Estratégia
CONCURSOS

Aul

NÃO ATIVARLinguagens de Programação p/ SSAAP-MT (Técnico em Desenvolvimento de Sistemas) - Pós-Edital

Professor: Equipe Informática e TI, Judah Reis, Pedro Henrique Chagas Freitas

LÓGICA DE PROGRAMAÇÃO

Vamos falar sobre Lógica de Programação! *Em 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:

> *Algoritmo: conjunto predeterminado e bem definido de passos destinados à solução de um problema, com um número finito de etapas.*

Professor, você pode dar um exemplo? Sim, o exemplo mais comum da bibliografia é mostrado acima: 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 0 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.

```
1  // class declaration
2  public class ProgrammingExample {
3
4      // method declaration
5      public void sayHello() {
6
7          // method output
8          System.out.println("Hello World!");
9      }
10 }
```

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

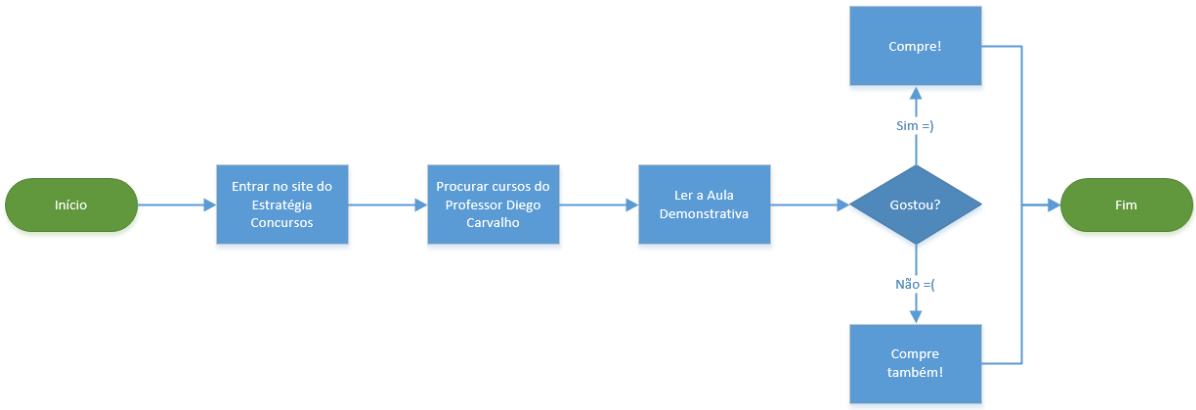
forma genérica de escrever um algoritmo, utilizando uma linguagem simples sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. Trata-se de um pseudocódigo, logo não pode ser executado em um sistema real.

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.

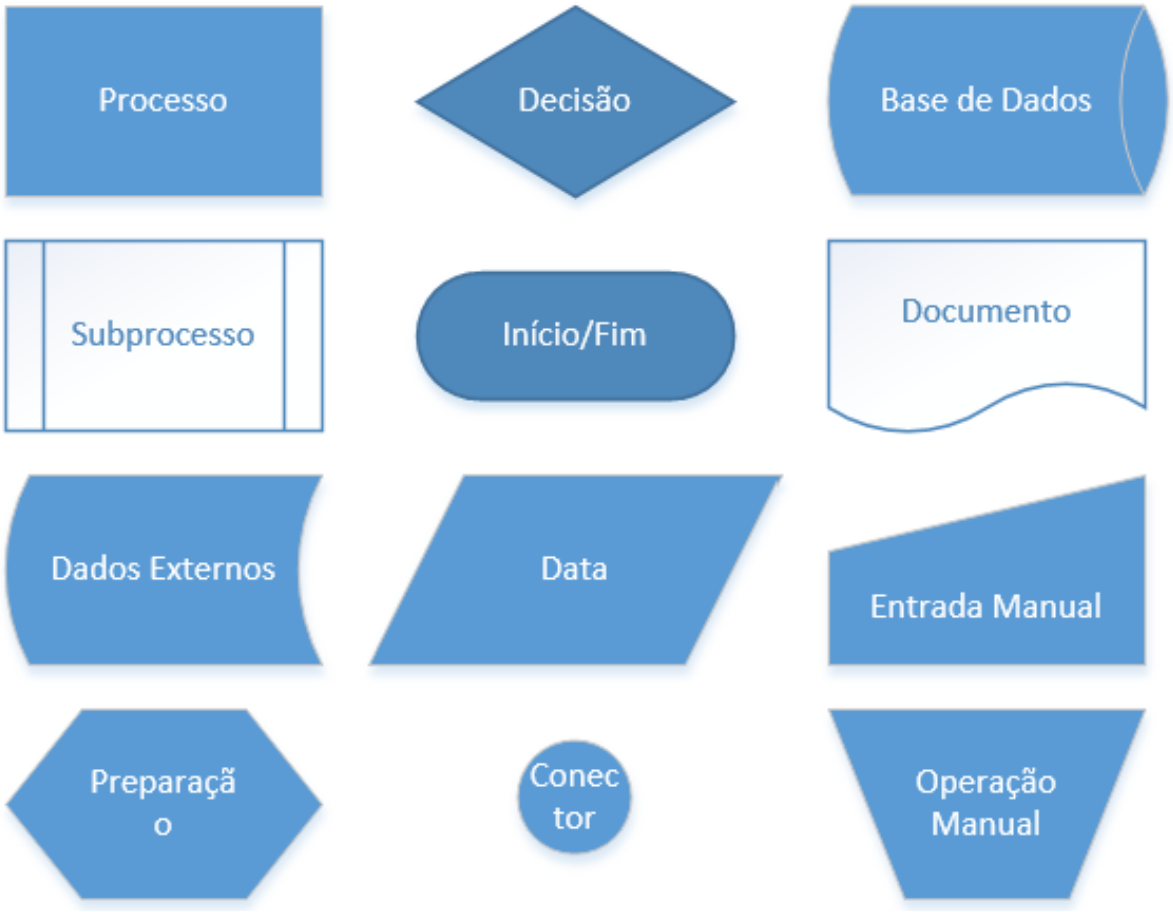
Embora o Portugol seja uma linguagem bastante simplificada, possui todos os elementos básicos e uma estrutura semelhante à de uma linguagem de programação de computadores. Portanto resolver problemas com português estruturado pode ser uma tarefa tão complexa quanto a de escrever um programa em uma linguagem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



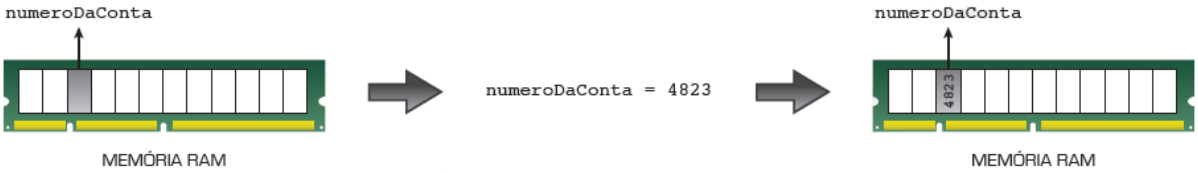
Os principais símbolos de um Fluxograma são:



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!

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.

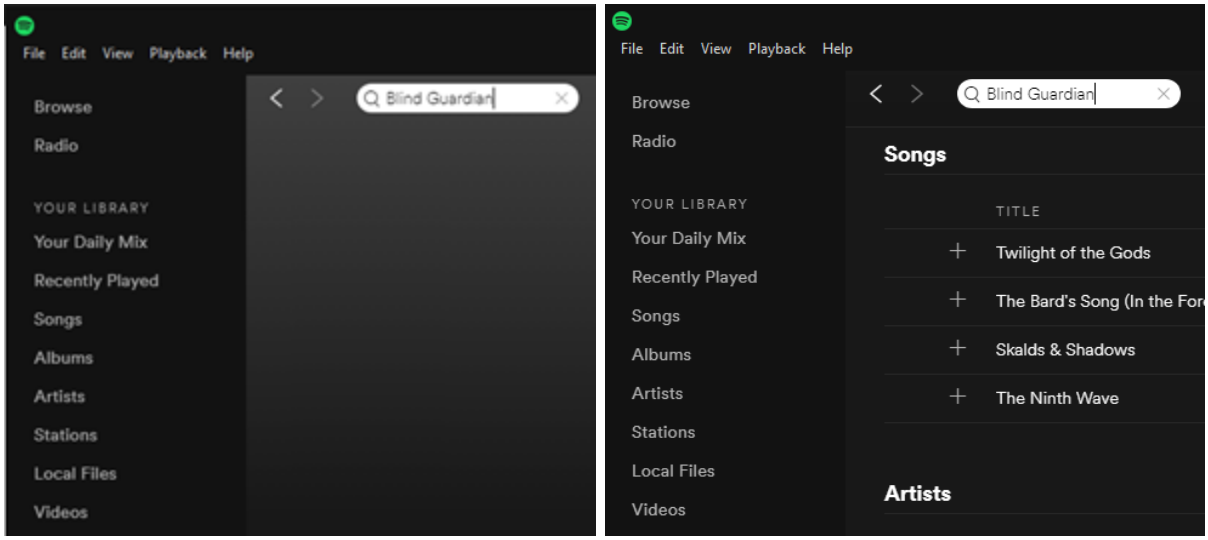


O conteúdo de uma variável pode ser alterado, consultado ou apagado diversas vezes durante a execução de um algoritmo, porém o valor apagado é perdido. *Bacana, mas e a atribuição?* Bem, trata-se de uma notação para associar um valor a uma variável, i.e., armazenar o conteúdo no endereço de memória específico. 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/s^2 de Aceleração Gravitacional; raiz quadrada de 7; etc.

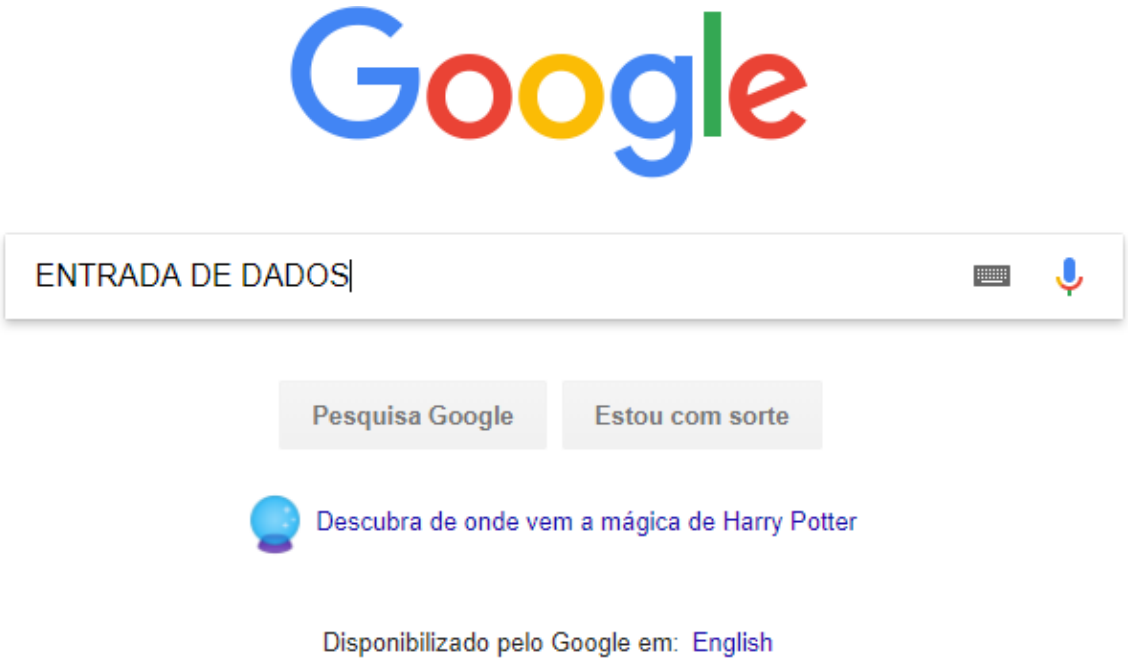
    - Caractere: também conhecido como Literal ou Char, são representações de letras, dígitos e símbolos. Quando colocadas em conjunto, formam um tipo estruturado chamado String ou Cadeia de Caracteres. Ex: 'a', '$', '5', 'D', etc.

- 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.

