito: **ERRADO**

Na regra de prioridade dos operadores temos:

1. Blocos
2. Exponenciação
3. Multiplicação, Divisão e Resto
4. Soma e Subtração

Resolvendo os blocos temos:

2^3/2^{6/2+1}-5*2-3^{2-1} --> 2^3/2^{3+1}-5*2-3^{1} --> 2^3/2^{4}-5*2-3^{1}

Resolvendo as exponenciações temos:

2^3/2^4-5*2-3^1 --> 8/16-5*2-3

Resolvendo as multiplicações/divisões temos:

8/16-5*2-3 --> 1.5-10-3

Resolvendo as somas/subtrações temos:

0.5-10-3 --> -12.5

Portanto, não é 12.0, gabarito **ERRADO**.

36. (FUNRIO / Técnico Legislativo (CM SJM) / 2018 / / Técnico em Informática)

Observe o algoritmo abaixo:

```
algoritmo "SJM"
var
  X, Y, CT : inteiro
Inicio
   X <- 13
   Y <- 13
   para CT de 0 ate 5 passo 2 faca
        X <- X - 1
        Y <- Y + 1
        escreva(X:3,Y:3)
   fimpara
fimalgoritmo
```

Após a execução, esse algoritmo irá gerar a seguinte sequência de números:

a) 12 11 10 14 15 16.

b) 14 12 16 11 18 10.

c) 12 14 11 15 10 16.

d) 14 15 16 12 11 10.

e) 12 16 14 10 15 11.

Gabarito: **Letra C**.

Entende-se que o comando "escreva(X:3,Y:3)" vai escrever a variável X em 3 posições e Y em 3 posições. Como ambas variáveis tem 2 dígitos, será escrito 2 dígitos e um espaço.

iteracao 0: ct=0, x=13, y=13
x = x - 1 = 13 - 1 = 12
y = y + 1 = 13 + 1 = 14

escreva(**12 , 14**)
iteracao 1: ct=2, x=12, y=14
x = x - 1 = 12 - 1 = 11
y = y + 1 = 14 + 1 = 15
escreva(**11 , 15**)
iteracao 2: ct=4, x=11, y=15
x = x - 1 = 11 - 1 = 10
y = y + 1 = 15 + 1 = 16
escreva(**10 , 16**)
iteracao 3: ct=6 (sai do loop, porque 6 > 5)

Logo, fica escrito **12 15 11 15 10 16**, **letra C.**

37. (FUNRIO / Técnico Legislativo (CM SJM) / 2018 / / Técnico em Informática)

As figuras abaixo mostram um algoritmo em (a) e uma janela com a correspondente execução em (b):

```
algoritmo "SJM"
var
  NR1, NR2, TROCA : inteiro
inicio
  ESCREVAL("<<< LEITURA DE DOIS NÚMEROS >>>")
  ESCREVA("NR1 = ")
  LEIA(NR1)
  ESCREVA("NR2 = ")
  LEIA(NR2)
  ESCREVAL("<<< MOSTRA DOS NÚMEROS LIDOS >>>")
  ESCREVAL("NR1 = ",NR1:3," NR2= ",NR2:3)
  ESCREVAL
  ESCREVAL("<<< TROCA DOS NÚMEROS LIDOS >>>")

  <<< BLOCO DAS INSTRUÇÕES DE TROCA >>>

  ESCREVAL("<<< MOSTRA DOS NÚMEROS TROCADOS >>>")
  ESCREVAL("NR1 = ",NR1:3," NR2= ",NR2:3)
fimalgoritmo
```

(a)

```
<<< LEITURA DE DOIS NÚMEROS >>>
NR1 = 7
NR2 = 5
<<< MOSTRA DOS NÚMEROS LIDOS >>>
NR1 =   7 NR2=   5

<<< TROCA DOS NÚMEROS LIDOS PELAS VARIÁVEIS >>>
<<< MOSTRA DOS NÚMEROS TROCADOS >>>
NR1 =   5 NR2=   7

*** Fim da execução.
```

(b)

O procedimento de troca de variáveis pode ser feito de dois modos: (1) com ajuda de uma variável auxiliar (TROCA) ou (2) sem o auxílio dessa variável auxiliar.

As soluções que mostram as instruções que devem compor o <<< BLOCO DAS INSTRUÇÕES DE TROCA >>> para os modos (1) e (2) estão indicadas, respectivamente, na seguinte alternativa:

a)

```
TROCA <- NR1
NR1 <- NR2
NR2 <- TROCA
```
e
```
NR1 <- NR1 + NR2
NR2 <- NR1 - NR2
NR1 <- NR1 - NR2
```

b)

```
TROCA <- NR1
NR1 <- NR2
NR2 <- TROCA
```
e
```
NR1 <- NR1 - NR2
NR2 <- NR1 + NR2
NR1 <- NR1 + NR2
```

c)

```
TROCA <- NR1
NR1 <- NR2
NR2 <- TROCA
```
e
```
NR1 <- NR1 + NR2
NR1 <- NR1 - NR2
NR2 <- NR1 - NR2
```

d)

```
TROCA <- NR1
NR2 <- NR1
NR2 <- TROCA
```
e
```
NR1 <- NR1 - NR2
NR2 <- NR1 + NR2
NR1 <- NR1 + NR2
```

e)

```
TROCA <- NR1
NR2 <- NR1
NR2 <- TROCA
```
e
```
NR1 <- NR1 + NR2
NR2 <- NR1 - NR2
NR1 <- NR1 - NR2
```

Gabarito: **Letra A**.

Existem 3 formas básicas de trocar 2 variáveis de posição:

Modo 1 - Com ajuda de uma variável auxiliar
//Inicial: **NR1 = x, NR2 = y**
TROCA <- NR1 //como NR1 = x, TROCA = NR1 = x
NR1 <- NR2 //como NR2 = y, NR1 = NR2 = y
NR2 <- TROCA //como TROCA = x, NR2 = TROCA = x
//Final: **NR1 = y, NR2 = x**

Modo 2 – sem o auxílio dessa variável auxiliar

Forma do Acúmulo: (+ - -)
//Inicial: **NR1 = x, NR2 = y**
NR1 <- NR1 + NR2 // NR1 = x + y
NR2 <- NR1 - NR2 // mas NR1=x+y, e NR2=y, então NR2 = (x + y) - y = x
NR1 <- NR1 - NR2 // mas NR1=x+y, e NR2=x, então NR1 = (x + y) - x = y
//Resultado: **NR1 = y, NR2 = x**

Modo 2 – sem o auxílio dessa variável auxiliar

Forma da Diferença: NR1 = (- + -)
//Inicial: NR1 = x, NR2 = y
NR1 <- NR1 - NR2 // NR1 = x - y
NR2 <- NR1 + NR2 // mas NR1=x-y, então NR2 = x-y + y = x
NR1 <- NR2 - NR1 // mas NR1=x-y, então NR1 = x - (x - y) = y

//Resultado: **NR1 = y, NR2 = x**

Nos itens:

a) Modo 1 e Modo 2 **CORRETOS**.

b) Modo 1 **CORRETO**, Modo 2 **ERRADO**.

c) Modo 1 **CORRETO**, Modo 2 **ERRADO**.

d) Modo 1 **ERRADO**, Modo 2 **ERRADO**.

e) Modo 1 **ERRADO**, Modo 2 **CORRETO**.

38. (FUNRIO / Analista Legislativo (CM SJM) / 2018 / / Analista em Tecnologia)

Observe o algoritmo abaixo, que mostra o uso de uma função.

```
funcao CM(X:inteiro):inteiro
inicio
se X < 2 entao
   retorne 1
senao
   retorne X * CM(X-1)
fimse
fimfuncao
inicio
M <- 17
N <- 12
para K de 7 ate 4 passo -1 faca
     se K MOD 2 = 0  entao
        N <- N MOD 3
        X <- CM(N)2
     senao
        Y <- 3
        Y <- CM(Y)
     fimse
fimpara
escreval("X = ",X:4," Y = ",Y:4)
fimalgoritmo
```

Após a execução, esse algoritmo irá gerar, respectivamente, os seguintes valores para X e Y:

a) 0 e 3.

b) 0 e 6.

c) 1 e 1.

d) 1 e 3.

e) 1 e 6.

Gabarito: **ANULADA**.

Essa questão não tem resolução possível porque no de iteração, aparecem duas variáveis que não são declaradas em lugar nenhum:

se K MOD 2 = 0 entao

N <- N MOD 3

X <- CM(N)2

senao

Y <- 3

Y <- CM(**Y**)

fimse

As variáveis X e Y não foram declaradas... portanto, questão anulada.

Apesar de não ser possível dizer qual o resultado do algoritmo, é muito claro que a função CM realiza o cálculo do Fatorial pela recursividade.

39. (FGV / Analista de Tecnologia da Informação (BANESTES) / 2018 / / Desenvolvimento de Sistemas)

Analise o código C# a seguir.

```
using System;
    public class X
    {
        public static int Test(int a, int b.)
        {
            while (a != b.) {
                if (a > b.) {a -= b;}
                else {b -= a;}
            }
            return a;
     }
    static void Main(string[] args)
    {
        int x = 36;
        int y = 7;
        Console.WriteLine(X.Test(x, y));
```

```
    }
}
```

O número produzido pela execução desse código é:
a) 1;
b) 2;
c) 3;
d) 7;
e) 36.

Gabarito: **Letra A**.

Analisando iteração por iteração:

a=36, b=7

Iteração 1 - while(a != b) - 36 != 7 - true
 if(a > b) 36 > 7 - true
  a = a - b = 36 - 7 = 29

Iteração 2 - while(a != b) - 29 != 7 - true
 if(a > b) 29 > 7 - true
  a = a - b = 29 - 7 = 22

Iteração 3 - while(a != b) - 22 != 7 - true
 if(a > b) 22 > 7 - true
  a = a - b = 22 - 7 = 15

Iteração 4 - while(a != b) - 15 != 7 - true
 if(a > b) 15 > 7 - true
  a = a - b = 15 - 7 = 8

Iteração 5 - while(a != b) - 8 != 7 - true
 if(a > b) 8 > 7 - true
  a = a - b = 8 - 7 = 1

Iteração 6 - while(a != b) - 1 != 7 - true
 if(a > b) 1 > 7 - false
  b = b - a = 7 - 1 = 6

Iteração 7 - while(a != b) - 1 != 6 - true
if(a > b) 1 > 6 - false
b = b - a = 6 - 1 = 5

Iteração 8 - while(a != b) - 1 != 5 - true
if(a > b) 1 > 5 - false
b = b - a = 5 - 1 = 4

Iteração 9 - while(a != b) - 1 != 4 - true
if(a > b) 1 > 4 - false
b = b - a = 4 - 1 = 3

Iteração 10 - while(a != b) - 1 != 3 - true
if(a > b) 1 > 3 - false
b = b - a = 3 - 1 = 2

Iteração 11 - while(a != b) - 1 != 2 - true
if(a > b) 1 > 2 - false
b = b - a = 2 - 1 = 1

Iteração 12 - while(a != b) - 1 != 1 - false

return a = 1

Portanto, será impresso 1 em tela, resposta **Letra A**.

40. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

Um programador precisa elaborar um método que diga se uma matriz quadrada recebida como parâmetro é a matriz identidade de ordem n. Esse método recebe uma matriz quadrada (mat) e sua ordem (n) como parâmetros, e retorna true, se a matriz recebida for a matriz identidade de ordem n, ou false, caso contrário.

Qual método executa o que foi especificado acima?

a) public static boolean identidade(int mat[][], int n) {

for(int i=0; i<n; i++) {

```
            if(mat[i][i]!=1)
                return false;
            for(int j=i+1; j<n; j++)
                if(mat[i][j]!=0 || mat[j][i]!=0)
                    return false;
        }
        return true;
    }
```

b)
```
public static boolean identidade(int mat[][], int n) {
        int x=1,y=0;


        for(int i=0; i<n; i++)
            for(int j=0; j<n; j++)
                if(i==j)
                    x*=mat[i][j];
                else
                    y*=mat[i][j];
        if(x==1 && y==0)
            return true;
        else
            return false;
    }
```

c)
```
public static boolean identidade(int mat[][], int n) {


        int x=0,y=0;


        for(int i=0; i<n; i++)
            for(int j=0; j<n; j++)
                if(i==j)
                    x+=mat[i][j];
                else
                    y+=mat[i][j];
        if(x==n && y==0)
            return true;
        else
            return false;
    }
```

d)
```
public static boolean identidade(int mat[][], int n) {
```

```
        for(int i=0; i<n; i++)
                for(int j=0; j<n; j++)
                        if((i==j && mat[i][j]==1) || (i!=j && mat[i][j]==0))
                                return true;
         return false;
    }
```

e) public static boolean identidade(int mat[][], int n) {

```
        for(int i=0; i<n; i++)
                for(int j=0; j<n; j++)
                        if((i==j && mat[i][j]!=1) && (i!=j && mat[i][j]!=0))
                                return false;
        return true;
    }
```

Gabarito: **Letra A**.

Uma matriz é dita "matriz identidade de ordem n" se, para a posição M[x][y], o valor será 1 se x = y, e o valor será 0 se x != y. Em termos práticos, essa matriz tem o número 1 na diagonal, e o número 0 nas outras posições. Por exemplo, a matriz identidade de ordem 4 seria:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

Nos algoritmos:

a) public static boolean identidade(int mat[][], int n) {
    for(int i=0; i<n; i++) {
      if(mat[i][i]!=1) // verifica se todos os elementos

```
                // na diagonal são diferentes de 1
        return false; // algum for, a matriz não é identidade
      for(int j=i+1; j<n; j++) // se entrar nesse 'for',
                    // a diagonal já foi verificada.
                    // agora, verifica os outros
                    // elementos
        if(mat[i][j]!=0 || mat[j][i]!=0)
                // se algum outro elemento
                // for diferente de 0
                // não é matriz identidade
         return false;
   }
   return true; // se chegou nesse ponto, a diagnoal é 1,
           // e todos os outros elementos são 0.
  }

  // Esse algoritmo está perfeito!!!

b) public static boolean identidade(int mat[][], int n) {
    int x=1,y=0;
    for(int i=0; i<n; i++)
      for(int j=0; j<n; j++)
        if(i==j) // x vai acumular a multiplicação das diagonais
          x*=mat[i][j];
        else    // y vai acumular a multiplicação das não-diagonais
          y*=mat[i][j];
     if(x==1 && y==0)
       return true;
     else
       return false;
  }
  // O problema desse algoritmo é que, mesmo se houver um número
  // diferente de 0 nas não-diagonais, basta uma não-diagonal ser 0,
  // que y terminará em 0 (0 vezes qualquer coisa é 0). Dessa forma,
  // uma matriz não identidade seria considerada identidade...

c) public static boolean identidade(int mat[][], int n) {
    int x=0,y=0;
    for(int i=0; i<n; i++)
      for(int j=0; j<n; j++)
        if(i==j) // x vai acumular a soma das diagonais
          x+=mat[i][j];
        else    // y vai acumular a soma das não-diagonais
          y+=mat[i][j];
     if(x==n && y==0)
      return true;
```




```
  else
    return false;
  }
 // Se voce tiver em uma diagonal um número 0 e um número 2, a soma
 // ainda será 2, e você não terá a matriz identidade. O mesmo ocorre
 // para uma não-diagonal com valores 1 e -1... a soma ainda é 0, mas
 // não é a matriz identidade.

d) public static boolean identidade(int mat[][], int n) {
    for(int i=0; i<n; i++)
      for(int j=0; j<n; j++)
       if((i==j && mat[i][j]==1) || (i!=j && mat[i][j]==0))
          return true;
   return false;
  }
 // Esse algoritmo está errado porque, no caso do elemento mat[0][0]
 // ser igual a 1, o teste (i==j && mat[i][j]==1) é verdadeiro, e o
 // método já retorna verdadeiro sem testar o resto da matriz...

e) public static boolean identidade(int mat[][], int n) {
    for(int i=0; i<n; i++)
      for(int j=0; j<n; j++)
       if((i==j && mat[i][j]!=1) && (i!=j && mat[i][j]!=0))
          return false;
    return true;
  }
 // O "if" está completamente errado, e sempre vai ser falso.
 // Ele testa, ao mesmo tempo, se i==j && i!=j, e isso nunca
 // será verdadeiro.
```

Portanto, **Letra A** é a correta.

41. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

Qual função recebe como parâmetros uma matriz simétrica contendo números inteiros (mat) e sua ordem (n), e retorna a soma dos elementos dessa matriz?

a)

```
int somaTransp(int mat[][], int n) {
  int soma=0;


for(int i=0; i < n; i++) {
```

```
    soma+=mat[i][i];
    for(int j=i; j < n; j++)
        soma+=2*mat[i][j];
}


return soma;


}
```

b)

```
int somaTransp(int mat[][], int n) {
int soma=0;


for(int i=0; i < n; i++)


    for(int j=i+1; j < n; j++)
        soma+=2*mat[i][j];


return soma;
}
```

c)

```
int somaTransp(int mat[][], int n) {
   int soma=0;


   for(int i=0; i < n; i++) {
       soma+=mat[i][i];
       for(int j=i; j < n; j++)
           soma+=mat[i][j];
 }
 return 2*soma;
}
```

d)

```
int somaTransp(int mat[][], int n) {
  int soma=0;
```

```
for(int i=0; i < n; i++)


     for(int j=0; j <= i; j++)
          soma+=mat[i][j];
  return 2*soma;
}
```

e)

```
int somaTransp(int mat[][], int n) {
int soma=0;


for(int i=0; i < n; i++) {
     soma+=mat[i][i];
     for(int j=0; j < i; j++)
          soma+=2*mat[i][j];
 }
 return soma;
}
```

Gabarito: Letra E.

Uma matriz é simétrica se o elemento $\mathbf{a_{ij} = a_{ji}}$ para quaiquer j e i. Dessa forma, existem 2 maneiras simples de somar todos os elementos de uma matriz NxN:

1. Iterar sobre todas as linhas e colunas somando os elementos (mais fácil); Para isso, só é preciso 2 loops, onde **0 <= i < N**, e**0 <= j < N.**
2. Somar todos os elementos na diagonal, e somar 2 vezes os elementos de um dos lados simétricos.Para isso é preciso de:
    1. Ou um loop 0 <= i < N, e i < j < N (considera a parte simétrica superior-direita).
    2. Ou um loop 0 <= i < N, e 0 <= j < i (considera a parte simétrica inferior-esquerda).

Assim, analisando os algoritmos:

Letra A) ERRADO. Utiliza o método 2.1, mas **i <= j < N**, ao invés de **i < j < N**.

Letra B) ERRADO. Utiliza o método 2.1, mas esquece de somar os elementos na diagonal a_{ii}.

Letra C) ERRADO. Utiliza o método 2.1, mas **i <= j < N**, ao invés de **i < j < N,** e esquece de dobrar o valor (são duas partes simétricas).

Letra D) ERRADO. Utiliza o método 1, mas dobra o valor (nesse método, todo elemento já é contado individualmente)

Letra E) CERTO. Utiliza o método 2.2, com **o <= j < N**, dobra os elementos da parte simétrica, e não esquece de somar os elementos da diagonal a_{ii}.

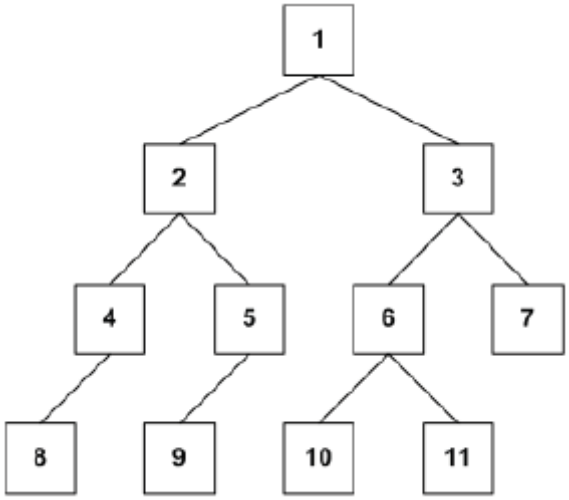
Portanto, **Letra E** está correta.

42. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

Um programador escreveu uma função para percorrer uma árvore binária, recebida como parâmetro, em pós-ordem e inserir em uma pilha, inicialmente vazia, os valores armazenados nos nós dessa árvore, à medida que eles forem sendo visitados. Ao término do percurso, a função retorna a pilha.

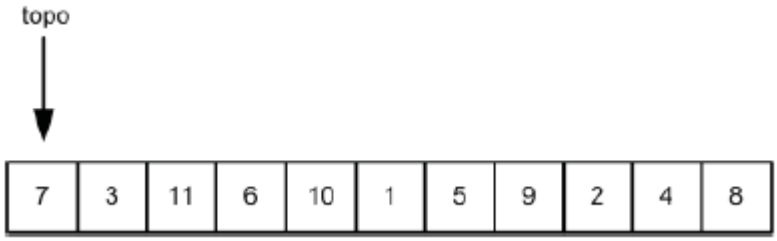
Suponha que a árvore exibida na Figura abaixo seja passada como parâmetro em uma chamada dessa função.



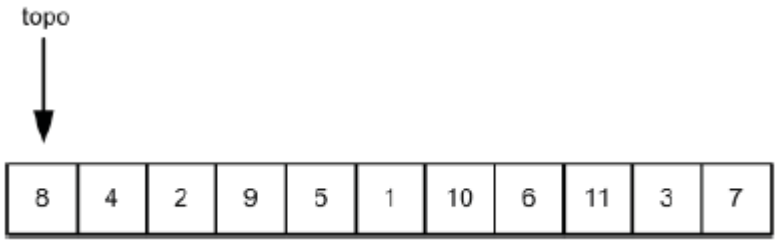


Qual será a configuração da pilha retornada por essa função?

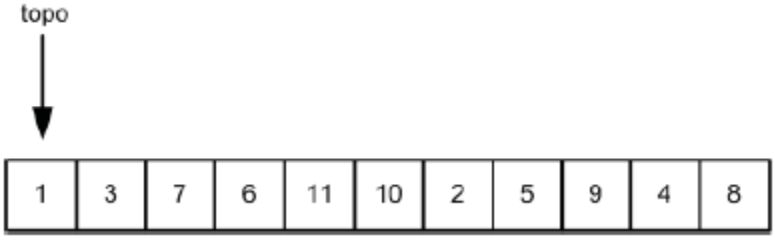
a)



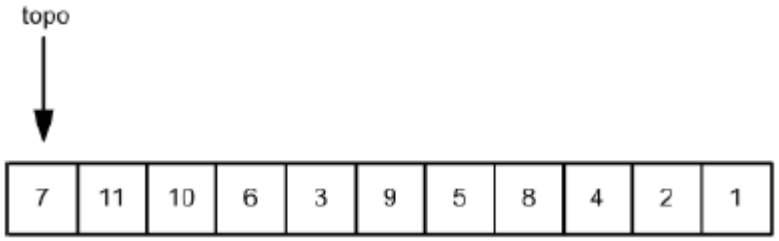
b)



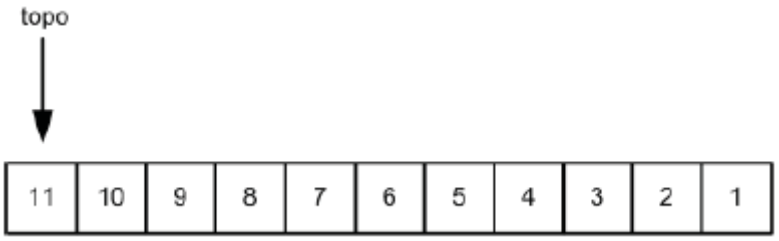
c)



d)



e)



Gabarito: **Letra C**.

Questão muito bonita e inteligente, para dar um diferencial para quem entende de estruturas de dados!

O caminho mais longo é você ler a árvore em pós-ordem, e registrar os valores: 8-4-9-5-2-10-11-6-7-3-1. Ao empilhar os valores, teremos topo-> 1-3-7-6-11-10-2-5-9-4-8. Portanto, resposta correta é a **Letra C.**

Mas existe um caminho bem rápido! A leitura pós-ordem lê o nó por último. Portanto, o último número lido será 1. Logo, o topo da pilha será o 1. A **Letra C** é a única que atende essa condição. Rápido, né?

43. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

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);
```

Gabarito: **CERTO**.

As linhas de 1 a 4 definem a função pfactor, e a linha 6 faz sua invocação, passando o valor 5 como parâmetro. Portanto, temos:

pfactor(5) = pfactor(4) + pfactor(3) =

= [pfactor(3) + pfactor(2)] + pfactor(3) = 2*pfactor(3) + pfactor(2) =

= 2*[pfactor(1) +pfactor(2)] + pfactor(2) = 3*pfactor(2) + 2*pfactor(1) =

= 3*[pfactor(1) + pfactor(0)] + 2*pfactor(1) = 5*pfactor(1) + 3*pfactor(0)

Porém, a função pfactor retorna 1, se num = 1 ou num = 0, logo:

pfactor(5) = 5*1 + 3*1 = **8**.

Portanto, item **CERTO**.



3 – Sistemas de numeração e aritmética computacional



Sistemas numéricos são diferentes formas de representar os números. Um conjunto de cinquenta e duas laranjas, por exemplo, pode ser representado pelos algarismos 52 em decimal (base 10), mas também por 110100 em binário (base 2), 64 em octal (base 8), 34 em hexadecimal (base 16) ou qualquer outra representação com base definida. *Beleza, até aqui?*

Estamos tão acostumados a um só sistema que é raro pensarmos que o uso do sistema decimal é apenas uma das infinitas possibilidades. Como estamos tratando de computação, no entanto, podemos restringir bastante os sistemas utilizados. **O binário é o sistema numérico das máquinas, dos "fios" com correntes elétricas que apresentam dois estados: ligado e desligado.**

Outros sistemas muito utilizados são o octal e o hexadecimal, também pelo fato de utilizarem bases que são potências de 2 ($2^3 = 8$ e $2^4 = 16$). Nós esperamos que, ao final dessa aula, cada aluno entenda a piadinha infame que diz que: ***"Só existem 10 tipos de pessoas no mundo: aquelas que entendem binário e aquelas que não".*** Se não entenderem, perguntem! :)

Os sistemas numéricos possuem características em comum na representação dos números. A partir da compreensão dessas regras simples, se torna bem mais fácil converter números de um sistema para outro.



A partir de um número XYZ, ABC qualquer, em num sistema numérico com base n qualquer, temos que as regras a seguir se aplicarão sempre:

- Z unidades da base n elevada a 0, ou seja, $n^0 = 1$;
- Y unidades da base n elevada a 1, ou seja, $n^1 = n$;
- X unidades da base n elevada a 2, ou seja, n^2;
- A unidades da base elevado a -1, ou seja, $1/n$;
- B unidades da base elevado a -2, ou seja, $1/n^2$;
- C unidades da base elevado a -3, ou seja, $1/n^3$;

Percebam que a potência passa de positiva para a parte inteira do número (antes da vírgula) para negativa na parte fracionária (depois da vírgula). Vamos testar um exemplo das regras apresentadas para um número da base 10 (sistema decimal): 431,975. De acordo com a decomposição mostrada, cada algarismo do número se aplica a uma potência da base, dependendo da sua posição no número:

4	3	1,	9	7	5
$4x10^2$	$3x10^1$	$1x10^0$	$9x10^{-1}$	$7x10^{-2}$	$5x10^{-3}$

ou ainda, aplicando as potências:

4	3	1,	9	7	5
4x100	3x10	1x1	9/10	7/100	5/1000

Do mesmo modo, para um número binário, ou seja, que utiliza base 2, temos as mesmas regras. O número 110,111 possui a decomposição:

1	1	0,	1	1	1
$1x2^2$	$1x2^1$	$0x2^0$	$1x2^{-1}$	$1x2^{-2}$	$1x2^{-3}$

O número 110,111 é, portanto: (4+2) = 6 na parte inteira, e (1/2+1/4+1/8) = 0,875 na parte fracionária. Assim, 110,111 em binário (base 2) é o mesmo que 6,875 em decimal (base 10). **O sistema decimal utiliza 10 algarismos diferentes para representar os números, de 0 a 9.** É o sistema que melhor conhecemos, talvez por termos 10 dedos nas mãos.



O sistema binário utiliza 2 algarismos diferentes para representar os números, 0 e 1. É o sistema numérico das linguagens de máquina, que são compostos por dois estados, ligado e desligado, que são representados por 0 e 1. Algumas representações mostram a base em subscrito logo após ao número para mostrar que é um número binário, por exemplo, 110101_2 (base 2), ou ainda, $110101_{bin.}$

Já o sistema octal utiliza 8 algarismos diferentes para representar os números, de 0 a 7. É muito utilizado para representar números binários de forma mais compacta, tendo em vista que a cada dígito octal temos a representação de três bits. Mais adiante veremos como isso se dá com mais detalhes. Muitas vezes, números do sistema octal são representados com um pequeno "o" (Ex: 721_o ou ainda, 721_{oct}).

Um sistema um pouco diferente, somente na questão de representação de seus algarismos, é o sistema hexadecimal (base 16). Ele utiliza 16 algarismos diferentes para representar os números. Como não há algarismos que representam diretamente os números maiores que nove (não temos um símbolo único para representar 12, p. ex.), precisamos pegar letras emprestado do alfabeto.

Em hexadecimal, precisamos de letras para representar os números dez, onze, doze, treze, quatorze e quinze, ou seja, A, B, C, D, E e F respectivamente. Os números hexadecimais são precedidos de "ox" (um zero e um xis) para indicar que a representação é de um número na base 16, por exemplo, oxFA. **Saibam que conversões entre sistemas numéricos são bastante cobrados pelas bancas.** *Ok?*

Por isso vamos passar por cada um para não restar dúvidas. Vale a pena exercitar para ficar afiado e não perder tempo precioso na prova. Vamos analisar as conversões para binário, tendo em vista que as outras conversões podem tirar proveito desse sistema para facilitar as contas. **Primeiro, conversão de decimal (Base 10) para binário (Base 2).**

A maneira mais prática para converter um número decimal para binário é realizar uma decomposição do número base 10 em potências de 2. *Como é isso, professor? Entendi nada!* Temos que qualquer número decimal X pode ser decomposto em uma soma de potências de 2, de forma que $X = x_1 \times 2^0 + x_2 \times 2^1 + x_3 \times 2^2 + x_4 \times 2^3$... onde x_i é sempre zero ou um. *Eita, professor, agora é que complicou tudo!*

Um exemplo sempre ajuda! Vamos seguir a decomposição do número 486 nos passos abaixo! Vamos lá...

- Primeiramente buscamos a maior potência de 2 que seja menor que o número 486;
- Lembremos que as potências de dois são 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, etc;
- 256 é menor que 486, e esta é a maior potência de 2 menor que o número decomposto – o próximo seria 512 que seria maior que 486;
- *E o que é 486?* Galera, 486 = 256 + 230. Ou seja, 256 (maior potência de 2 menor que 486) + 230 = 486;
- Temos que continuar a decomposição, pois o que sobrou não é uma potência de 2. Então, vamos decompor 230.
- Busquemos a maior potência de 2 que seja menor que o número 230 (como fizemos acima);

- *Que número é esse?* É o 128, uma vez que o próximo número (256) é maior que o número 230.
- *E o que é 230?* É 128 + 102. E seguimos: 102 pode ser decomposto como 64 + 38; 38 pode ser decomposto como 32 + 6; 6 = 4 + 2; e fim!
- Logo, 486 pode ser decomposto como 486 = 256 + 128 + 64 + 32 + 4 + 2, ou ainda 486 = $1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^1 + 0 \times 2^0$

A partir da decomposição, preenchemos o número no sistema binário da seguinte forma:

Potência de 2	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decomposição	1	1	1	1	0	0	1	1	0

O número decimal (base 10) 486 convertido em binário (base 2), portanto, é 111100110. **A conversão de octal (base 8) para binário (base 2) se dá a partir da tradução de cada algarismo em octal para um conjunto de três bits (2^3 = 8 possibilidades), começando sempre pela direita.** Desse modo, o número 25173 em octal pode ser convertido em binário como se segue:

Octal	2	5	1	7	3
Binário	010	101	001	111	011

25173 (Octal) é 010101001111011 (Binário). **O inverso também é verdadeiro, ou seja, o que eu quero dizer é que um número binário pode ser separado de três em três bits para ser convertido em octal, sempre começando pela direita (parte inteira).** O número 010101001111011 em binário pode ser convertido para octal como é mostrado abaixo:

Binário	010	101	001	111	011
Octal	2	5	1	7	3

De modo semelhante, a conversão de hexadecimal para binário se dá com um conjunto de 4 bits (2^4 = 16 possibilidades), lembrando que nós vamos começar pela direita (parte inteira). Em outras palavras, o número AFC02 pode ser convertido em binário traduzindo cada posição hexadecimal no número respectivo em binário, como é apresentado abaixo:



Hexadecimal	A	F	C	0	2
Binário	1010	1111	1100	0000	0010

O número AFC02 em hexadecimal é convertido em 10101111110000000010. Também para o sistema hexadecimal a conversão inversa funciona, ou seja, para cada 4 algarismos binários, basta convertê-lo para seu respectivo algarismo hexadecimal. *E a conversão de decimal (base 10) para octal (base 8)?* **As conversões de decimal para octal são mais simples se utilizando do sistema binário como um intermediário.**

Desse modo, a melhor forma é converter o número decimal em binário e de binário para octal. Este último é bem simples como vimos. Vamos converter o número 3289 em decimal para octal. Decompondo o número 3289 em potências de 2 (vide conversão decimal para binário) temos que 3289 = 2048 + 1024 + 128 + 64 + 16 + 8 + 1, ou seja, 3289 em binário é igual a

Potência de 2	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decomposição	1	1	0	0	1	1	0	1	1	0	0	1

Depois da primeira conversão, basta convertermos de binário para octal, i.e., agrupando os bits de três em três, começando pela direita, o número 110011011001:

Binário	110	011	011	001
Octal	6	3	3	1

O número 3289 em decimal é representado pelo número 6331 em octal. **Conversão de decimal (base 10) para hexadecimal (base 16). Do mesmo modo que a conversão anterior, de números decimais para hexadecimais, a melhor forma é primeiramente converter para base 2 (binário).** O número 861 em decimal pode ser convertido em hexadecimal da seguinte forma:

- 861 = 512 + 256 + 64 + 16 + 8 + 4 + 1, ou seja, em binário é igual a 1101011101;

Separando o número binário de quatro em quatro bits, sempre começando pela direita, temos:

Binário	0011	0101	1101
Hexadecimal	3	5	D

O número decimal 861 é igual a 35D em hexadecimal. **As conversões de octal para decimal e hexadecimal para decimal são triviais quando utilizamos o sistema binário como intermediário, pois a conversão dos algarismos octal e hexadecimal para binário são diretos (3 bits para octal e 4 bits para hexadecimal).** Somente como exemplo, vamos converter 634 octal e 9FA para decimal:

Octal	6	3	4
Binário	110	011	100

Para converter de binário para decimal, basta somar as potências de 2:

Potência de 2	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Número binário	1	1	0	0	1	1	1	0	0

Ou seja, 110011100 em binário em decimal é igual a $2^8 + 2^7 + 2^4 + 2^3 + 2^2 = 256\ 128 + 16 + 8 + 4 = 412$. O número 634 em octal é, portanto, 412 em decimal.

Convertendo o número hexadecimal 1FA em binário, temos:

Hexadecimal	9	F	A
Binário	1001	1111	1010

Para converter de binário para decimal, basta somar as potências de 2:

Potência de 2	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Número binário	1	0	0	1	1	1	1	1	1	0	1	0

O número 9FA em hexadecimal é igual a 100111111010 em binário que, por sua vez, é igual ao número em decimal: $2^{11} + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1 = 2048 + 256 + 128 + 64 + 32 + 16 + 8 + 2 = 2554$ Desse modo,

chegamos ao fim da aula teórica em que cobrimos os principais sistemas de numeração e as suas conversões. Vamos praticar um pouco com exercícios ***E a piada, todo mundo entendeu?*** :)

QUESTÕES COMENTADAS

1. (CESPE - 2015 – TRE/RS – Técnico Judiciário - Operação de Computadores) Com relação a sistemas de numeração, é correto afirmar que o equivalente, em decimal, do binário 1001,101 é:

a) 11,5.

b) 9,3.

c) 11,3.

d) 9,5.

e) 9,625.

Comentários:

A conversão de números fracionários é algo que pode pegar muitos alunos desavisados de surpresa, mas não vocês! :) Conforme vimos na parte teórica, a parte inteira do número tem potências positivas e crescentes à medida em que o algarismo aparece mais à esquerda, e negativas e decrescentes na parte fracionária. A decomposição do número 1001,101 se dá da seguinte forma:

1	0	0	1,	1	0	1
$1X2^3$	$0X2^2$	$0X2^1$	$1X2^0$	$1X2^{-1}$	$0X2^{-2}$	$1X2^{-3}$

A parte inteira é, portanto, (8 + 1) = 9, e a parte fracionária (1/2 + 1/8) = 0,625. O número 1001,101 em binário é representado em decimal por 9,625. Conforme vimos em aula, a resposta corresponde à letra E, ou seja, 1001,101 (base 2) é igual a 9,625 (base 10).

Gabarito: E

---

2. (SRH - 2015 – UERJ – Analista de Sistemas) A conversão do binário 11001111.01 para hexadecimal é:

a) $8F.2_{16}$

b) $DF.1_{16}$

c) $CE.4_{16}$

d) $CF.4_{16}$

Comentários:

Esse exercício é ótimo para testar nossos conhecimentos de conversão de binário para hexadecimal, com parte fracionária. O pulo do gato é completarmos os bits até termos conjuntos de 4 para convertermos diretamente para hexadecimal, da seguinte forma:

Binário	1100	1111,	0100
Hexadecimal	C	F,	4

Os bits em vermelho completamos para formar um conjunto de 4 bits. Sabemos que devemos completar, depois da vírgula, sempre à direita, pois, da mesma forma que no sistema decimal, temos que 3,4 é o mesmo que 3,40 ou mesmo 3,40000. Se fosse necessário completar a parte inteira, a ordem seria inversa, ou seja, completar à esquerda, pois 3,2 é o mesmo que 03,2 ou 00003,2. O número 11001111,01 em binário é, portanto, igual a CF,4 em hexadecimal

Gabarito: D