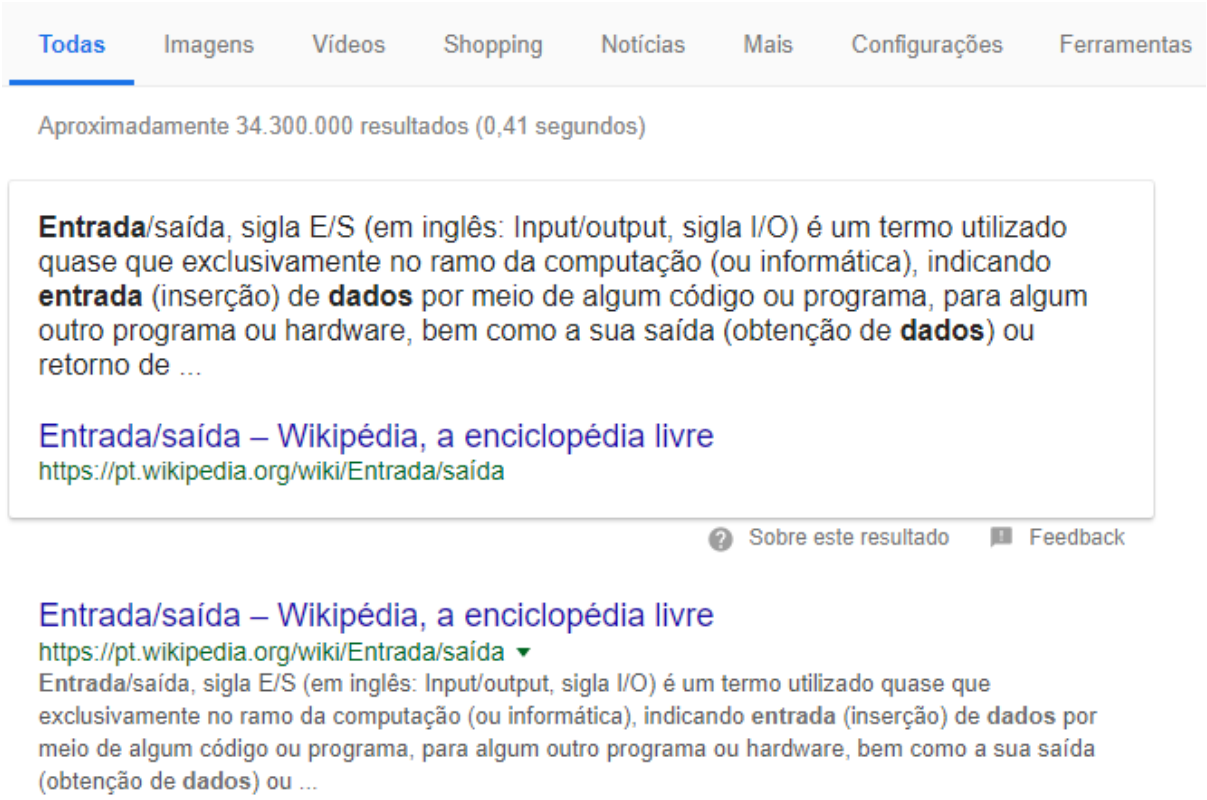
- Tipos Estruturados: são aqueles que podem ser decompostos. *Ora, se eu disser que o nome da bola da copa é Brazuca, é possível decompor esse nome?* Sim, basta dividi-lo em caracteres: 'B', 'r', 'a', 'z', 'u', 'c', 'a'. Há infinitos tipos estruturados[1], pois eles são a combinação de vários outros,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

Operador	Símbolo
Igual a	=
Diferente de	<> ou !=

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

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.

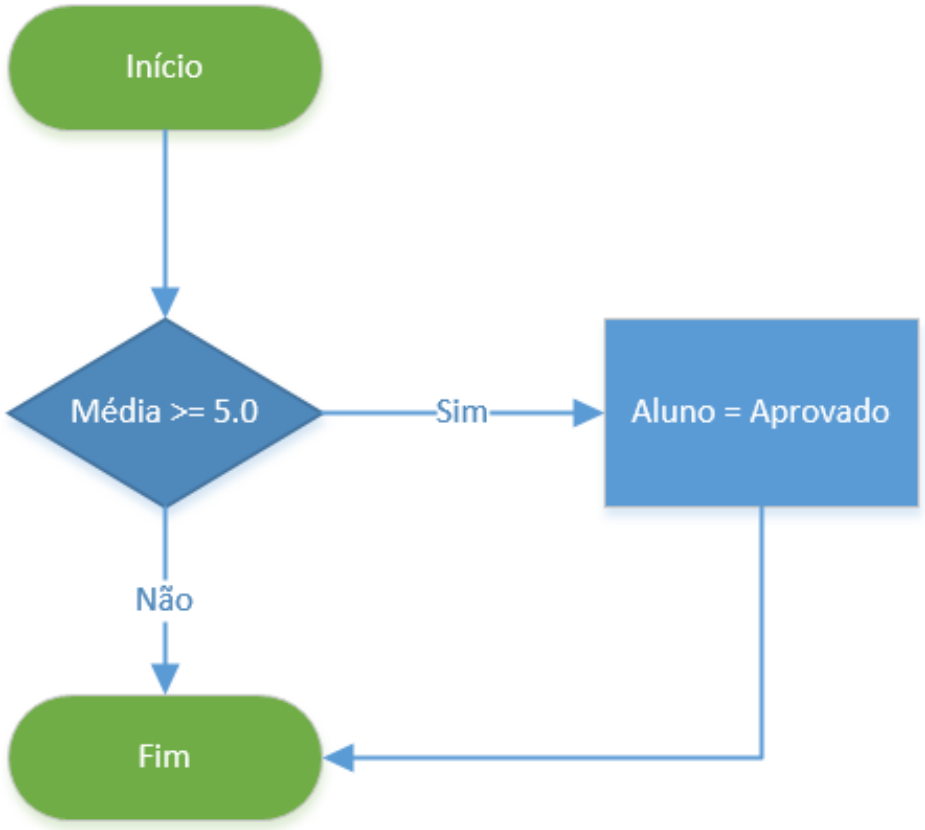
ESTRUTURAS DE CONTROLE: DECISÃO E REPETIÇÃO

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
```

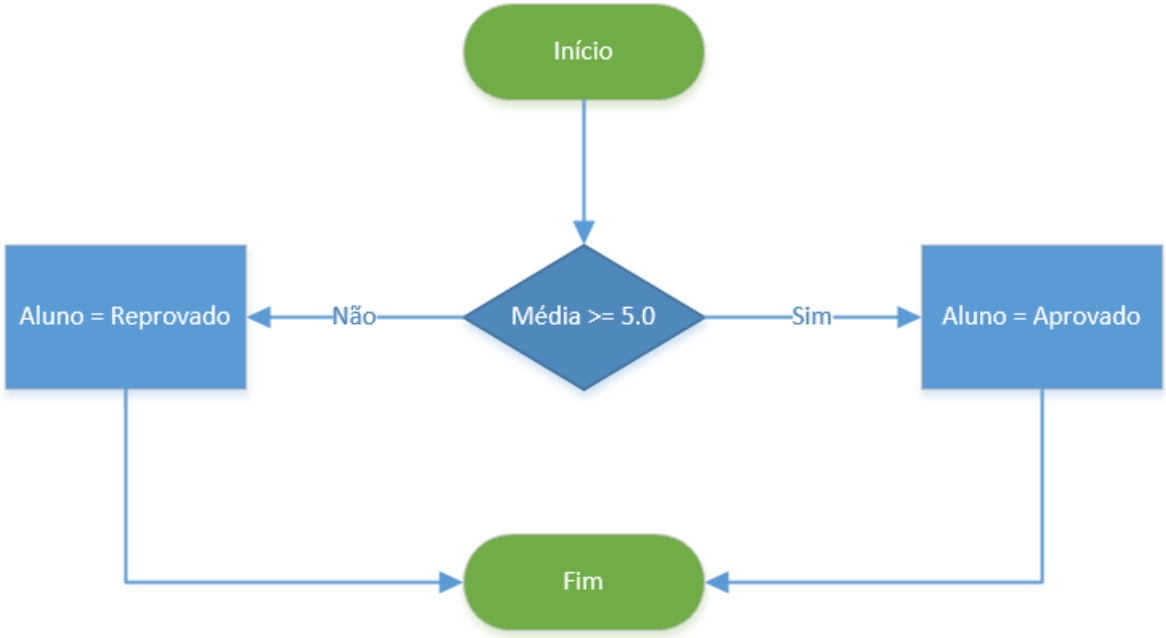


O primeiro caso, e mais simples, é uma estrutura de decisão com um teste (Média >= 5.0). O programa avalia se a variável Média tem o valor maior ou igual a 5.0, no caso de o teste ser verdadeiro a instrução "Aluno = Aprovado" é executada. Em caso negativo, o programa pula a instrução "Aluno = Aprovado"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



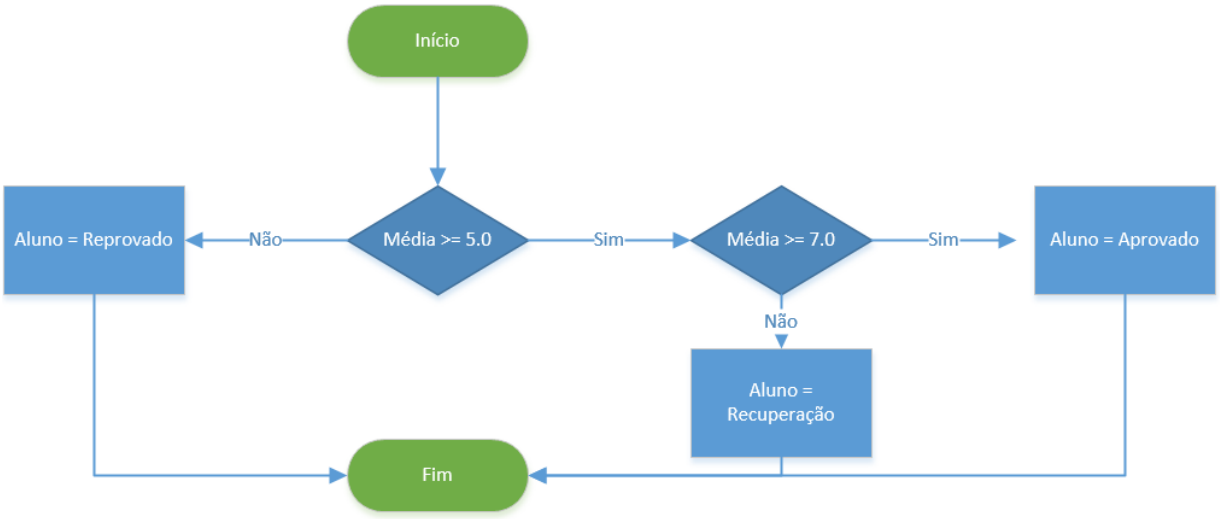
O segundo caso é só um pouquinho mais complexo, onde temos dois fluxos possíveis, um que passa pelo comando "Aluno = Aprovado" e outro que passa pelo "Aluno = Reprovado". Importante ressaltar que, nesse caso, sempre alguma dessas

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:

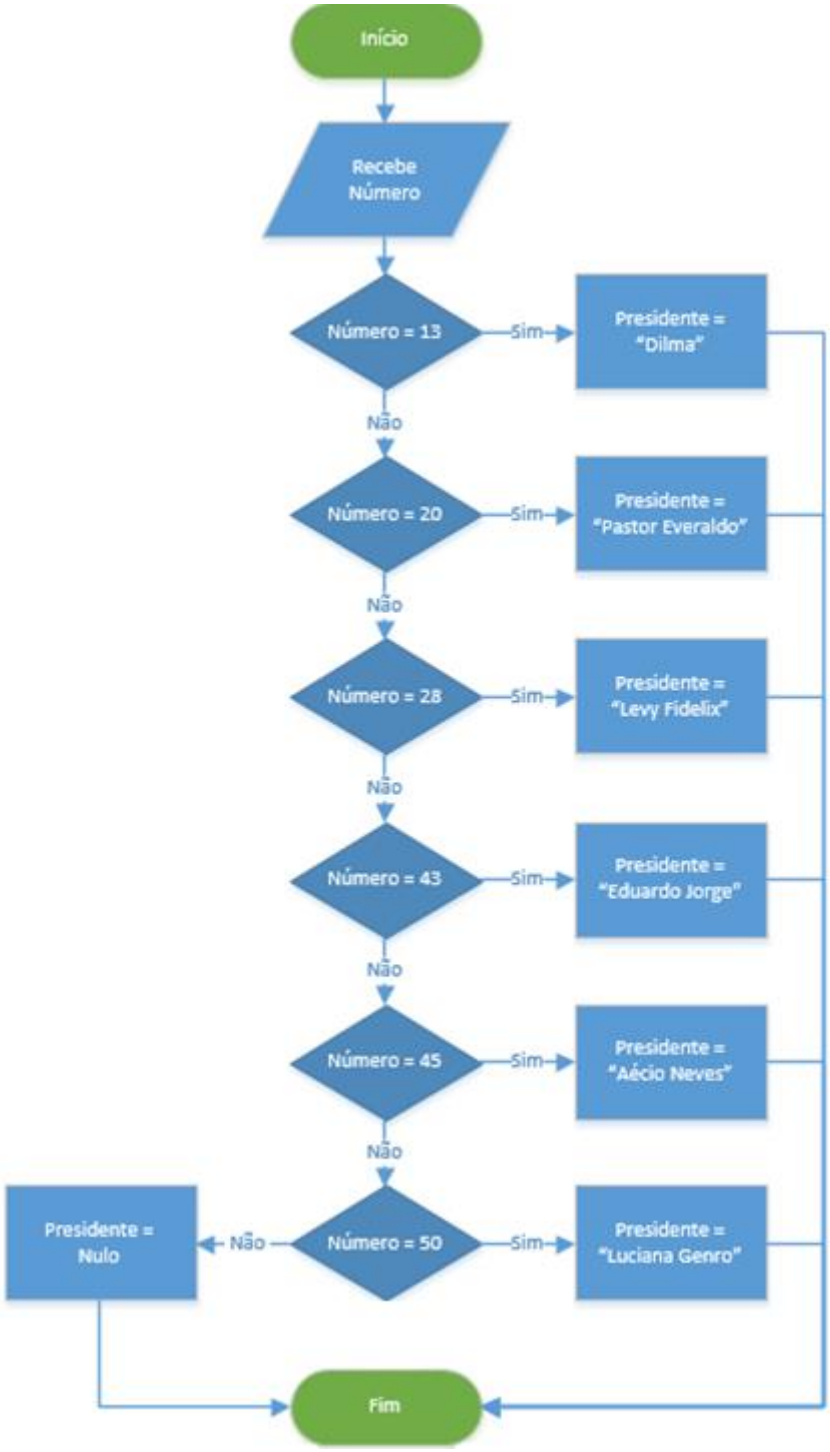
```
Selecione (Número)
      Caso 13: Presidente = "Dilma"
      Caso 20: Presidente = "Pastor Everaldo"
      Caso 28: Presidente = "Levy Fidelix"
      Caso 43: Presidente = "Eduardo Jorge"
      Caso 45: Presidente = "Aécio Neves"
      Caso 50: Presidente = "Luciana Genro"
      Caso Outro: Presidente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