3. (Prefeitura do Rio de Janeiro - 2014 – Câmara Municipal do Rio de Janeiro – Analista Legislativo - Administração de Servidores) O número hexadecimal 9C é representado nos sistemas binário e decimal, respectivamente, como:

a) 10011100 e 167

b) 10111010 e 167

c) 10011100 e 156

d) 10111010 e 156

Comentários:

A questão nos facilita bastante quando pede a resposta tanto em binário (base 2) quanto em decimal (base 10). Isto porque, como vimos na aula, é muito mais simples converter números hexadecimais primeiramente em binário, para depois em decimal. 9C em binário é igual a:

Hexadecimal	9	C
Binário	1001	1100

E 10011100 em decimal é igual a $2^7 + 2^4 + 2^3 + 2^2 = 128 + 16 + 8 + 4 = 156$. A resposta, portanto é 10011100 e 156, letra C.

Gabarito: C

---

4. (CESPE - 2015 – Telebrás – Engenheiro - Engenharia da Computação) Situação hipotética: Um circuito lógico compara as entradas X e Y e fornece, na saída S, o valor lógico 1, se X > Y, ou 0, em caso contrário. Assertiva: Nessa situação, se a entrada X for representada pelo número binário 00100111 e a entrada Y pelo número hexadecimal 2A, então a saída S será igual a 1.

Comentários:

O circuito apresenta valor 1 se X > Y e valor 0 se X <= Y. Para comparar dois números, eles devem estar numa mesma base, o que não é o caso. Temos várias maneiras de solucionar o exercício, pois podemos converter os números 00100111 (binário) e 2A (hexadecimal) para qualquer base. Vamos fazer de três formas possíveis, mas que fique claro que, na hora da prova, é melhor escolher somente um dos números para converter para a base do outro, nesse exemplo, converter 00100111 para hexadecimal ou o número 2A para binário.

1ª forma: 00100111 em hexadecimal é igual a 0010 = 2 e 0111 = 7, ou seja, 27 em hexadecimal. Qual dos números é maior em hexadecimal, 2A ou 27? Claramente, 2A, ou seja, Y > X e a saída do circuito é 0.

2ª forma: 2A em binário é igual a 2 = 0010 e A = 1010, ou seja, 00101010. O número 00101010 é maior que 00100111 (só comparar os bits da esquerda para a direita e avaliar qual é maior). Chegamos, portanto, à mesma conclusão, Y > X e o circuito retorna 0.

3ª forma: converter os dois números para decimal. 00100111 é igual a $2^5 + 2^2 + 2^1 + 2^0 = 32 + 4 + 2 + 1 = 39$, já 2A é igual a 00101010 = $2^5 + 2^3 + 2^1 = 32 + 8 + 2 = 42$. Como 42 > 39, Y > X e o circuito retorna 0.

Gabarito: E

---

5. (FCC - 2015 – DPE-RR – Engenheiro Eletrônico ou Mecatrônico) Um comerciante de peças de bicicleta utiliza-se da numeração hexadecimal como código para representar o preço de custo dos seus artigos, de maneira que, quando um cliente pede um desconto, ele sabe até que valor pode chegar. Para tanto, representou a parte inteira por um número hexadecimal, e os centésimos de real por outro número sempre com dois algarismos. Assim, o preço de custo de um produto que apresenta o código 2F3C é, em reais,

a) 65,20.

b) 47,60.

c) 129,70.

d) 242,15.

e) 39,34.

Comentários:

A parte que o comerciante utiliza para os valores em centavos sempre tem dois algarismos hexadecimais, ou seja, em 2F3C, ele utiliza 2F para a parte inteira e 3C para os centavos. Utilizando a técnica mostrada na aula, basta convertermos para binário para depois converter para base 10 (decimal)

Hexadecimal	2	F	3	C
Binário	0010	1111	0011	1100

A parte inteira do preço é 2F ou 00101111 = $2^5 + 2^3 + 2^2 + 2^1 + 2^0 = 32 + 8 + 4 + 2 + 1 = 47$. A parte que ele utiliza para codificar os centavos é 3C ou 00111100 = $2^5 + 2^4 + 2^3 + 2^2 = 32 + 16 + 8 + 4 = 60$. O preço, portanto é 47,60!

Gabarito: B

---

6. (VUNESP - 2014 – PRODEST-ES – Analista de Tecnologia da Informação - Desenvolvimento de Sistemas) Considere o número 999 na notação decimal. Esse mesmo número, na notação hexadecimal é igual a:

a) 2F6 h

b) 3E7 h

c) 4D6 h

d) 5F7 h

e) 6B6 h

Comentários:

Simples conversão de decimal para binário. Utilizamos uma conversão intermediária de 999 para binário para facilitar a conversão final. Lembrando como vimos na aula que temos que decompor o número em potências de 2. Desse modo, temos que 999 = $2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0$ = 512 + 256 + 128 + 64 + 32 + 4 + 2 + 1.

Em binário, portanto o número 999 é igual a 1111100111. Para converter em hexadecimal temos que separar o número binário de quatro em quatro, começando sempre pela direita e completando o à esquerda o que faltar:

Binário	0011	1110	0111
Hexadecimal	3	E	7

Atenção para os bits em vermelho, completamos à direita para que tenhamos conjuntos de 4 bits. Pela conversão temos que o número decimal 999 é igual a 3E7 na base 16 (hexadecimal).

Gabarito: B

---

7. (FGV - 2015 – TJ-BA – Técnico Judiciário - Tecnologia da Informação) O número binário 11111010 é representado na notação hexadecimal como:

a) F8

b) AF

c) FF

d) FA

e) FB

Comentários:

Devemos separar 11111010 de quatro em quatro bits, começando sempre pela direita. Assim, temos que 1111 = F e 1010 = A. A resposta da questão é FA.

Gabarito: D

---

8. **(FGV - 2014 – DPE-RJ – Técnico Superior Especializado - Suporte)** Na notação hexadecimal, o código binário 1100001111110111 é escrito como

a) C3F

b) C37Fo



c) C_3F_7

d) EF_3C

e) FE_3CA

Comentários:

O número binário 1100001111110111, dividido de quatro em quatro bits, sempre pela direita, resulta em: 1100 0011 1111 0111. Convertendo cada 4 bits num algarismo hexadecimal temos 1100 = C, 0011 = 3, 1111 = F e 0111 = 7, ou seja, o número em hexadecimal é C_3F_7.

Gabarito: C

---

9. (FGV - 2010 - BADESC - Analista de Sistemas - Desenvolvimento de Sistemas) O sistema binário representa a base para o funcionamento dos computadores. Assim, um odômetro binário mostra no display o número 10101111. A representação desse número em decimal e em hexadecimal e o próximo número binário mostrado no display, serão, respectivamente:

a) 175, AE e 10101110

b) 175, EF e 10110000

c) 175, AF e 10110000

d) 191, EA e 10110000

e) 191, FA e 10101110

Comentários:

10101111 em hexadecimal é 1010 = A e 1111 = F, ou seja, AF em hexadecimal. Em decimal, $10101111 = 2^7 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 32 + 8 + 4 + 2 + 1 = 175$. O próximo número, em decimal, seria 176 que, decompondo em potências de 2, $176 = 128 + 32 + 16 = 27 + 25 + 24$. Em binário, portanto, o número 176 é igual a 10110000. A resposta final seria 175, AF e 10110000, ou seja, letra C



Gabarito: C

10. (FGV - 2010 – CODESP-SP - Analista de Sistemas) Se o sistema decimal é utilizado pelos seres humanos, o sistema binário constitui a base para a representação da informação nos computadores. Nesse contexto, um equipamento dispõe de três displays, o primeiro que mostra números em formato decimal, o segundo em binário e o terceiro em hexadecimal, havendo uma correspondência entre as representações. Se o display decimal mostra o número 250, os equivalentes em binário e em hexadecimal mostrarão, respectivamente,

a) 11111010 e FA.

b) 11111010 e FE.

c) 11111010 e FC.

d) 11111110 e FE.

e) 11111110 e FA.

Comentários:

Precisamos converter 250 em binário e em hexadecimal. A maneira mais fácil é utilizar o sistema binário como intermediário, mas, para exercitar, vamos mostrar a forma direta de converter um número decimal em hexadecimal. Para tal, precisamos decompor o número decimal em potências de 16, bem semelhante ao que fizemos na conversão de decimal para binário.

Temos que decompor utilizando a maior potência de 16 que é menor que 250. $16^2 = 256$, que é maior que 250, ou seja, a maior potência de 16 menor que 250 é $16^1 = 16$. Na decomposição, temos, portanto $250 = 15 * 16^1 + 10 * 16^0$. Desse modo, temos que 250 é 15 vezes 16 mais 10 vezes 1, em formato hexadecimal 15 é F e 10 é A, ou seja, 250 em hexadecimal é FA.

Lembrando que a conversão passando pelo binário é bem mais rápida e simples, pois não requer divisões que podem ser complexas na hora da prova. Esse método direto é somente para ilustrar que é possível converter decimal para qualquer base a partir do método da decomposição. Finalmente, convertendo 250 em decimal para base 2 (binário) temos a decomposição: $250 = 128 + 64 + 32 + 16 + 8 + 2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$. 250 em binário, portanto, é 11111010.

Somente como prova dos nove, vamos converter para hexadecimal a partir do binário para termos certeza do cálculo inicial? 11111010 separando de quatro em quatro bits 1111 1010, 1111 = F e 1010 = A, ou seja 150 (base 10) = FA (base 16) = 11111010 (base 2).

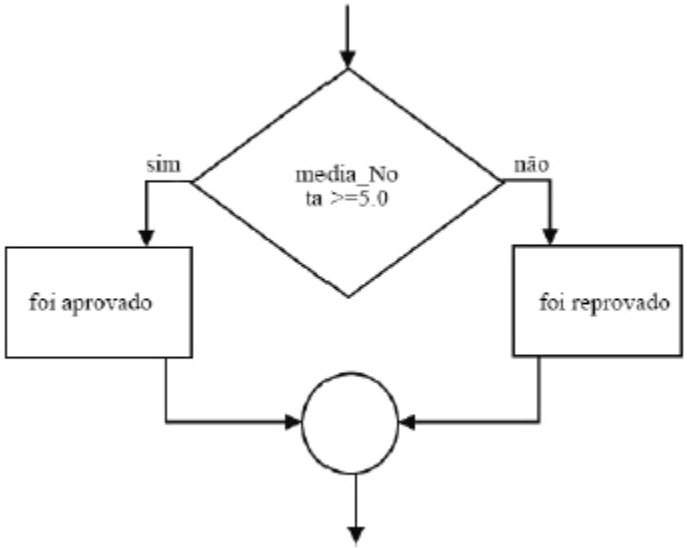
Gabarito: A

ACERTEI	ERREI

LISTA DE QUESTÕES - LÓGICA DE PROGRAMAÇÃO

Lógica de programação

1. (CESPE / Analista Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)



A estrutura lógica presente no diagrama apresentado é do tipo

a) SE ENTÃO.

b) CASO SELECIONE.

c) CASO REPITA.

d) SE ENTÃO SENÃO.

2. (FGV / Analista Legislativo Municipal (CM Salvador) / 2018 / / Tecnologia da Informação)

Expressões lógicas são frequentemente utilizadas em linguagens de programação. Por exemplo, um comando if com a expressão

if not (A and B)

pode ser reescrito, para quaisquer valores lógicos de A e B, com a expressão:

a) A or B

b) not A or not B

c) not A or B

d) not (not A or not B)

e) A and B

3. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 6)

As estruturas de controle de fluxo WHILE e DO...WHILE possuem a mesma finalidade e seus respectivos blocos de comandos são executados pelo menos uma vez em cada uma delas.

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

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

5. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

Uma variável com capacidade de armazenar um baite pode representar valores no intervalo de !512 a 512.

6. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

Para a determinação da parte decimal de um número real, pode-se utilizar a função INT(x), como no exemplo a seguir, onde INT(x) retorna a parte inteira de x.

x = 3.1415926;

escreva x-INT(x)

7. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

Para o seu correto funcionamento, os algoritmos devem ser implementados como um conjunto de métodos e mensagens.

8. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

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.

9. (CESPE / Oficial Técnico de Inteligência / 2018 / / 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.

O laço de repetição na estrutura de repetição para será executado pelo menos uma vez.

10. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

A estrutura de controle seleção não pode ser utilizada nas situações em que duas alternativas dependam de uma mesma condição — uma de a condição ser verdadeira e outra de a condição ser falsa.

11. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

Uma das vantagens de se construir um algoritmo por meio do pseudocódigo é o fato de que a passagem do algoritmo para uma linguagem de programação qualquer se torna uma atividade quase que instantânea.

12. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 9)

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.

13. (FUNRIO / Analista Legislativo (CM SJM) / 2018 / / Analista em Tecnologia)

A programação de computadores necessita das estruturas de controle abaixo referenciadas para que possa ser utilizada com eficiência. Neste contexto, relacione as estruturas de controle a seguir com as características correspondentes.

(EE) ENQUANTO ... FAÇA ... FIM ENQUANTO

(RR) REPITA ... ATÉ ... FIM REPITA

( ) A condição de teste da estrutura é inserida no fim da estrutura de controle.

( ) A condição de teste da estrutura é inserida no início da estrutura de controle.

( ) Se o resultado do teste for FALSO, a execução do programa permanece no loop.

( ) Se o resultado do teste for VERDADEIRO, a execução do programa permanece no loop.

( ) A saída do loop ocorre quando o teste da condição de controle retorna valor FALSO.

( ) A saída do loop ocorre quando o teste da condição de controle retorna valor VERDADEIRO.

A relação correta, de cima para baixo, é:

a) (RR), (EE), (EE), (RR), (RR) e (EE).

b) (EE), (EE), (RR), (RR), (EE) e (RR).

c) (RR), (RR), (EE), (RR), (EE) e (EE).

d) (RR), (EE), (RR), (EE), (EE) e (RR).

e) (EE), (RR), (EE), (EE), (RR) e (RR).

14. (FCC / Estagiário (SABESP) / 2018 / / Ensino Médio Regular)

Considere, por hipótese, que a SABESP utiliza diferentes preços de tarifas para os serviços de abastecimento de água e/ou coleta de esgoto para o município de São Paulo. Para a categoria Residencial/Favela as tarifas são:

Consumo	Valor da Tarifa
0 a 10	6,25/mês
11 a 20	0,71/m³
21 a 30	2,36/m³
31 a 50	7,14/m³
acima de 50	7,89/m³

Foi solicitado a um estagiário propor a lógica de programação para a solução do seguinte problema: ler o valor do consumo de um usuário da categoria Residencial/Favela (variável consumo) e calcular o valor a pagar com base nas tarifas (variável valor).

O Estagiário sugeriu utilizar

a) o tipo básico inteiro para ambas as variáveis.

b) o tipo básico caracter para consumo e o tipo básico inteiro para valor.

c) a instrução leia para ler o valor que o usuário deverá pagar.

d) as instruções se então senão de forma aninhada para avaliar as diferentes faixas de consumo.

e) a instrução escolha caso para avaliar os diferentes tipos de valor que poderão ser pagos pelo usuário.

15. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

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.

16. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

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";
```