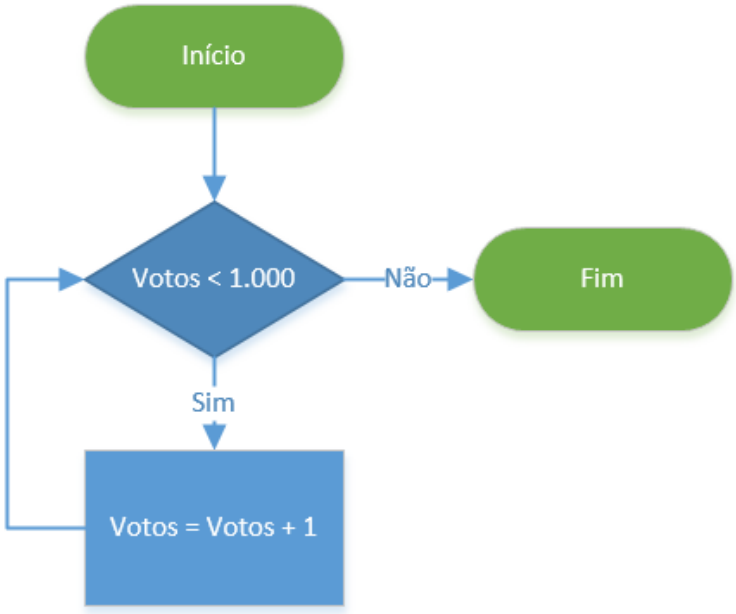
Agora veremos as famosas Estruturas de Repetição! Essas estruturas são utilizadas quando se deseja que um determinado conjunto de instruções ou comandos sejam executados por mais de uma vez. A quantidade pode ser definida, indefinida, ou enquanto um estado se mantenha ou seja alcançado. Parece confuso,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

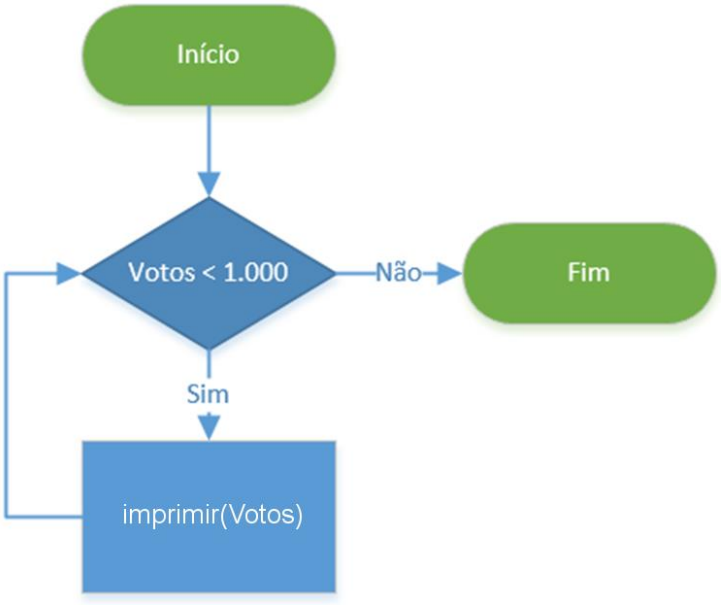
```
Votos = 500

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:

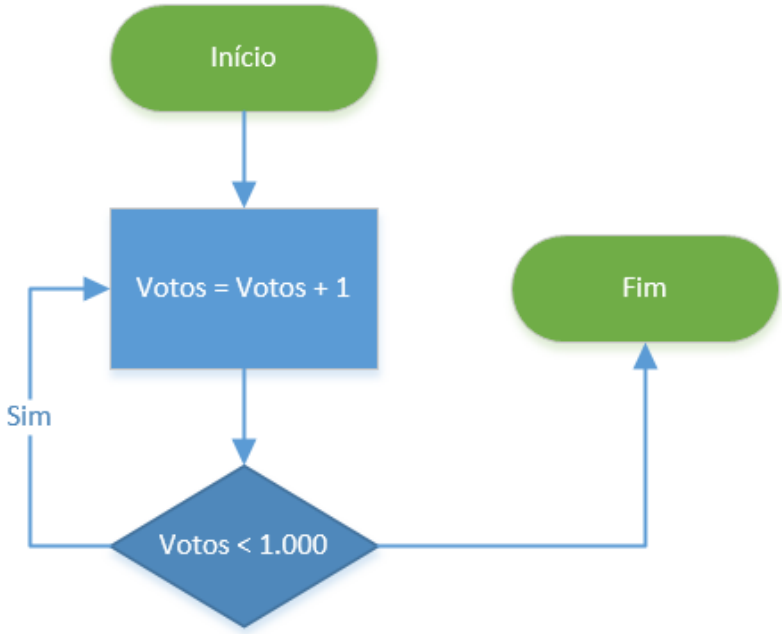
- Testa se Votos é igual a 1000;
- Se é igual, encerra a repetição;
- Se é menor, soma 1 à variável de controle "Votos";
- Mais uma repetição

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?*

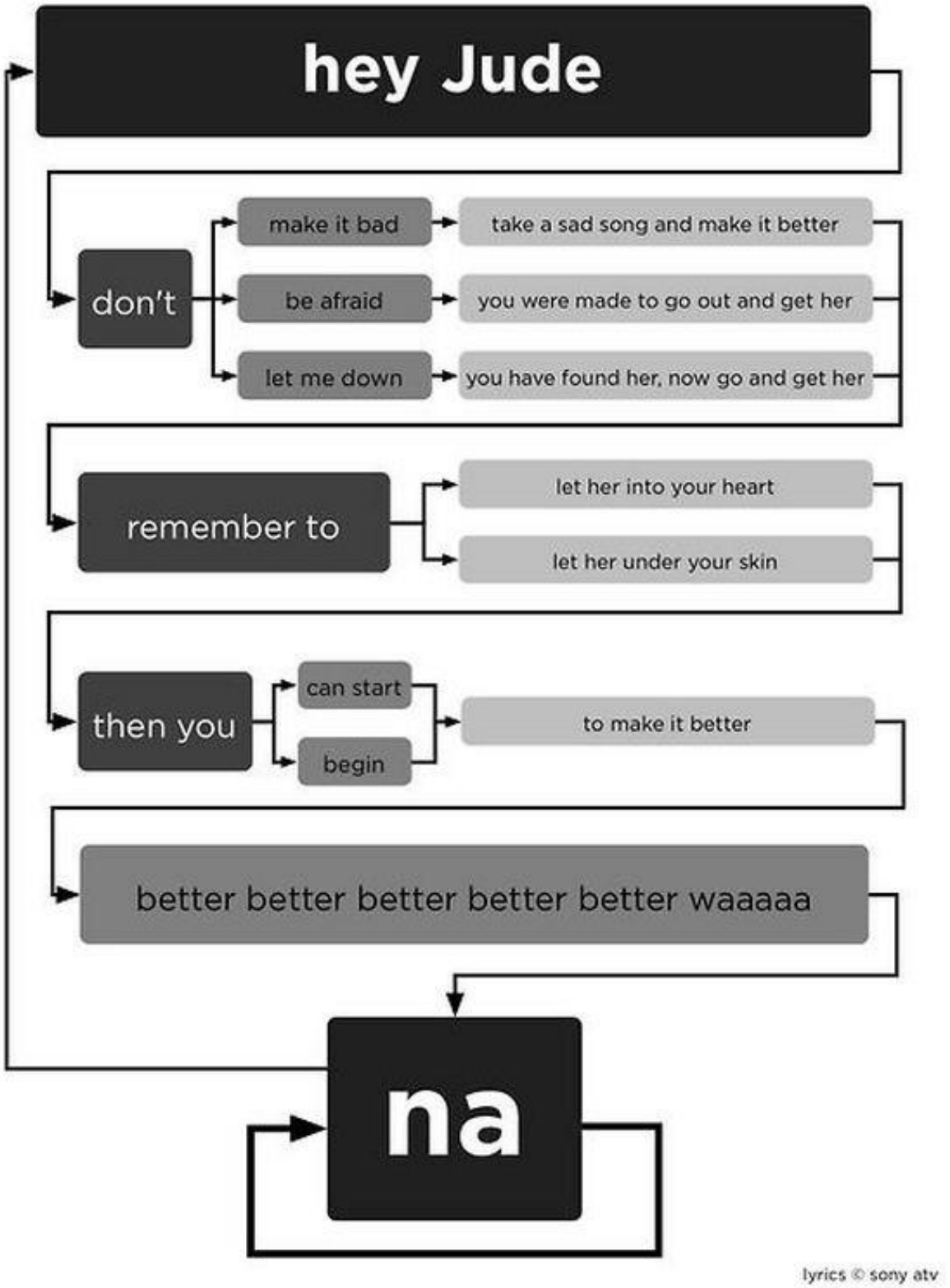
```
01 estrofe = 0
02
03 Enquanto (estrofe < 3) Repetir
04   estrofe == estrofe + 1
05   imprimir("Hey Jude, don't ")
06
07   Se (estrofe == 1) Então
08     imprimir("make it bad")
09     imprimir("take a sad song and make it better")
10   Senão Se (estrofe == 2) Então
11     imprimir("be afraid")
12     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.

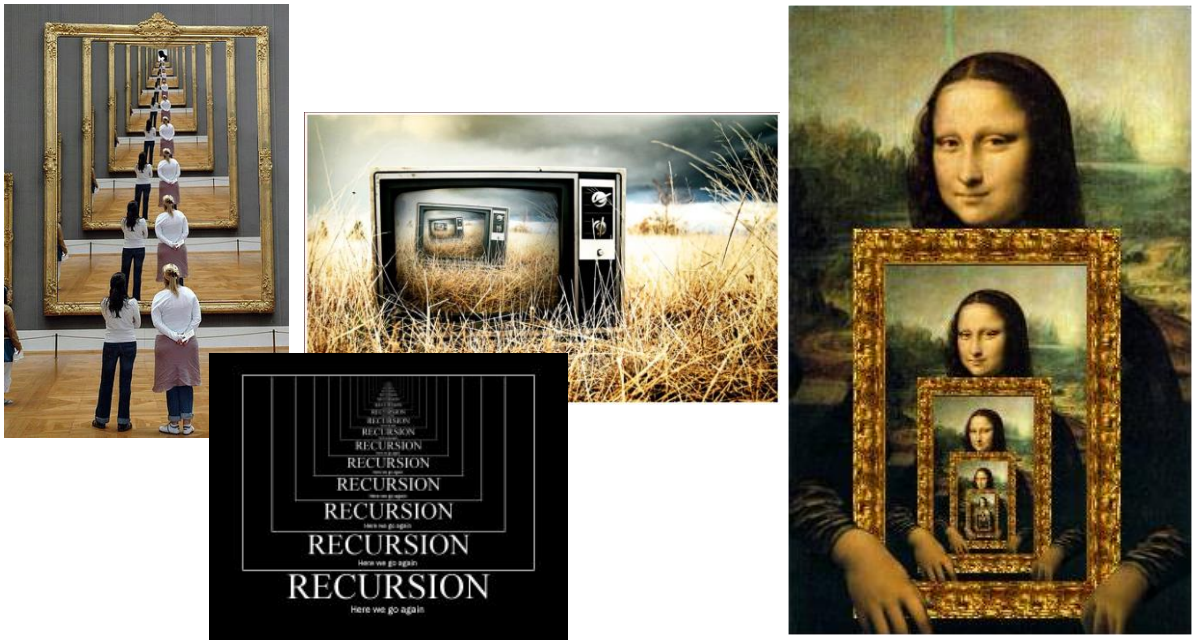
Após o primeiro "Se Então Senão", temos outro, que testa se estrofe é par ou ímpar (linha 28). Essa estrutura também é composta, pois possui um bloco "Senão". Por fim, temos uma estrutura de seleção (decisão) simples (linha 35), ou seja, somente com bloco "Se Então" e, dentro desse, encontramos uma estrutura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


hey Jude
don't
make it bad
take a sad song and make it better
be afraid
you were made to go out and get her
let me down
you have found her, now go and get her
remember to
let her into your heart
let her under your skin
then you
can start
begin
to make it better
better better better better better waaaaa
na
lyrics © sony atv

RECURSIVIDADE

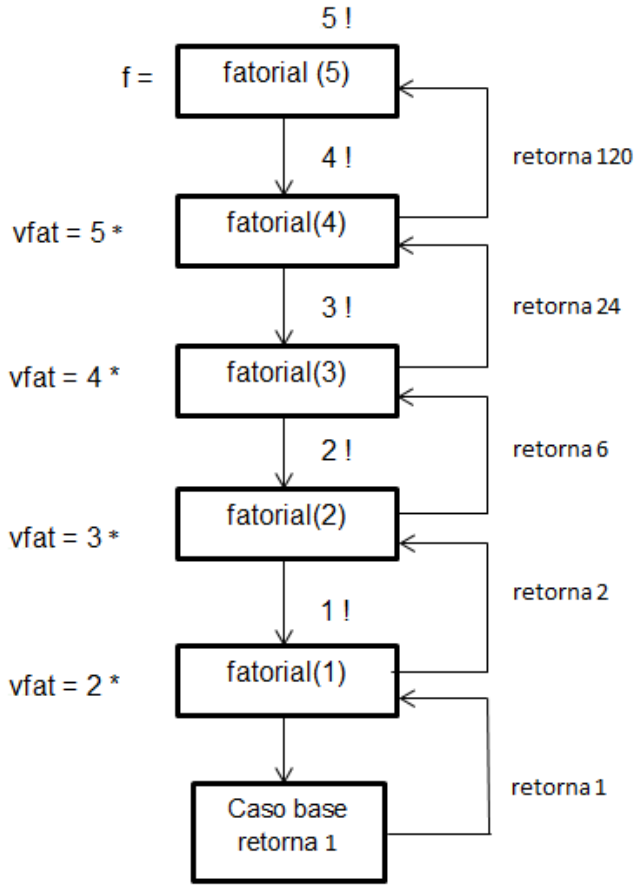
Por fim, vamos falar sobre recursividade! *O que é isso, professor?* Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por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

QUESTÕES

1. (CESGRANRIO – 2014 – EPE – Tecnologia da Informação) Analise o algoritmo abaixo, onde a%b representa o resto da divisão de a por b.

```
inicio
    inteiro x,y,i,r
    ler x
    ler y
    para i de 1 até x
      se (x%i=0) e (y%i=0) então
          r <- i
      fim se
    próximo
    escrever r
fim
```

Qual será a resposta, caso as entradas sejam 128, para x, e 56, para y?

a) 2
b) 8
c) 56
d) 64
e) 128

Comentários:

Galera, vejam como não é difícil. Nós, primeiro temos um comando **início** para indicar o início do algoritmo. Em seguida ele declara 4 variáveis (espaços na memória) para armazenar um dado do tipo **inteiro** e dá o nome de **x, y, i, r.** Depois nós temos o comando **ler** em que ele vai ler uma entrada do usuário e armazenar em **x** e outro comando **ler** em que ele vai ler outra entrada do usuário e armazenar em **y**.

Que valores são esses, professor? Ele diz no enunciado: 128 para x e 56 para y. Dito isso, o próximo comando tem uma estrutura de repetição **para**. E ela diz que para **i = 1 até i = x** (lembrem-se que x = 128), repita o que tem abaixo – lembrando que no final do comando **para** tem um comando **próximo**, a cada repetição, a variável de controle i é incrementada em 1. Relaxa que é simples de entender: a cada para eu aumento o valor de i em 1. Aí, abaixo nós temos outro comando que diz: **se x%i = 0 e y%i = 0**, então r ← i. E ele disse no enunciado que **a%b** representa o resto da divisão de a por b, logo x%i representa o resto da divisão de x por i. *Então, vamos testar?*

```
x = 128
y = 56
```

x%i = 128%1. Qual o resto da divisão de 128 por 1? Quociente 128, Resto 0.
x%i = 56%1. Qual o resto da divisão de 56 por 1? Quociente 56, Resto 0.

Então cumprimos as duas condições do comando **se x%i = 0 e y%i = 0**, logo executamos o comando abaixo: **r ← i**, ou seja, r agora será i, então r = 1.

--

Vejam agora o comando próximo. Isso significa que agora i = 2 e nós repetimos tudo de novo.

x%i = 128%2. Qual o resto da divisão de 128 por 2? Quociente 64, Resto 0.
x%i = 56%2. Qual o resto da divisão de 56 por 2? Quociente 28, Resto 0.

Então cumprimos as duas condições do comando **se x%i = 0 e y%i = 0**, logo executamos o comando abaixo: **r ← i**, ou seja, r agora será i, então r = 2.

--

Vejam agora o comando próximo. Isso significa que agora i = 3 e nós repetimos tudo de novo.

x%i = 128%3. Qual o resto da divisão de 128 por 3? Quociente 42, Resto 2.
x%i = 56%3. Qual o resto da divisão de 56 por 3? Quociente 18, Resto 2.

Então não cumprimos as duas condições do comando `se x%i = 0 e y%i = 0`, o resto não é zero para nenhum dos dois casos, logo não executamos o comando abaixo: `r ← i`, portanto r continua sendo igual a 2, então r = 2.

--

Vejam agora o comando próximo. Isso significa que agora i = 4 e nós repetimos tudo de novo.

`x%i = 128%4.` Qual o resto da divisão de 128 por 4? Quociente 32, Resto 0.
`x%i = 56%4.` Qual o resto da divisão de 56 por 4? Quociente 14, Resto 0.

Então cumprimos as duas condições do comando `se x%i = 0 e y%i = 0`, logo executamos o comando abaixo: `r ← i`, ou seja, r agora será i, então r = 4.

Pessoa, nós podemos continuar fazendo na mão até descobri o valor final de r, mas vocês perceberam que esse algoritmo está querendo descobrir o Máximo Divisor Comum (MDC) entre 128 e 56? Ele sempre está decompondo os dois números e fará isso até descobrir qual será o MDC entre eles. Uma vez descoberto isso, eu não preciso fazer na mão. Sabemos 128 é divisível por 1, 2, 4, 8, 16, 32 e 64. Sabemos que 56 é divisível por 1, 2, 4, 7 e 8. Vimos que ambos têm em comum os divisores 1, 2, 4 e 8. Então o MÁXIMO divisor comum é o 8.

Olha que legal! Se você coloca isso em uma linguagem de programação e manda o computador executar, ele sempre vai descobrir o MDC entre quaisquer dois números. E é isso que o computador faz, ele responde a um algoritmo criado por um programador.

Gabarito: B

---

2. (CESGRANRIO – 2014 – PETROBRÁS - Analista de Sistemas) Analise o algoritmo abaixo em português estruturado.

```
algoritmo segredo;
variáveis
  x,y,z : inteiro;
fim-variáveis
início
   x:=15;
   y:=10;
   z:=0;
   enquanto y>0 faça
      z:=z+x;
      y:=y-1;
   fim-enquanto
   imprima(z);
fim
```

Que número seria impresso caso esse programa executasse?

a) 0
b) 10
c) 15
d) 100
e) 150

Comentários:

Vamos lá, galera... vimos que o nome do algoritmo é segredo e que ele tem três variáveis: x, y, z do tipo inteiro. No início, ele diz que: x = 15; y = 10 e z = 0. E aí vamos para a estrutura de repetição **enquanto**:

Sabemos que y = 10, ele diz: enquanto y > 0 repita o que está abaixo. E abaixo está:

z = z + x. Logo, z = 0 + 15 = 15. Agora z = 15.
y = y – 1. Logo, y = 10 – 1 = 9. Agora y = 9.

E terminamos o **enquanto**, mas lembrem-se que é uma estrutura de repetição. Então, nós só saímos dela quando y > 0 (não é o caso, porque y = 9). Então, de novo:

z = z + x. Logo, z = 15 + 15 = 30. Agora z = 30.
y = y – 1. Logo, y = 9 – 1 = 8. Agora y = 8.

E terminamos o **enquanto**, mas lembrem-se que é uma estrutura de repetição. Então, nós só saímos dela quando y > 0 (não é o caso, porque y = 8). Então, de novo:

z = z + x. Logo, z = 30 + 15 = 45. Agora z = 45.
y = y – 1. Logo, y = 8 – 1 = 7. Agora y = 7.

Galera, vocês percebem um padrão? Esse algoritmo vai contando de 15 em 15 até que y > 0 não seja mais verdadeira. Em outras palavras, ele somará o valor 15, 10 vezes seguida, ou seja, vai fazer 10x15 = 150. Eu posso continuar fazendo na mão, mas como eu já descobri o padrão, eu já descobri a resposta :)

Gabarito: E

---

3. (CESGRANRIO – 2014 – BASA - Analista de Sistemas) A saída do algoritmo apresentado abaixo para as entradas 100 e 20, respectivamente, é

```
inicio
inteiro X, Y
Ler X
Ler Y
Enquanto X ≥ Y - 1 faz
X < - X – 1
Y < - Y + 2 Fim Enquanto
Escrever "saída =" , Y – X
Fim
```

a) -5
b) -2
c) 1
d) 4
e) 7

Comentários:

Vamos lá! O algoritmo declara duas variáveis do tipo inteiro X e Y. Manda ler X e Y do usuário (o enunciado diz que foram lidos X = 100 e Y = 20). E vamos para a estrutura de repetição **enquanto**:

Enquanto X >= Y-1 faça, ou seja, enquanto 100 >= 19 faça.

X ← X – 1, ou seja, X ← 99;
Y ← Y + 2, ou seja, Y ← 22;

--

Enquanto X >= Y-1 faça, ou seja, enquanto 99 >= 21 faça.

X ← X – 1, ou seja, X ← 98;
Y ← Y + 2, ou seja, Y ← 24;

--

Enquanto X >= Y-1 faça, ou seja, enquanto 98 >= 21 faça.

X ← X – 1, ou seja, X ← 97;
Y ← Y + 2, ou seja, Y ← 26;

Já descobriram o padrão? X vai sendo decrementado de 1 em 1; e Y vai sendo incrementado de 2 em 2. Quando X >= Y-1 não for mais verdadeiro, sai do enquanto e escreve a saída. Temos então as seguintes combinações:

X = 100, Y = 20;
X = 99, Y = 22;
X = 98, Y = 24;
X = 97, Y = 26;
X = 96, Y = 28;
X = 95, Y = 30;
X = 94, Y = 32;
X = 93, Y = 34;
X = 92, Y = 36;
X = 91, Y = 38;
X = 90, Y = 40;
X = 89, Y = 42;
X = 88, Y = 44;
X = 87, Y = 46;
X = 86, Y = 48;
X = 85, Y = 50;
X = 84, Y = 52;
X = 83, Y = 54;
X = 82, Y = 56;
X = 81, Y = 58;
X = 80, Y = 60;
X = 79, Y = 62;
X = 78, Y = 64;

X = 77, Y = 66;
X = 76, Y = 68;
X = 75, Y = 70;
X = 74, Y = 72;
X = 73, Y = 74;
X = 72, Y = 76;

Observem que nesse ponto, X >= Y - 1 não é mais verdadeiro. *Por que?* Porque X = 72 e Y = 76, então 72 >= 76-1 é falso porque 72 é MENOR que 75. Então nós podemos sair da estrutura de repetição **enquanto** e escrever a saída, que ele pede que seja Y-X. Y = 76 e X = 72, logo Y-X = 4.