17. (FGV / Analista em Tecnologia da Informação e Comunicação (SEPOG RO) / 2017)

Considere o algoritmo em pseudocódigo descrito a seguir.

```
para i=0 até n
```

```
inicio
  j = 1
  enquanto j<n
  inicio
    j = 2 * j
    para k = 0 até j
    inicio
      execute f
    fim
  fim
fim
```

Assinale a opção que indica o número de vezes em que o código irá executar a função f para n igual a 8.
a) 25
b) 153
c) 278
d) 481
e) 587

18. (FCC / Técnico (DPE RS) / 2017 / / Informática)

Considere o seguinte algoritmo em pseudocódigo:

```
Algoritmo Valida
tipo V = vetor [1..4] de inteiro
var vet: V
```




```
indice, numero: inteiro


Inicio
indice←1
enquanto (indice<=4)faça
  leia(numero);
enquanto(..... I ) faça
    imprima("Valor invalido. Digite um valor dentro do limite.")
    leia(numero)
  fim_enquanto
  vet[indice]←numero
  indice←indice + 1
fim_enquanto


para (indice de 1 até 4 passo 1) faça
..... II
fim_para
Fim
```

Para que o algoritmo acima leia quatro valores de anos de 1900 até 2017 e os apresente na tela, a lacuna

a) I deve ser preenchida com numero>=1900 e numero<=2017

b) II deve ser preenchida com leia(vet[indice])

c) I deve ser preenchida com numero<1900 ou numero>2017

d) II deve ser preenchida com imprima ("Valor valido = ", vetor[indice])

e) I deve ser preenchida com numero>=1900 ou numero<=2017

19. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Texto 10A1AAA

O algoritmo a seguir realiza o cálculo do dígito verificador (DV) do código de barras (COD) a ser usado em um sistema de controle de processos.

```
1 inicio
2   tipo VET = vetor [1..11] de inteiros;
3   VET: VCOD;
4   inteiro: I, J, K, COD, SOMA1, SOMA2,
        FATOR, DV;
5   SOMA1, SOMA2, FATOR, DV ← 0;
6   leia COD;
7   para I de 1 até 11 passo 1 faça
8     J ← 11 – I;
9     K ← pot (10,J)
10     VCOD [I] ← COD div K;
11     COD ← COD mod K;
12  fim para;
13  para I de 1 até 11 passo 2 faça
14    SOMA1 ← SOMA1 + VCOD[I];
15  fim para;
16  SOMA1 ← SOMA1 * 3;
17  para I de 2 até 10 passo 2 faça
18    SOMA2 ← SOMA2 + VCOD[I];
19  fim para;
20  SOMA2 ← SOMA2 + SOMA1;
21  FATOR ← SOMA2 div 10;
22  FATOR ← FATOR + 1;
23  FATOR ← FATOR * 10;
24  DV ← FATOR – SOMA2;
25  escreva DV;
26 fim.
```

O algoritmo apresentado no texto **10A1AAA** realiza, entre as linhas 7 e 12, o
a) somatório dos valores dos dígitos parciais, a partir dos valores de cada uma das posições de um vetor.
b) preenchimento de um vetor com os valores de cada umdos dígitos do código de barras lido.
c) preenchimento de um vetor com os somatórios das parcelas das posições ímpares do código de barras lido.
d) cálculo do dígito verificador a partir das parcelas somatórias extraídas do código de barras lido.

20. (... mesma prova da anterior, mesmo código da anterior..)

Considere que, na execução do algoritmo apresentado no texto **10A1AAA,** o valor do código lido na linha 6 seja 12345678901. Nesse caso, o valor da variável SOMA1, imediatamente após a linha 16, será igual a
a) 138.
b) 60.
c) 78.
d) 46.

21. (... mesma prova da anterior, mesmo código da anterior..)

Na execução do algoritmo apresentado no texto 10A1AAA, se o valor de K for 0, então, na linha 10, VCOD[I]
a) receberá o valor de COD.
b) receberá o valor 0.
c) não será atualizado, porque um erro causará a interrupção do algoritmo.
d) receberá o valor 1.

22. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

10 A ← 5;

11 B ← A * -2;

12 C ← A – 1;

13 D ← A – 2;



14 H ← (((4*A)div D)-B) – pot(A,2)mod C;

Considerando a execução do trecho de algoritmo precedente, assinale a opção que apresenta o valor atribuído a H na linha 14.
a) 16
b) –9
c) 15
d) –5

23. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

21 VET[1] ← 1;

22 VET[2] ← 1;

23 para I de 3 até 27 passo 1 faça
24 VET[I] ← VET[I–1] + VET [I–2];

25 fim para;

Considerando a execução completa do trecho de algoritmo precedente, assinale a opção que apresenta o valor armazenado em VET[12].
a) 233
b) 192
c) 144
d) 89

24. (CESPE / Técnico Judiciário (TRT 7ª Região) / 2017 / Tecnologia da Informação / Apoio Especializado)

Em determinada organização, existem cinco salas de reunião e, para cada uma delas, são destinados horários fixos para o agendamento de até seis reuniões por dia. Um especialista em tecnologia da informação criou uma matriz para armazenar os agendamentos solicitados. O sistema percorre a estrutura de dados em busca de uma sala com horário livre para hospedar uma reunião.

Considerando a situação apresentada, assinale a opção cujo trecho de algoritmo apresentado realiza a referida busca.

a)
```
1 leia (SOLICITANTE);
2   para I de 1 até 5 passo 1 faça
3     para J de 1 até 6 passo 1 faça
4       AGESALA[I,J] ← SOLICITANTE
5       escreva "Agendamento com sucesso";
6       escreva "Sala:", I, "Horário:", J;
7     fim para;
8   fim para;
9   se I = 5 então
10    escreva "Agendamento não realizado"
11    escreva "Não há salas disponíveis"
12  fim se;
```

b)
```
1 leia (SOLICITANTE);
2   para I de 1 até 5 passo 1 faça
3     para J de 1 até 6 passo 1 faça
4       se AGESALA[I,J] = " "
5         AGESALA[I,J] ← SOLICITANTE
6         escreva "Agendamento com sucesso";
7         escreva "Sala:", I, "Horário:", J;
8         J ← 100;
9         I ← 100;
10      fim se;
11    fim para;
12  fim para;
13 se I <> 100 então
14   escreva "Agendamento não realizado"
15   escreva "Não há salas disponíveis"
16 fim se;
```

c)
```
1 leia (SOLICITANTE);
2   se SOLICITANTE <> " " então
3     I ← SOLICITANTE;
4     para J de 1 até 6 passo 1 faça
5       se AGESALA[I,J] = " "
6         AGESALA[I,J] ← SOLICITANTE
7         escreva "Agendamento com sucesso";
8         escreva "Sala:", I, "Horário:", J;
9       fim se;
10    fim para;
11  fim se;
12  se AGESALA[I,J] = 30 então
13    escreva "Agendamento não realizado"
14    escreva "Não há salas disponíveis"
```




```
15   fim se;
```

d)
```
1 leia (SOLICITANTE);
2    se SOLICITANTE <> " " então
3       para I de 1 até 6 passo 1 faça
4          para J de 1 até 5 passo 1 faça
5             AGESALA[I,J] ← 0
6          fim para;
7       fim para;
8       leia SALA;
9       leia HORARIO;
10      I ← SALA;
11      J ← HORARIO;
12      AGESALA[I,J] ← SOLICITANTE
13      se AGESALA[I,J] <> 0 então
14         escreva "Agendamento com sucesso";
15         escreva "Sala:", I, "Horário:", J;
16      fim se;
17      se AGESALA[I,J] = 0 então
18         escreva "Agendamento não realizado"
19          escreva "Não há salas disponíveis"
20       fim se;
```

25. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)

```
algoritmo

  var numero: inteiro

inicio

 funcao abc(numero)

   se(numero < = 1)

    retorne numero

   senao

    retorne numero * abc(numero - 1)

 fim-se
```

fim

mostre abc(4)

fim

Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

a) 24

b) 0

c) 12

d) 4

e) 15

26. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)

algoritmo

var numero: inteiro

inicio

numero = 12

se (numero mod 2 = 0) entao

escreva ("A")

senao

escreva ("B")

fim-se



```
se (numero > 12) entao
  escreva ("C")
fim-se

fim
```

Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

a) B

b) A

c) AC

d) C

e) BC

27. (CESPE / Técnico Judiciário (TRE TO) / 2017 / Programação de Sistemas / Apoio Especializado)

```
inicio
 funcao abc(n : inteiro)
  inicio
 a = 0;
  b = 1;
  c = 1;
 para i = 1 ate i < n faca
     c = a + b
     a = b
```




```
    b = c
  fim-para
  retornar c
 fim
mostrar abc(5)
fim
```

Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

a) 4

b) 2345

c) 1235

d) 1234

e) 5

28. (FGV / Auditor Fiscal de Tributos Estaduais (SEFIN RO) / 2018)

Analise o trecho de pseudocódigo a seguir.

```
a := 2;
b := a * 10;
while a < 10 and b > 14
begin
 if a <> b
  begin
    if a > 5
      print (a, b)
    else
      a := a + 3;
  end
```




```
 else
  begin
    a := b – 2;
    print (a);
  end;
 a := a + 3
end;
```

Assinale a opção que exibe o conteúdo integral do resultado que seria produzido numa hipotética execução desse código.

a)

```
2   20
5   20
8   20
```

b)

```
2   20
```

c)

```
8   20
```

d)

```
2
5
8
```

e)

```
2   20
  17
5   20
  14
8   20
  11
```




29. (FCC / Técnico Judiciário (TST) / 2017 / Programação / Apoio Especializado)

Atenção: Para responder à questão, considere o algoritmo adaptado na forma de pseudocódigo abaixo.

```
var real: r
   inteiro: n, aux
início
 repita
   leia (n)
enquanto (n <= 1)
r ← 1.0
para aux de 2 até n passo 1 faça
  r ← r + 1.0 / aux
fim_para
exiba(r)

    fim
```

O algoritmo apresentado
a) utiliza a condição enquanto incorretamente, pois ela deve vir antes da instrução leia.
b) resolve corretamente r = 1 + 1/2 + 1/3 + 1/4 +...+ 1/n, para n maior do que 1.
c) exibirá o valor 3.083 se for lido o valor 4 para n.
d) gerará um erro de buffer overflow para valores de n maiores do que 10.
e) tenta resolver a equação r = 1 + 1/2 + 1/4 +...+ 1/n, mas ocorrerá um erro se n for ímpar.