Gabarito: D

---

4. (CESGRANRIO – 2010 – PETROBRÁS – Técnico em Informático) Relacionado à programação de computadores, um algoritmo, seja qual for a sua complexidade e a linguagem de programação na qual será codificado, pode ser descrito por meio da:

a) reografia.
b) criptografia.
c) linguagem de marcação.
d) engenharia estruturada.
e) pseudolinguagem.

Comentários:

> *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.*

Essa é tranquilinha! Trata-se da pseudolinguagem.

Gabarito: E

---

5. (FCC - 2010 - DPE-SP - Agente de Defensoria - Analista de Sistemas) É utilizada para avaliar uma determinada expressão e definir se um bloco de código deve ou não ser executado. Essa é a definição da estrutura condicional:

a) For
b) If...Then...Else
c) While
d) Do...While
e) Next

Comentários:

Pessoal... falou em estrutura *condicional*, trata-se de decisão! Logo, é o If-Then-Else.

Gabarito: B

---

6. (FCC - 2010 – TRT/SE - Analista de Sistemas) Objeto que se constitui parcialmente ou é definido em termos de si próprio. Nesse contexto, um tipo especial de procedimento (algoritmo) será utilizado, algumas vezes, para a solução de alguns problemas. Esse procedimento é denominado:

a) Recursividade.
b) Rotatividade.
c) Repetição.
d) Interligação.
e) Condicionalidade.

Comentários:

> *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.*

Conforme vimos em aula, trata-se da recursividade.

Gabarito: A

---

7. (FCC - 2009 – TJ/SE - Analista de Sistemas) A recursividade na programação de computadores envolve a definição de uma função que:

a) apresenta outra função como resultado.

b) aponta para um objeto.
c) aponta para uma variável.
d) chama uma outra função.
e) pode chamar a si mesma.

Comentários:

> *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.*

Conforme vimos em aula, ela pode chamar a si mesma.

Gabarito: E

---

8. (CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos) Uma estrutura de repetição possibilita executar um bloco de comando, repetidas vezes, até que seja encontrada uma dada condição que conclua a repetição.

Comentários:

Essa é uma definição perfeita da Estrutura de Repetição.

Gabarito: Certo

---

9. (CESPE - 2010 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Se um trecho de algoritmo tiver de ser executado repetidamente e o número de repetições for indefinido, então é correto o uso, no início desse trecho, da estrutura de repetição Enquanto.

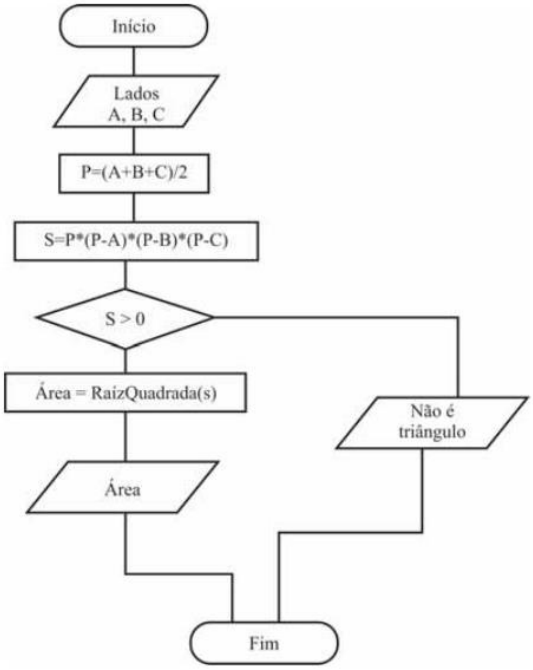
Comentários:

Perfeito! *Lembram-se que nós temos três estruturas de repetição?* Pois é, o Enquanto-Faça serve para esse propósito!

Gabarito: Certo

---

10. (CESPE - 2013 - CNJ - Programador de computador) No fluxograma abaixo, se A = 4, B = 4 e C = 8, o resultado que será computado para Área é igual a 32.



Comentários:

Vamos lá:

P = (4+4+8)/2

Lembrem-se que parênteses sempre têm prioridade:

P = (16)/2 = 8

Seguimos para S = P*(P-A)*(P-B)*(P-C):

S = 8(8-4)*(8-4)*(8-8)*

Simplificando:

*S = 8*4*4*0 = 0*

S <= 0, logo: Não é Triângulo!

Gabarito: Errado

---

11. (CESPE - 2011 - TJ-ES - Analista Judiciário - Análise de Banco de Dados - Específicos) Em uma estrutura de repetição com variável de controle, ou estrutura PARA, a verificação da condição é realizada antes da execução do corpo da sentença, o que impede a reescrita desse tipo de estrutura por meio de estrutura de repetição pós-testada.

Comentários:

Nós temos três tipos de estruturas de repetição com variável de controle: While/Enquanto, For/Para e Do-While/Faça-Enquanto - as duas primeiras pré-testadas e a última pós-testada. De fato, a estrutura For/Para é pré-testada. No entanto, é possível reescrevê-la como uma Estrutura de Repetição Pós-Testada (Do-While/Faça-Enquanto), de forma que elas sejam equivalentes.

Gabarito: Errado

---

12. (CESPE - 2010 – DETRAN/ES - Analista de Sistemas) O método de recursividade deve ser utilizado para avaliar uma expressão aritmética na qual um procedimento pode chamar a si mesmo, ou seja, a recursividade consiste em um método que, para que possa ser aplicado a uma estrutura, aplica a si mesmo para as subestruturas componentes.

Comentários:

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.

Conforme vimos em aula, a questão está correta. No entanto, sendo rigoroso, ele bem que podia ter usado um verbo diferente para evitar confusão (*"O método de recursividade pode ser utilizado..."* ou *"O método de recursividade é utilizado..."*).

Gabarito: Certo

---

13. (CESPE - 2013 – CPRM - Analista de Sistemas) Na implementação de recursividade, uma das soluções para que se evite o fenômeno de terminação do programa – que possibilita a ocorrência de um looping infinito – é definir uma função ou condição de terminação das repetições.

Comentários:

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.

Conforme vimos em aula, está perfeito novamente.

Gabarito: Certo

---

14. (CESPE - 2014 – ANATEL - Analista de Sistemas) A recursividade é uma técnica que pode ser utilizada na implementação de sistemas de lógica complexa, com a finalidade de minimizar riscos de ocorrência de defeitos no software.

Comentários:

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.

Conforme vimos em aula, definitivamente essa não é uma finalidade da recursividade. Uma implementação simples não significa de maneira alguma mais eficiente. Em geral, um algoritmo iterativo (não-recursivo) é mais eficiente que um algoritmo recursivo (por conta da manutenção de estado, pilhas, etc). Portanto, não há essa correlação entre riscos de defeito e complexidade de implementação.

Gabarito: Errado

---

15. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Tanto a recursividade direta quanto a indireta necessitam de uma condição de saída ou de encerramento.

Comentários:

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).

Conforme vimos em aula, existe recursividade direta e indireta, mas ambas precisam de um caso-base, que é uma condição de saída, para não ficar em loop infinito.

Gabarito: Certo

---

16. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo.

```
Atribuir 13 a X;
Repetir
      Atribuir X – 2 a X;
      Imprimir (X);
Até que X < -1;
```

A estrutura será executada até que X seja igual ao seguinte valor:

a) – 1
b) – 3

Comentários:

Vamos lá: *X = 13*
No laço, ele afirma: X = X-2
X = 13-2 = 11
Segue esse loop novamente:
X = 11-2 = 9
X = 9-2 = 7
X = 7-2 = 5
X = 5-2 = 3
X = 3-2 = 1
X = 1-2 = -1
X = -1-2 = -3 < -1, logo saímos do loop!

Portanto, a estrutura é executada até X = -3.

Gabarito: B

---

17. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo, que mostra o emprego da estrutura de controle enquanto ... faça ...

```
atribuir 0 a n;
enquanto n < 7 faça
   início
      imprimir (n);
      atribuir n+1 a n;
```

```
    fim;
```

A opção que utiliza a estrutura para ... faça ... correspondente, que gera o mesmo resultado, é:

a) Para n de 0 até 6 faça imprimir(n);
b) Para n de 0 até 7 faça imprimir(n);

Comentários:

Vejamos: N = 0 e irá até N < 7, logo N de 0 a 6.

Gabarito: A

---

18. (FEPESE - 2010 - SEFAZ-SC - Auditor Fiscal da Receita Estadual - Parte III - Tecnologia da Informação) Assinale a alternativa correta a respeito das variáveis e constantes, utilizadas em diversas linguagens de programação.

a) O número de constantes deve ser menor ou igual ao número de variáveis em um programa.

b) O número de constantes deve ser menor ou igual ao número de procedimentos em um programa.

c) O número de constantes deve ser igual ao número de variáveis em um programa.

d) O número de constantes independe da quantidade de variáveis em um programa.

e) O número de constantes deve ser igual ao número de procedimentos em um programa.

Comentários:

Galera, não há essa relação! O número de constantes e variáveis são independentes.

Gabarito: D

---

19. (NUCEPE - 2015 – SEDUC/PI - Analista de Sistemas) O código abaixo é usado para calcular o fatorial de números. Assinale a alternativa CORRETA sobre esse código:

```
função fatorial(n)
{
  se (n <= 1)
    retorne 1;
  senão
    retorne n * fatorial(n-1);
}
```

a) Este é um exemplo de procedimento.

b) O comando retorne pode ser retirado do código e a função terá o mesmo efeito.

c) Exemplo clássico de recursividade.

d) Não é possível chamar a função fatorial dentro dela mesma.

e) O resultado da função sempre retornará um valor elevado a ele mesmo (valor ^ valor).

Comentários:

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.

Conforme vimos em aula, trata-se de um exemplo clássico de recursividade.

Gabarito: C

---

20. (IADES - 2011 – PG/DF - Analista Jurídico - Analista de Sistemas) Os algoritmos são compostos por estruturas de controle de três tipos: sequencial, condicional e de repetição. Assinale a alternativa que apresenta apenas um tipo de estrutura de controle:

a) ...

```
escreva (“Digite seu nome: ”)
leia (nome)
escreva (“Digite sua idade: ”)
leia (idade)
limpe a tela
escreva (“Seu nome é:”, nome)
escreva (“Sua idade é:”, idade)
se (nome = “João”) entao
  se (idade > 18) entao
     escreva (nome, ” é maior de 18 anos!”)
  fim se
fim se
...
```

b) ...

```
escreva (“Pressione qualquer tecla para começar...”)
leia (tecla)
mensagem ← “Não devo acordar tarde...”
numero ← 0
enquanto (numero < 100)
     escreva (mensagem)
     numero ← (numero + 1)
fim enquanto
escreva (“Pressione qualquer tecla para
terminar...”)
leia (tecla)
escreva (“Tecla digitada: ”)
escreva (tecla)
...
```

c) ...

```
leia (nome)
escreva (“nome digitado: ”)
escreva (nome)
se (nome = “Wally”) entao
    escreva (“Encontrado o Wally!”)
senao
    cont ← 5
    enquanto (cont > 0)
```

```
            escreva (“Não é Wally”...”)
            cont ← (cont – 1)
          fim enquanto
      fim se
      ...
```

d) ...

```
      var
        nome: literal
        num: inteiro
      inicio
        escreva (“Digite seu nome: ”)
        leia (nome)
        num ← 0
        se (nome = “José”) entao
            num ← (num + 1)
        fim se
       escreva (“Quantidade de João encontrados:
      ”)
       escreva (num)
      ...
```

e) ...

```
      var
         nome: literal
         idade: inteiro
      inicio
        escreva (“Digite seu nome: ”)
        leia (nome)
        escreva (“Digite sua idade: ”)
        leia (idade)
        limpe a tela
        escreva (“Seu nome é:”)
        escreva (nome)
        escreva (“Sua idade é:”)
        escreva (idade)
      fim algoritmo
      ...
```

Comentários:

(a) possui instruções sequenciais e também uma estrutura de decisão (uma delas aninhada), ou seja, temos dois tipos; (b) possui instruções sequenciais e também uma estrutura de repetição; (c) possui instruções sequenciais, uma estrutura de decisão e uma de repetição; (d) possui instruções sequenciais e também uma estrutura de decisão; (e) não possui estruturas de decisão (seleção), nem de repetição, somente sequenciais.