30. (FCC / Assistente Técnico em Tecnologia da Informação de Defensoria (DPE AM) / 2018 / / Programador)

Considere que há 3 categorias para pagantes de pensões alimentícias: a primeira engloba os que pagam até 1 valor base (R$ 900.00), a segunda os que pagam de 2 até 4 valores base e a terceira os que pagam acima de 4 valores base. Um programador apresentou o trecho em pseudocódigo abaixo como solução para identificar os pagantes destas 3 categorias.

```
Var valor: real

...
```




```
 imprima("Qual é o valor da pensão alimentícia? ")
leia(valor)
escolha (valor)
   caso 900.00: imprima("Categoria 1")
   caso 1800.00, 2700.00, 3600.00: imprima("Categoria 2")
   senão imprima("Categoria 3")
   fim_escolha
....
```

Um Técnico Programador, ao analisar o trecho acima, afirma corretamente que

a) não há erro de lógica nem de sintaxe.

b) o comando escolha deve ser substituído por um conjunto de comandos condicionais (se) aninhados que trate os valores da variável valor (do tipo real) como solicitado.

c) o comando escolha é o mais adequado para a solução, pois os valores das pensões são múltiplos de 900.00.

d) embora haja erro de sintaxe no comando escolha, a lógica da solução atende de forma correta o que foi solicitado no problema.

e) para que a lógica da solução fique correta, basta trocar o tipo da variável valor para inteiro e retirar os .00 dos valores de cada caso do comando escolha.

31. (FGV / Analista Legislativo Municipal (CM Salvador) / 2018 / / Tecnologia da Informação)

Observe o trecho de pseudocódigo exibido a seguir.

```
a := 1;
b := 3;
```

```
c := 5;
while b <> a and c < 20
{
   if a > c {
     c := c – 2
   }
   else {
     c := c + 2;
     if a + b < c {
        a := b – a;
        b := b + 2
     }
   }
}
print a, b, c;
```

Numa hipotética execução desse código, os valores exibidos seriam:

a) 2, 5, 7;

b) 6, 13, 15;

c) 6, 13, 19;

d) 7, 15, 21;

e) 7, 17, 23.

32. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

A expressão a seguir especifica que: 1 será adicionado a x, se x for maior que 0; 1 será subtraído de x, se x for menor que 0; o valor de x será mantido, se x for igual a zero.

Se (x > 0) então x++; senão if (x < 0) x-- ;

33. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

O pseudocódigo a seguir, após executado, apresentará como resultado 2.370.

```
inteiro contador = 1;

inteiro exp = 1;

real y = 0;

real aux = 1;

real n = 1;


faça {

  y = (1 + (1 / n));


  enquanto (exp <= contador) {

   aux = y * aux;

   exp++;

  }
```




```
  exp = 1;

  escreva(aux);


  contador++;

  aux = 1;

  n++;


} enquanto (contador <= 2);
```

34. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

O pseudocódigo a seguir, após executado, apresentará como resultado 13.

```
funcao X (n) {
   se (n == 1 ou n == 2) então
     retorne n;
   senão
     retorne X (n-1) + n * X (n-2);
}
escreva X(4);
```

35. (CESPE / Oficial Técnico de Inteligência / 2018 / / Área 8)

A expressão aritmética a seguir tem valor igual a 12.0.

2^3/2^{6/2+1}-5*2-3^{2-1}

36. (FUNRIO / Técnico Legislativo (CM SJM) / 2018 / / Técnico em Informática)

Observe o algoritmo abaixo:

```
algoritmo "SJM"
var
  X, Y, CT : inteiro
Inicio
   X <- 13
   Y <- 13
   para CT de 0 ate 5 passo 2 faca
        X <- X - 1
        Y <- Y + 1
        escreva(X:3,Y:3)
   fimpara
fimalgoritmo
```

Após a execução, esse algoritmo irá gerar a seguinte sequência de números:

a) 12 11 10 14 15 16.

b) 14 12 16 11 18 10.

c) 12 14 11 15 10 16.

d) 14 15 16 12 11 10.

e) 12 16 14 10 15 11.

37. (FUNRIO / Técnico Legislativo (CM SJM) / 2018 / / Técnico em Informática)

As figuras abaixo mostram um algoritmo em (a) e uma janela com a correspondente execução em (b):

```
algoritmo "SJM"
var
  NR1, NR2, TROCA : inteiro
inicio
  ESCREVAL("<<< LEITURA DE DOIS NÚMEROS >>>")
  ESCREVA("NR1 = ")
  LEIA(NR1)
  ESCREVA("NR2 = ")
  LEIA(NR2)
  ESCREVAL("<<< MOSTRA DOS NÚMEROS LIDOS >>>")
  ESCREVAL("NR1 = ",NR1:3," NR2= ",NR2:3)
  ESCREVAL
  ESCREVAL("<<< TROCA DOS NÚMEROS LIDOS >>>")

  <<< BLOCO DAS INSTRUÇÕES DE TROCA >>>

  ESCREVAL("<<< MOSTRA DOS NÚMEROS TROCADOS >>>")
  ESCREVAL("NR1 = ",NR1:3," NR2= ",NR2:3)
fimalgoritmo
```

(a)

```
<<< LEITURA DE DOIS NÚMEROS >>>
NR1 = 7
NR2 = 5
<<< MOSTRA DOS NÚMEROS LIDOS >>>
NR1 =   7 NR2=   5

<<< TROCA DOS NÚMEROS LIDOS PELAS VARIÁVEIS >>>
<<< MOSTRA DOS NÚMEROS TROCADOS >>>
NR1 =   5 NR2=   7

*** Fim da execução.
```

(b)

O procedimento de troca de variáveis pode ser feito de dois modos: (1) com ajuda de uma variável auxiliar (TROCA) ou (2) sem o auxílio dessa variável auxiliar.

As soluções que mostram as instruções que devem compor o <<< BLOCO DAS INSTRUÇÕES DE TROCA >>> para os modos (1) e (2) estão indicadas, respectivamente, na seguinte alternativa:

a)

```
TROCA <- NR1
NR1 <- NR2
NR2 <- TROCA
```
e
```
NR1 <- NR1 + NR2
NR2 <- NR1 - NR2
NR1 <- NR1 - NR2
```

b)

```
TROCA <- NR1
NR1 <- NR2
NR2 <- TROCA
```
e
```
NR1 <- NR1 - NR2
NR2 <- NR1 + NR2
NR1 <- NR1 + NR2
```

c)

```
TROCA <- NR1
NR1 <- NR2
NR2 <- TROCA
```
e
```
NR1 <- NR1 + NR2
NR1 <- NR1 - NR2
NR2 <- NR1 - NR2
```

d)

```
TROCA <- NR1
NR2 <- NR1
NR2 <- TROCA
```
e
```
NR1 <- NR1 - NR2
NR2 <- NR1 + NR2
NR1 <- NR1 + NR2
```

e)

```
TROCA <- NR1
NR2 <- NR1
NR2 <- TROCA
```
e
```
NR1 <- NR1 + NR2
NR2 <- NR1 - NR2
NR1 <- NR1 - NR2
```

38. (FUNRIO / Analista Legislativo (CM SJM) / 2018 / / Analista em Tecnologia)

Observe o algoritmo abaixo, que mostra o uso de uma função.

```
funcao CM(X:inteiro):inteiro
inicio
se X < 2 entao
   retorne 1
senao
   retorne X * CM(X-1)
fimse
fimfuncao
inicio
M <- 17
N <- 12
para K de 7 ate 4 passo -1 faca
     se K MOD 2 = 0  entao
        N <- N MOD 3
        X <- CM(N)2
     senao
        Y <- 3
        Y <- CM(Y)
     fimse
fimpara
escreval("X = ",X:4," Y = ",Y:4)
fimalgoritmo
```

Após a execução, esse algoritmo irá gerar, respectivamente, os seguintes valores para X e Y:

a) 0 e 3.

b) 0 e 6.

c) 1 e 1.

d) 1 e 3.

e) 1 e 6.

39. (FGV / Analista de Tecnologia da Informação (BANESTES) / 2018 / / Desenvolvimento de Sistemas)

Analise o código C# a seguir.

```
using System;
    public class X
    {
        public static int Test(int a, int b.)
        {
            while (a != b.) {
                if (a > b.) {a -= b;}
                else {b -= a;}
            }
            return a;
     }
    static void Main(string[] args)
    {
        int x = 36;
        int y = 7;
        Console.WriteLine(X.Test(x, y));
   }
}
```

O número produzido pela execução desse código é:

a) 1;
b) 2;
c) 3;
d) 7;
e) 36.



40. (CESGRANRIO / Analista de Sistemas Júnior (TRANSPETRO) / 2018 / / Infraestrutura)

Um programador precisa elaborar um método que diga se uma matriz quadrada recebida como parâmetro é a matriz identidade de ordem n. Esse método recebe uma matriz quadrada (mat) e sua ordem (n) como parâmetros, e retorna true, se a matriz recebida for a matriz identidade de ordem n, ou false, caso contrário.

Qual método executa o que foi especificado acima?

a) public static boolean identidade(int mat[][], int n) {

```
        for(int i=0; i<n; i++) {
                if(mat[i][i]!=1)
                        return false;
                for(int j=i+1; j<n; j++)
                        if(mat[i][j]!=0 || mat[j][i]!=0)
                                return false;
        }
        return true;
}
```

b) public static boolean identidade(int mat[][], int n) {

```
        int x=1,y=0;

        for(int i=0; i<n; i++)
                for(int j=0; j<n; j++)
                        if(i==j)
                                x*=mat[i][j];
                        else
                                y*=mat[i][j];
        if(x==1 && y==0)
                return true;
        else
                return false;
}
```

c) public static boolean identidade(int mat[][], int n) {

```
        int x=0,y=0;

        for(int i=0; i<n; i++)
                for(int j=0; j<n; j++)
```

```
                    if(i==j)
                        x+=mat[i][j];
                    else
                        y+=mat[i][j];
        if(x==n && y==0)
            return true;
        else
            return false;
    }
```

d)
```
public static boolean identidade(int mat[][], int n) {


        for(int i=0; i<n; i++)
            for(int j=0; j<n; j++)
                if((i==j && mat[i][j]==1) || (i!=j && mat[i][j]==0))
                    return true;
         return false;
    }
```

e)
```
public static boolean identidade(int mat[][], int n) {


        for(int i=0; i<n; i++)
            for(int j=0; j<n; j++)
                if((i==j && mat[i][j]!=1) && (i!=j && mat[i][j]!=0))
                    return false;
        return true;
    }
```

41. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

Qual função recebe como parâmetros uma matriz simétrica contendo números inteiros (mat) e sua ordem (n), e retorna a soma dos elementos dessa matriz?