Gabarito: E

---

21. (IADES - 2011 – TRE-PA - Programador de Computador)

```
VAR
N1, N2 : INTEIRO;
N1 ← 2;
N2 ← 30;
INICIO
ENQUANTO N1<N2 FAÇA
      N2 ← N2 + N1;
      N1 ← N1 * 3;
FIM ENQUANTO;
N1 ← N2 + 11;
FIM
```

Dado o algoritmo escrito em pseudocódigo, quais os valores de N1 e N2, respectivamente, ao final da execução?

a) 162 e 110.
b) 110 e 121.
c) 110 e 162.
d) 121 e 110.
e) 173 e 110.

Comentários:

A questão traz um exemplo de estrutura de repetição pré-testada e quer saber se vocês entenderam bem a aula teórica! ☺ *Vamos executar passo a passo para realizar os testes e mostrar como funciona cada iteração (repetição) do bloco da estrutura de repetição?*

1ª iteração:

N1 é igual a 2 (atribuição da 3ª linha)

N2 é igual a 30 (atribuição da 4ª linha)
Teste: N1 < N2?? Sim, pois 2 é menor que 30!! Então entramos no loop para executar as instruções do bloco da estrutura
N2 recebe 30 + 2
N1 recebe 2 * 3

2ª iteração:

N1 é igual a 6
N2 é igual a 32
Teste: N1 < N2?? Sim, pois 6 é menor que 32!! Então entramos no loop para executar as instruções do bloco da estrutura
N2 recebe 32 + 6
N1 recebe 6 * 3

3ª iteração:

N1 é igual a 18
N2 é igual a 38
Teste: N1 < N2?? Sim, pois 18 é menor que 38!! Então entramos no loop para executar as instruções do bloco da estrutura
N2 recebe 38 + 18
N1 recebe 18 * 3

4ª iteração:

N1 é igual a 54
N2 é igual a 56
Teste: N1 < N2?? Sim, pois 54 é menor que 56!! Então entramos no loop para executar as instruções do bloco da estrutura
N2 recebe 56 + 54
N1 recebe 54 * 3

5ª iteração:

N1 é igual a 162
N2 é igual a 110

Teste: N1 < N2? Não (finalmente :D), visto que 162 não é menor que 110!! Desse modo, saímos do loop e voltamos para a instrução imediatamente posterior, ou seja: N1 recebe 110 + 11. Fim do Código. Ao final, N1 tem o valor de 121 e N2 de 110.

Gabarito: D

---

22. (CESPE – 2017 – TRE/BA – Analista Judiciário – Analista de Sistemas) Assinale a opção que apresenta a saída resultante da execução do algoritmo antecedente.

```
var a = 0, b = 1, f = 1;
      for (var i = 2; i <= 6; i++) {
            f = a + b;
            a = b;
            b = f;
      document.write(b);
      }
```

a) 123456
b) 12121
c) 12345
d) 0112
e) 12358

Comentários:

Antes do loop, nossas variáveis valem:

```
a = 0; b = 1; f = 1;
```

Após entrar no loop, não sai enquanto i <= 6. Então, temos que:

```
Quando i = 2, temos:
Primeira instrução:     f = a + b; f = 0 + b; f = 0 + 1; f = 1;
Segunda instrução:      a = b; a = 1;
Terceira instrução:     b = f; b = 1;
Quarta instrução:       Saída = 1;
Logo, temos que:        i = 2; f = 1; a = 1; b = 1;

Quando i = 3, temos:
Primeira instrução:     f = a + b; f = 1 + b; f = 1 + 1; f = 2;
Segunda instrução:      a = b; a = 1;
Terceira instrução:     b = f; b = 2;
Quarta instrução:       Saída = 12;
Logo, temos que:        i = 3; f = 2; a = 1; b = 2;
```

```
Quando i = 4, temos:
Primeira instrução:     f = a + b; f = 1 + b; f = 1 + 2; f = 3;
Segunda instrução:      a = b; a = 2;
Terceira instrução:     b = f; b = 3;
Quarta instrução:       Saída = 123;
Logo, temos que:        i = 4; f = 3; a = 2; b = 3;

Quando i = 5, temos:
Primeira instrução:     f = a + b; f = 2 + b; f = 2 + 3; f = 5;
Segunda instrução:      a = b; a = 3;
Terceira instrução:     b = f; b = 5;
Quarta instrução:       Saída = 1235;
Logo, temos que:        i = 5; f = 5; a = 3; b = 5;

Quando i = 6, temos:
Primeira instrução:     f = a + b; f = 3 + b; f = 2 + 5; f = 8;
Segunda instrução:      a = b; a = 3;
Terceira instrução:     b = f; b = 8;
Quarta instrução:       Saída = 12358;
Logo, temos que:        i = 6; f = 8; a = 3; b = 8;
```

A partir daí, não entra mais no loop porque a variável de controle será maior que seis. Beleza? Então, a saída será: 12358.

Gabarito: Errado

---

23. (CONSULPLAN - 2012 - TSE – Técnico – Programação de Sistemas) Analise o pseudocódigo, que ilustra o uso de uma função recursiva.

```
programa PPRRGG;
variáveis
   VERDE, AZUL : numérica;
função FF(AUX:numérica):numérica;
início
   atribuir VERDE+1 a VERDE;
   se AUX <= 2
   então atribuir 5 a FF
   senão atribuir AUX*FF(AUX-1) a FF;
fim; { fim da função FF }
início
  atribuir 0 a VERDE;
  atribuir FF(4) a AZUL;
  escrever(VERDE,AZUL);
fim.
```

O valor de retorno de FF e a quantidade de vezes que a função será executada serão, respectivamente,

a) 5 e 1.
b) 15 e 2.
c) 60 e 3.
d) 300 e 4.

Comentários:

Galera, como vimos, as sub-rotinas que chamam a si mesmas são chamadas de recursivas. Na questão temos a função FF, que na última linha faz uma chamada a ela mesma. Vamos fazer o passo a passo para entender como funciona a função, lembrando que a recursividade requer um critério de parada a partir de algum teste de um valor da variável usada como controle.

Segundo o início do código temos:

VERDE tem o valor 0
AZUL tem o valor retornado por FF(4)

1ª Chamada da função FF

AUX tem o valor 4 (passagem de parâmetro na chamada do programa principal)
VERDE RECEBE VERDE + 1, ou seja, VERDE RECEBE 1
AUX <= 2? Ou ainda, 4 <= 2? Não! Bloco senão é executado
FF retorna AUX * FF(AUX-1), ou seja, FF retorna 4 * FF(3)

2ª Chamada da função FF

AUX tem o valor 3 (passagem de parâmetro na chamada recursiva)
VERDE RECEBE VERDE + 1, ou seja, VERDE RECEBE 2

/ Opa, opa, professor, porque VERDE continua com o valor que recebeu na 1a chamada de FF? Ora, meu caro padawan, porque as variáveis VERDE e AZUL foram declaradas fora da função num escopo mais geral, ou seja, as alterações dentro da função valem fora dela e para todas as suas chamadas recursivas. Agora de volta ao código :) */*

AUX <= 2? Ou ainda, 3 <= 2? Não! Bloco senão é executado
FF retorna AUX * FF(AUX-1), ou seja, FF retorna 3 * FF(2)

3ª Chamada da função FF

AUX tem o valor 2 (passagem de parâmetro na chamada recursiva)

VERDE RECEBE VERDE + 1, ou seja, VERDE RECEBE 3
AUX <= 2? Ou ainda, 3 <= 3? Sim! Bloco então é executado
FF retorna 5

/ Agora é hora de voltar nas chamadas recursivas, do último para o primeiro
Isso lembra alguma coisa?? LIFO?? Pilhas! São utilizadas nas chamadas
recursivas para voltar a execução na hierarquia das chamadas */*

Voltando à 2ª Chamada de FF

FF retorna 3 * 5, ou seja, FF retorna 15

Voltando à 1ª chamada de FF

FF retorna 4 * FF(3), ou seja, FF retorna 4 * 15, FF retorna 60. Em outras palavras, o valor de FF(4) é 60 e a função FF é chamada 3 vezes.

Gabarito: C

---

24. (CESPE – 2017 – TRE/BA - Analista de Sistemas)

```
var i = 0;
 while (i < 5) {
   i++;
   if (i == 3) {
         continue;
   }
 document.write(i);
 }
```

Assinale a opção que apresenta a saída resultante da execução do algoritmo antecedente.

a) 12345
b) 1245
c) 3
d) 0124
e) 1234

Comentários:

Antes do loop, temos que: `i = 0;`

Após entrar no loop, não sai enquanto i < 5. Então, temos que:

```
Quando i = 0, temos:
Primeira instrução:     i++ é o mesmo que i = i + 1, logo i = 0 + 1 = 1;
Segunda instrução:      i == 3? Não, i == 1. Então, não entra no if;
Terceira instrução:     Saída = 1;

Quando i = 1, temos:
Primeira instrução:     i++ é o mesmo que i = i + 1, logo i = 1 + 1 = 2;
Segunda instrução:      i == 3? Não, i == 2. Então, não entra no if;
Terceira instrução:     Saída = 12;

Quando i = 2, temos:
Primeira instrução:     i++ é o mesmo que i = i + 1, logo i = 2 + 1 = 3;
Segunda instrução:      i == 3? Sim, o continue faz iterar o loop do início;

Quando i = 3, temos:
Primeira instrução:     i++ é o mesmo que i = i + 1, logo i = 3 + 1 = 4;
Segunda instrução:      i == 3? Não, i == 4. Então, não entra no if;
Terceira instrução:     Saída = 124;

Quando i = 4, temos:
Primeira instrução:     i++ é o mesmo que i = i + 1, logo i = 4 + 1 = 5;
Segunda instrução:      i == 3? Não, i == 5. Então, não entra no if;
Terceira instrução:     Saída = 1245;

Quando i = 5, temos:    Não entra mais no loop.
```

A partir daí, não entra mais no loop porque a variável de controle não será menor que cinco. *Capiche?* Então, a saída será: 1245.

Gabarito: B

---

25. (VUNESP – 2014 – SP/URBANISMO – Analista Administrativo) Analise o algoritmo a seguir, apresentado na forma de uma pseudolinguagem (Português Estruturado). Esse algoritmo deverá ser utilizado para responder às questões.

```
Início
  Inteiro x1, x2, x3, i;
  Leia x1, x2, x3;
  Para i de 1 até x1 faça
  [
    x2 ← x1 + x3;
    x3 ← x1 - x2;
  ]
Imprima (x2 + x3);
Fim
```

Considere que os valores lidos para x1, x2 e x3 tenham sido, respectivamente, 5, 4 e 3. É correto afirmar que o valor impresso ao final da execução do algoritmo é igual a:

a) –3
b) 0
c) 5
d) 8
e) 11

Comentários:

Sabendo que x1 recebe 5, x2 recebe 4 e x3 recebe 3. Executa-se o loop do algoritmo da forma a seguir.

Quando i = 1, temos que:
x2 ← x1 + x3 = 5 + 3 = 8 (x1, x2, x3 = 5, 8, 3)
x3 ← x1 - x2 = 5 - 8 = -3 (x1, x2, x3 = 5, 8, -3)

Quando i = 2, temos que:
x2 ← x1 + x3 = 5 - 3 = 2 (x1, x2, x3 = 5, 2, -3)
x3 ← x1 - x2 = 5 - 2 = 3 (x1, x2, x3 = 5, 2, 3)

Quando i = 3, temos que:
x2 ← x1 + x3 = 5 + 3 = 8 (x1, x2, x3 = 5, 8, 3)
x3 ← x1 - x2 = 5 - 8 = -3 (x1, x2, x3 = 5, 8, -3)

Quando i = 4, temos que:
x2 ← x1 + x3 = 5 - 3 = 2 (x1, x2, x3 = 5, 2, 3)
x3 ← x1 - x2 = 5 - 2 = 3 (x1, x2, x3 = 5, 2, 3)

Quando i = 5, temos que:
x2 ← x1 + x3 = 5 + 3 = 8 (x1, x2, x3 = 5, 8, 3)

x3 ← x1 - x2 = 5 - 8 = -3 (x1, x2, x3 = 5, 8, -3)

Logo, x2 + x3 = 8 - 3 = 5.