a)

```
int somaTransp(int mat[][], int n) {
  int soma=0;


for(int i=0; i < n; i++) {
    soma+=mat[i][i];
```

```
    for(int j=i; j < n; j++)
        soma+=2*mat[i][j];
}


return soma;


}
```

b)

```
int somaTransp(int mat[][], int n) {
int soma=0;


for(int i=0; i < n; i++)


    for(int j=i+1; j < n; j++)
        soma+=2*mat[i][j];


return soma;
}
```

c)

```
int somaTransp(int mat[][], int n) {
   int soma=0;


   for(int i=0; i < n; i++) {
       soma+=mat[i][i];
       for(int j=i; j < n; j++)
           soma+=mat[i][j];
 }
 return 2*soma;
}
```

d)

```
int somaTransp(int mat[][], int n) {
  int soma=0;


for(int i=0; i < n; i++)
```

```
    for(int j=0; j <= i; j++)
        soma+=mat[i][j];
  return 2*soma;
}
```

e)

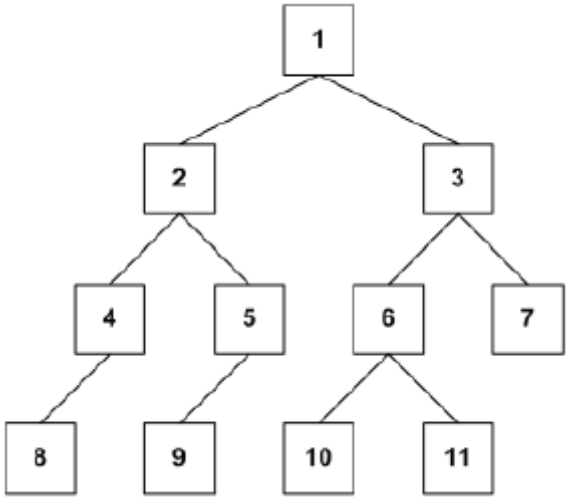
```
int somaTransp(int mat[][], int n) {
int soma=0;


for(int i=0; i < n; i++) {
    soma+=mat[i][i];
    for(int j=0; j < i; j++)
        soma+=2*mat[i][j];
 }
 return soma;
}
```

42. (CESGRANRIO / Engenheiro (PETROBRAS) / 2018 / Eletrônica / Equipamentos Júnior)

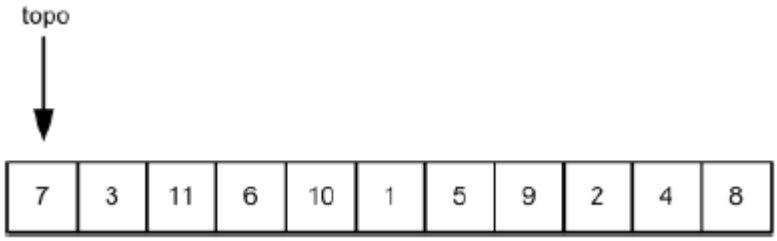
Um programador escreveu uma função para percorrer uma árvore binária, recebida como parâmetro, em pós-ordem e inserir em uma pilha, inicialmente vazia, os valores armazenados nos nós dessa árvore, à medida que eles forem sendo visitados. Ao término do percurso, a função retorna a pilha.

Suponha que a árvore exibida na Figura abaixo seja passada como parâmetro em uma chamada dessa função.

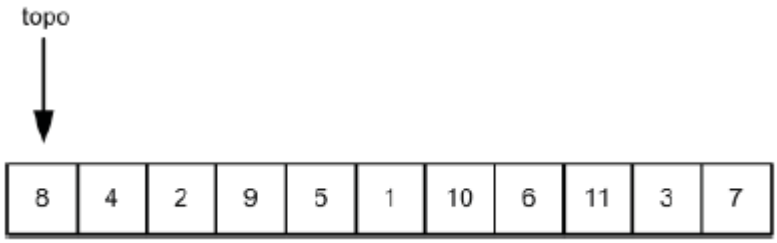


Qual será a configuração da pilha retornada por essa função?

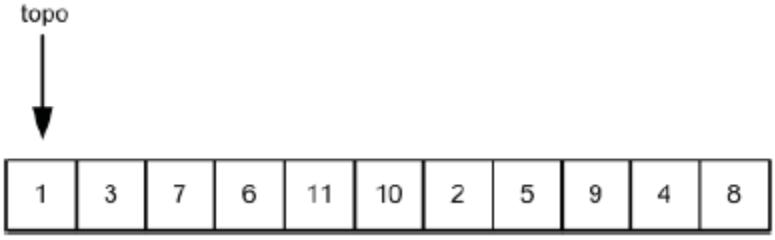
a)



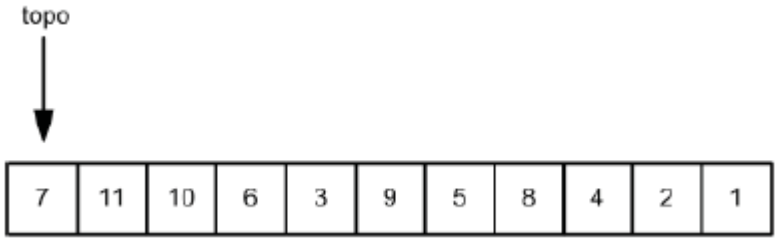
b)



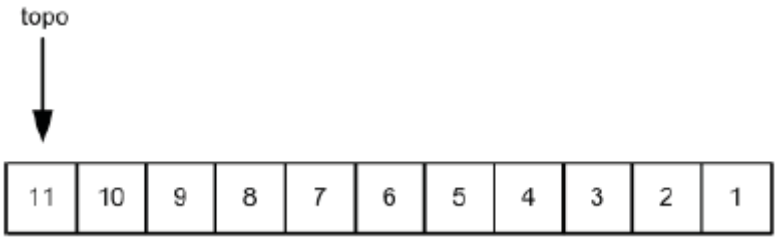
c)



d)



e)



43. (CESPE / Especialista Técnico (BNB) / 2018 / / Analista de Sistema)

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);
```

LISTA DE QUESTÕES - SISTEMAS DE NUMERAÇÃO

(CESPE - 2015 – TRE/RS – Técnico Judiciário - Operação de Computadores) Com relação a sistemas de numeração, é correto afirmar que o equivalente, em decimal, do binário 1001,101 é:

a) 11,5.

b) 9,3.

c) 11,3.

d) 9,5.

e) 9,625.

2. **(SRH - 2015 – UERJ – Analista de Sistemas)** A conversão do binário 11001111.01 para hexadecimal é:

a) $8F.2_{16}$

b) $DF.1_{16}$

c) $CE.4_{16}$

d) $CF.4_{16}$

3. **(Prefeitura do Rio de Janeiro - 2014 – Câmara Municipal do Rio de Janeiro – Analista Legislativo - Administração de Servidores)** O número hexadecimal 9C é representado nos sistemas binário e decimal, respectivamente, como:

a) 10011100 e 167

b) 10111010 e 167

c) 10011100 e 156

d) 10111010 e 156

4. **(CESPE - 2015 – Telebrás – Engenheiro - Engenharia da Computação)** Situação hipotética: Um circuito lógico compara as entradas X e Y e fornece, na saída S, o valor lógico 1, se X > Y, ou 0, em caso contrário. Assertiva: Nessa situação, se a entrada X for representada pelo número binário 00100111 e a entrada Y pelo número hexadecimal 2A, então a saída S será igual a 1.

5. **(FCC - 2015 – DPE-RR – Engenheiro Eletrônico ou Mecatrônico)** Um comerciante de peças de bicicleta utiliza-se da numeração hexadecimal como código para representar o preço de custo dos seus artigos, de maneira que, quando um cliente pede um desconto, ele sabe até que valor pode chegar. Para tanto, representou a parte inteira por um número hexadecimal, e os centésimos de real por outro número sempre com dois algarismos. Assim, o preço de custo de um produto que apresenta o código 2F3C é, em reais,

a) 65,20.

b) 47,60.

c) 129,70.

d) 242,15.

e) 39,34.

6. **(VUNESP - 2014 – PRODEST-ES – Analista de Tecnologia da Informação - Desenvolvimento de Sistemas)** Considere o número 999 na notação decimal. Esse mesmo número, na notação hexadecimal é igual a:

a) 2F6 h

b) 3E7 h

c) 4D6 h

d) 5F7 h

e) 6B6 h



7. **(FGV - 2015 – TJ-BA – Técnico Judiciário - Tecnologia da Informação)** O número binário 11111010 é representado na notação hexadecimal como:

a) F8

b) AF

c) FF

d) FA

e) FB

8. **(FGV - 2014 – DPE-RJ – Técnico Superior Especializado - Suporte)** Na notação hexadecimal, o código binário 1100001111110111 é escrito como

a) C3F

b) C37Fo

c) C3F7

d) EF3C

e) FE3CA

9. **(FGV - 2010 – BADESC – Analista de Sistemas - Desenvolvimento de Sistemas)** O sistema binário representa a base para o funcionamento dos computadores. Assim, um odômetro binário mostra no display o número 10101111. A representação desse número em decimal e em hexadecimal e o próximo número binário mostrado no display, serão, respectivamente:

a) 175, AE e 10101110

b) 175, EF e 10110000

c) 175, AF e 10110000



d) 191, EA e 10110000

e) 191, FA e 10101110

10. **(FGV - 2010 – CODESP-SP - Analista de Sistemas)** Se o sistema decimal é utilizado pelos seres humanos, o sistema binário constitui a base para a representação da informação nos computadores. Nesse contexto, um equipamento dispõe de três displays, o primeiro que mostra números em formato decimal, o segundo em binário e o terceiro em hexadecimal, havendo uma correspondência entre as representações. Se o display decimal mostra o número 250, os equivalentes em binário e em hexadecimal mostrarão, respectivamente,

a) 11111010 e FA.

b) 11111010 e FE.

c) 11111010 e FC.

d) 11111110 e FE.

e) 11111110 e FA.

GABARITO - LÓGICA DE PROGRAMAÇÃO

1. D
2. B
3. Errado
4. Certo
5. Errado
6. Certo
7. Errado
8. Errado
9. Certo
10. Errado
11. Certo
12. Certo
13. D
14. D
15. Certo
16. Certo
17. B
18. C
19. B
20. C
21. C
22. C
23. C
24. B
25. A
26. B
27. E
28. C
29. B
30. B
31. D
32. Certo
33. Errado
34. Certo
35. Errado
36. C
37. A
38. Anulada
39. A
40. A
41. E
42. C
43. Certo

GABARITO - SISTEMA DE NUMERAÇÃO

1. E
2. D
3. C
4. E
5. B
6. B
7. D
8. C
9. C
10. A

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.



Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.