Gabarito: C

---

26. (CESPE – 2017 – TRT 7ª Região – CE – Técnico Judiciário – Tecnologia da Informação)

```
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

Comentários:

Para achar o valor atribuído de H, precisamos descobrir os valores de A, B, C e D. Começando da ordem das linhas da 10 até a 13, temos que:

A ← 5;
B ← A x (-2) = 5 x (-2) = -10;
C ← A - 1 = 5 – 1 = 4;
D ← A – 2 = 5 – 2 = 3;

Sabendo desses valores, pode-se calcular o valor atribuído a H:

H ← (((4 * A) div D) - B) - pot(A,2) mod C, substituindo os valores;
H ← (((4* 5) div 3) - (-10)) - pot(5,2) mod 4;
H ← ((20 div 3) - (-10)) - pot(5,2) mod 4, resolvendo a multiplicação de 4 por 5, igual 20;
H ← (6 - (-10)) - pot(5,2) mod 4, div é o Quociente da divisão, 20 div 3 = 6;
H ← (6 + 10) - 25mod 4, pot é a potenciação, 5 elevado ao 2, que é igual a 25;
H ← 16 - 1, mod é o resto da divisão, 25 mod 4 = 1;

Portanto, H ← 15.

Gabarito: C

27. (QUADRIX – 2017 – SEDF/DF – Professor – Informática) É correto afirmar que o uso de algoritmos eficientes está relacionado ao emprego de estruturas de dados adequadas.

Comentários:

É difícil avaliar isso – trata-se de uma questão abstrata. No entanto, nós podemos fazer uma ilação e sugerir que a utilização de estruturas de dados adequadas garante uma maior eficiência no uso de algoritmos.

Gabarito: C

28. (IF/CE – 2017 – IF/CE – Técnico de Tecnologia da Informação) Observe a seguinte lógica de programação.

```
Inicio algoritmo
var
        N: inteiro
Inicio
        para N de 1 ate 9 faca
                se N mod 2 = 1 entao
                        escreva(N)
                fimse
        fimpara
fim algoritmo
```

Este algoritmo escreve a saída:

a) 3, 5, 7, 9
b) 1, 3, 5, 7, 9
c) 2, 4, 6, 8
d) 1, 2, 4, 6, 8
e) 1, 3, 5, 7, 8

Comentários:

Para cada número de 1 a 9, se o resto da divisão desse número por 2 for igual a 1, escreva esse número. Logo:

- 1 Mod 2: Resto 1 = 1. Logo, escreve 1;
- 2 Mod 2: Resto 0 ≠ 1. Logo, não escreve 2;
- 3 Mod 2: Resto 1 = 1. Logo, escreve 3;

- 4 Mod 2: Resto 0 ≠ 1. Logo, não escreve 4;
- 5 Mod 2: Resto 1 = 1. Logo, escreve 5;
- 6 Mod 2: Resto 0 ≠ 1. Logo, não escreve 6;
- 7 Mod 2: Resto 1 = 1. Logo, escreve 7;
- 8 Mod 2: Resto 0 ≠ 0. Logo, não escreve 8;
- 9 Mod 2: Resto 1 = 1. Logo, escreve 9;

Gabarito: B

---

29. (IF/PE – 2017 – IF/PE – Técnico de Laboratório - Informática para Internet) No que diz respeito a algoritmos, analise as proposições a seguir:

I. Algoritmo é uma sequência de procedimentos que são executados sequencialmente com o objetivo de resolver um problema específico.

II. O comando CASE não deve ser utilizado caso já exista no programa um comando IF.

III. Um algoritmo não representa, necessariamente, um programa de computador, e sim os passos necessários para realizar uma tarefa.

IV. Diferentes algoritmos não podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros.

V. Serve como modelo para programas, pois sua linguagem é intermediária à linguagem humana e às linguagens de programação, funcionando como uma boa ferramenta na validação da lógica de tarefas a serem automatizadas.

Estão CORRETAS as proposições:

a) I, IV e V.
b) II, III e IV.
c) I, III e V.
d) II, IV e V.
e) I, II e III.

Comentários:

(I) Certo. Definição de algoritmo, i.e., sequência de procedimento executados sequencialmente; (II) Errado. Pode-se utilizar o comando CASE caso já exista um IF;

(III) Certo. Algoritmo representa uma sequência de passo realizados, não necessariamente sendo um programa de computador; (IV) Errado. Pode-se ter diferente algoritmos realizando uma mesma tarefa, com diferentes instruções e diferentes tempos de execução; (V) Certo. Realiza as atividades de maneira lógica e sequencial, não somente utilizada para programas de computador.

Gabarito: C

---

30. (CESPE– 2017 – SEDF – DF – Professor de Educação Básica - Informática) Considere o algoritmo a seguir:

```
Inteiro x=1, y=4, z=5;
Enquanto (x<y) faça
        z=z+y%x;
        y=y-1;
        x=x+1;
Fim Enquanto
Imprima(z);
```

A operação % representa o resto da divisão entre dois inteiros. Assinale a alternativa que indica o valor que será impresso:

a) 5.
b) 6.
c) 7.
d) 8.
e) 9.

Comentários:

O algoritmo executa um loop enquanto o x for maior que y, nessa execução é adicionado ao valor de Z o resto da divisão entre x e y.

x = 1, y = 4, z = 5

x<y -> 1<4
z = z+y%x -> 5 = 5 + 4%1 = 5+0 = 5
y = y -1 = 4 -1 = 3
x = x + 1 = 1 + 1 = 2

x<y -> 2<3
z = z+y%x -> 5 = 5 + 3%2 = 5+1 = 6
y = y -1 = 3 -1 = 2
x = x + 1 = 2 + 1 = 3

x < y -> 3< 2? Não! Sai do loop e escreve z = 6.

Gabarito: B

31. (FCC– 2017 – TRT 24ª Região – MS – Técnico Judiciário - Tecnologia da Informação) Considere o algoritmo em pseudocódigo abaixo.

```
var v1, v2, v3: inteiro
início
  leia (v1, v2, v3)
  exiba(v1)
  enquanto v3>1 faça
     v1 ← v1 * v2
     v3 ← v3 - 1
     exiba(v1)
  fim_enquanto
fim
```

Se forem lidos para as variáveis v1, v2 e v3, respectivamente, os valores 3, 3 e 4, o último valor exibido será:

a) 729
b) 243
c) 27
d) 81
e) 128

Comentários:

O algoritmo executa um loop que multiplica os valores das variáveis v1 e v2, e coloca esse valor na variável v1, e faz esse passo até que a variável v3 seja menor que 1.

v1 = 3, v2 = 3, v3 = 4

v3 > 1 -> 4>1
v1 <- v1 * v2 = 3*3 = 9
v3 <- v3 - 1 = 4 -1 = 3

v3 > 1 -> 3>1
v1 <- v1 * v2 = 9*3 = 27
v3 <- v3 - 1 = 3 -1 = 2

v3 > 1 -> 3>1
v1 <- v1 * v2 = 27*3 = 81
v3 <- v3 - 1 = 2 -1 = 1

Exibe 81.

Gabarito: D

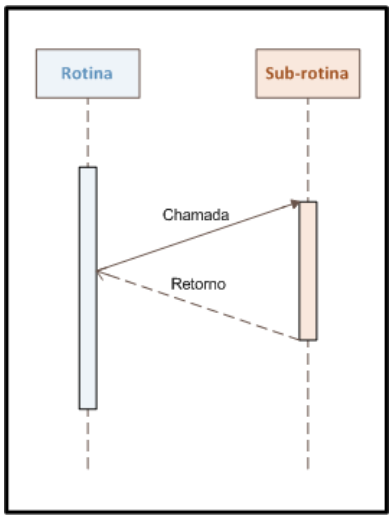
ACERTEI	ERREI

FUNÇÕES E PROCEDIMENTOS

Pessoal, é importante falarmos de alguns conceitos fundamentais! *O que é uma rotina?* Uma rotina nada mais é que um conjunto de instruções – um algoritmo. Uma sub-rotina (ou subprograma) é um pedaço menor desse conjunto de instruções que, em geral, será utilizado repetidamente em diferentes locais do sistema e que resolve um problema mais específico, parte do problema maior.

Em vez de repetir um mesmo conjunto de instruções diversas vezes no código, esse conjunto pode ser agrupado em uma sub-rotina que é chamada em diferentes locais. As sub-rotinas podem vir por meio de uma função ou de um procedimento. A diferença fundamental é que, no primeiro caso, retorna-se um valor e no segundo caso, não. *Entenderam?*

Galera, essa é a ideia geral! No entanto, algumas linguagens de programação têm funções e procedimentos em que nenhum retorna valor ou ambos retornam valor, ou seja, muitas vezes, sequer há uma distinção. No contexto da programação orientada a objetos, estas sub-rotinas são encapsuladas nos próprios objetos, passando a designar-se métodos.



A criação de sub-rotinas foi histórica e permitiu fazer coisas fantásticas. Não fossem elas, estaríamos fazendo *goto* até hoje. Uma sub-rotina é um trecho de código marcado com um ponto de entrada e um ponto de saída. *Entenderam?*

Quando uma sub-rotina é chamada, ocorre um desvio do programa para o ponto de entrada, e quando a sub-rotina atinge seu ponto de saída, o programa desaloca a sub-rotina, e volta para o mesmo ponto de onde tinha saído. Vejam a imagem ao lado!

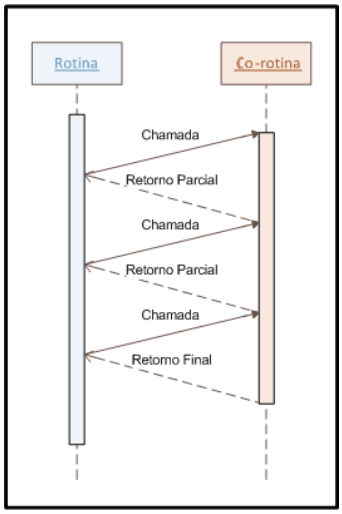
Algumas das vantagens na utilização de sub-rotinas durante a programação são: redução de código duplicado; possibilidade de reutilizar o mesmo código sem grandes alterações em outros programas; decomposição de problemas grandes em pequenas partes; melhorar a interpretação visual; esconder ou regular uma parte de um programa; reduzir o acoplamento; entre outros.

Quando uma sub-rotina termina, ela é desalocada das estruturas internas do programa, i.e., seu tempo de vida termina! É praticamente como se ela nunca tivesse existido. Podemos chamá-la de novo, isso é claro, mas será uma nova encarnação da sub-rotina: o programa novamente começará executando desde seu ponto de entrada. Uma co-rotina não funciona assim!

As co-rotinas são parecidas com as sub-rotinas, diferindo apenas na forma de transferência de controle, realizada na chamada e no retorno. Elas possuem um ponto de entrada e podemos dizer que são mais genéricas que as sub-rotinas. Na co-rotina, o trecho de código trabalha conjuntamente com o código chamador até que sua tarefa seja terminada.

O controle do programa é passado da co-rotina para o chamador e de volta para a co-rotina até que a tarefa da co-rotina termine. Numa co-rotina, existe o ponto de entrada (que é por onde se inicia o processamento), um ponto de saída (pela qual o tempo de vida da co-rotina termina), e um ponto de retorno parcial. Esse retorno parcial não termina com a vida da co-rotina.

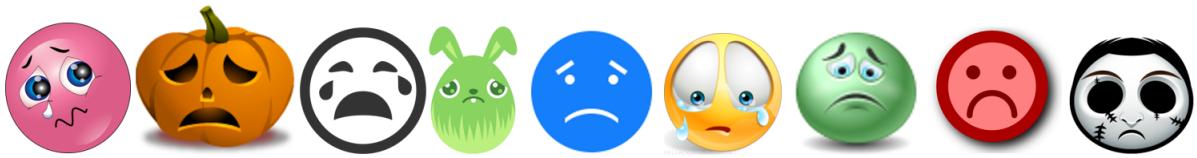
Pelo contrário, apenas passa o controle do programa para a rotina chamadora, e marca um ponto de reentrada. Quando a rotina chamadora devolver o controle para a co-rotina, o processamento começa a partir do último ponto de retorno parcial.



Na prática, uma co-rotina permite retornar valores diversas vezes antes de terminar o seu tempo de vida. Enquanto o tempo de vida das sub-rotinas é ditado pela pilha de execução, o tempo de vida das co-rotinas é ditado por seu uso e necessidade.

Por fim, é bom lembrar que existem funções pré-definidas. *O que é isso, professor?* Galera, é um conjunto de funções já prontinhas para serem utilizadas de cada linguagem de programação. Ex: na Linguagem C, temos: `ceil(x)`, que arredonda um número real x para cima; `sqrt(x)` que calcula a raíz quadrada de x; `strcat(x,y)`, que concatena duas strings; `strlen(x)`, que retorna a quantidade de caracteres, etc.

Infelizmente, eu não encontrei questões sobre esse tema.



Se alguém encontrar, eu posso gentilmente comentá-las.

PASSAGEM DE PARÂMETROS

Vamos falar sobre passagem de parâmetros por valor e por referência. Vocês sabem que, quando o módulo principal chama uma função ou procedimento, ele passa alguns valores chamados Argumentos de Entrada. Esse negócio costuma confundir muita gente, portanto vou explicar por meio de um exemplo, utilizando a função DobraValor(valor1, valor2) – apresentada na imagem abaixo:

```
#include <stdio.h>

void DobraValor(int valor1, int valor2)
{
    valor1 = 2*valor1;
    valor2 = 2*valor2;

    printf("Valores dentro da Função: \nValor 1 = %d\n Valor 2 = %d\n",valor1,valor2);
}

int main()
{
    int valor1 = 5;
    int valor2 = 10;

    printf("Valores antes de chamar a Função:\nValor 1 = %d\nValor 2 = %d\n",valor1,valor2);
    DobraValor(valor1,valor2);
    printf("Valores depois de chamar a Função:\nValor 1 = %d\nValor 2 = %d\n",valor1,valor2);

    return();
}
```

Essa função recebe dois valores e simplesmente multiplica sua soma por dois. *Então o que acontece se eu passar os parâmetros por valor para a função?* Bem, ela receberá uma cópia das duas variáveis e, não, as variáveis originais. Logo, antes de a função ser chamada, os valores serão os valores iniciais: 5 e 10. Durante a chamada, ela multiplica os valores por dois, resultando em: 10 e 20.

```
Valores antes de chamar a Função:
Valor 1 = 5
Valor 2 = 10
Valores dentro da Função:
Valor 1 = 10
Valor 2 = 20
Valores depois de chamar a Função:
Valor 1 = 5
Valor 2 = 10
```

Após voltar para a função principal, os valores continuam sendo os valores iniciais: 5 e 10, como é apresentado acima na execução da função. Notem que os valores só se modificaram dentro da função DobraValor( ). *Por que, professor?* Ora, porque foi

passada para função apenas uma cópia dos valores e eles que foram multiplicados por dois e, não, os valores originais.

```
#include <stdio.h>
void DobraValor(int *valor1, int *valor2)
{
        *valor1 = 2*(*valor1);
        *valor2 = 2*(*valor2);
        printf("Valores dentro da Função: \nValor 1 = %d\n Valor 2 = %d\n", *valor1,
*valor2);
}
int main()
{
        int valor1 = 5;
        int valor2 = 10;

        printf("Valores antes de chamar a Função:\nValor 1 = %d\nValor 2 =
%d\n",valor1,valor2);
        DobraValor(&valor1,&valor2);
        printf("Valores depois de chamar a Função:\nValor 1 = %d\nValor 2 =
%d\n",valor1,valor2);

        return();
}
```

Professor, o que ocorre na passagem por referência? Bem, ela receberá uma referência para as duas variáveis originais e, não, cópias. Portanto, antes de a função ser chamada, os valores serão os valores iniciais: 5 e 10. Durante a chamada, ela multiplica os valores por dois, resultando em: 10 e 20. Após voltar para a função principal, os valores serão os valores modificados: 10 e 20.

```
Valores antes de chamar a Função:
Valor 1 = 5
Valor 2 = 10
Valores dentro da Função:
Valor 1 = 10
Valor 2 = 20
Valores depois de chamar a Função:
Valor 1 = 10
Valor 2 = 20
```

Notem que os valores se modificaram não só dentro da função DobraValor( ), como fora também (na função principal). *Por que isso ocorreu, professor?* Ora, porque foi passada para função uma referência para os valores originais e eles foram multiplicados por dois, voltando à função principal com os valores dobrados! Por isso, os valores 10 e 20.

Resumindo: a passagem de parâmetro por valor recebe uma cópia da variável original e qualquer alteração não refletirá no módulo principal. A passagem de parâmetro por referência recebe uma referência para a própria variável e qualquer

alteração refletirá no módulo principal. Algumas linguagens permitem passagem por valor e por referência e outras permitem apenas uma dessas modalidades.

1. (FCC - 2012 – TJ/RJ – Analista Judiciário) O seguinte trecho de pseudo-código representa a definição de uma função (sub-rotina) f com um único argumento x.

```
f(x)
x ← x + 1
devolva x
```

Considere agora o seguinte trecho de código que invoca a função f definida acima.

```
a ← 0
escreva a
escreva f(a)
escreva a
```

A execução do trecho de código acima resultaria na escrita de:

a) 0, 1 e 0 no caso de passagem de parâmetros por valor e.
   0, 1 e 0 no caso de passagem de parâmetros por referência.

b) 0, 1 e 1 no caso de passagem de parâmetros por valor e.

0, 1 e 0 no caso de passagem de parâmetros por referência.

c) 0, 1 e 0 no caso de passagem de parâmetros por valor e.
0, 1 e 1 no caso de passagem de parâmetros por referência.

d) 0, 1 e 1 no caso de passagem de parâmetros por valor e.
0, 1 e 1 no caso de passagem de parâmetros por referência.

e) 0, 0 e 0 no caso de passagem de parâmetros por valor e.
0, 1 e 1 no caso de passagem de parâmetros por referência.

Comentários:

Vejam que questão interessante! Se o parâmetro `a` for passado por valor, quando ele retornar da sub-rotina, ele continuará com o mesmo valor inicial, caso contrário – se for passado por referência – ele continuará com o valor que saiu da sub-rotina. Logo, vamos para a questão:

```
a ← 0          // Atribui-se 0 a variável a;
escreva a      // Escreve 0;
escreva f(a)   // Escreve a = a + 1, ou seja, a = 0 + 1, ou seja a = 1;
escreva a      // Se for passagem por valor, escreve 0; se for por referência, escreve 1;
```

Ele escreve 0, 1 e 0, no caso de passagem de parâmetros por valor e 0, 1 e 1, no caso de passagem de parâmetros por referência.

Gabarito: C

---

2. (FCC - 2012 – ARCE – Analista Judiciário) Há duas maneiras de se passar argumentos ou parâmetros para funções: por valor e por referência. Todas as afirmativas sobre passagem de parâmetros estão corretas, EXCETO:

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

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

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

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

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

Comentários:

(a) Correto. Na passagem por valor, são passadas cópias do valor; na passagem por referência, são passados endereços de variáveis; (b) Correto. Na passagem por valor, são passadas cópias do valor; na passagem por referência, são passados endereços de variáveis; (c) Errado. Se ocorreu uma passagem por valor, ela é alterada apenas na sub-rotina, mas não no programa principal; (d) Correto. Conforme vimos, são passados endereços das variáveis, logo seu valor é alterado dentro da função e fora dela; (e) Correto. Conforme vimos, são passadas cópias do valor, logo seu valor é alterado apenas dentro da função, mas não fora dela.

Gabarito: C

---

3. (VUNESP – 2015 – TCE/SP – Analista de Sistemas) Um usuário implementou uma rotina de um programa, denominada Fatorial, e passou para essa rotina um parâmetro com o valor 6, mas deseja receber, após a execução da rotina, nesse mesmo parâmetro, o valor 6! (seis fatorial). Para isso, a passagem de parâmetro deverá ser por:

a) escopo.
b) hashing.
c) módulo.
d) referência.
e) valor.

Comentários:

Galera, observem a pegadinha da questão: ele manda 6 (seis) como parâmetro e no retorno da rotina o valor é 6! (seis fatorial). Observe que o valor foi modificado,

logo não pode ter sido uma passagem por valor - foi uma passagem por referência. Caso o retorno fosse 6 (seis), a passagem provavelmente seria por valor.

Gabarito: D

ACERTEI	ERREI

LINGUAGENS DE PROGRAMAÇÃO

Galera, o que seria uma linguagem de programação? Bem, nós podemos dizer que uma linguagem de programação é uma forma de um utilizador se comunicar com um computador. Essa comunicação é efetuada por meio de um conjunto de instruções, que – tal como em qualquer linguagem comum – obedecem a determinadas regras léxicas, sintáticas e semânticas.

Vocês sabem de uma coisa interessante? As linguagens de programação são anteriores ao advento do primeiro computador considerado efetivamente moderno. Se vocês pararem para pensar, se considerarmos os anos de 1940/50 como a altura em que as primeiras linguagens de programação modernas foram concebidas, a evolução das linguagens de programação foi extraordinária.

É de se ficar pasmo com a tremenda evolução a que tais linguagens foram sujeitas ao longo de pouco mais do que meio século. Embora a história da programação

não tenha nascido com tal linguagem, foi na década de 1950 que o Assembly, uma das primeiras linguagens de programação, apareceu para o mundo. *Quem aí já programou em Assembly na faculdade? MIPS?*

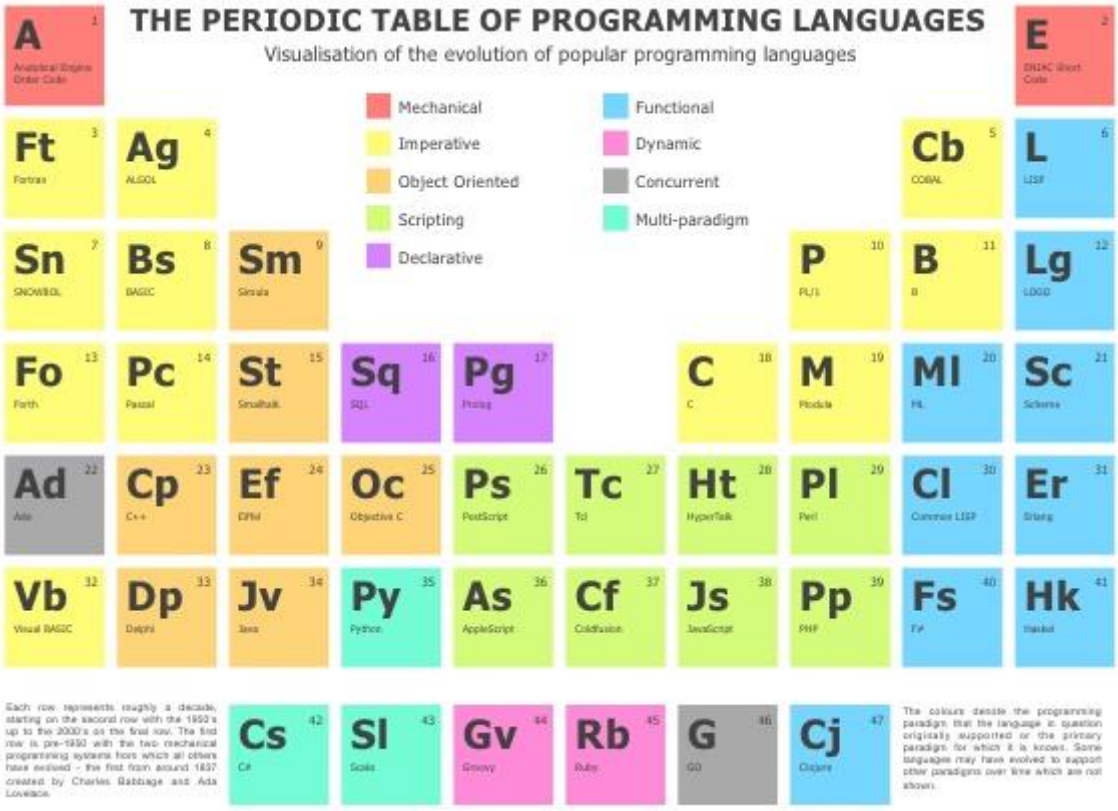
É considerada uma linguagem de baixo nível! *O que isso significa, professor?* Isso significa que a linguagem compreende as características da arquitetura do computador. Grosso modo, pode-se dizer que o nível da linguagem é proporcional a quanto você gasta pensando em resolver o seu problema (alto nível) ou em resolver problemas relacionados aos cálculos computacionais (baixo nível).

Linguagem de Alto Nível: linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana. Desse modo, as linguagens de alto nível não estão diretamente relacionadas à arquitetura do computador. O programador de uma linguagem de alto nível não precisa conhecer características do processador, como instruções e registradores.

Linguagem de Baixo Nível: linguagem que compreende as características da arquitetura do computador. Utiliza somente instruções do processador e para isso é necessário conhecer os registradores da máquina. Nesse sentido, estão diretamente relacionadas com a arquitetura do computador. Um exemplo é o Assembly que trabalha diretamente com os registradores do processador.

Quem já programou em Assembly sabe bem disso! No Assembly, para fazer uma tarefa simples, é necessário compreender a fundo as características e o funcionamento da arquitetura do computador. Galera, dá um trabalho absurdo (gosto nem de lembrar!) Ainda assim, em comparação com a programação em código binário, é uma linguagem bem mais fácil de entender e utilizar.

Com o objetivo de combater os problemas da programação em Assembly, John Backus criou, também na década de 1950, a linguagem FORTRAN (FORmula TRANslator), que é uma linguagem de alto nível e considerada uma das melhores da época. Ainda na mesma década, foram criados o LISP (LISt Processor) e o COBOL (COmmon Business Oriented Language).

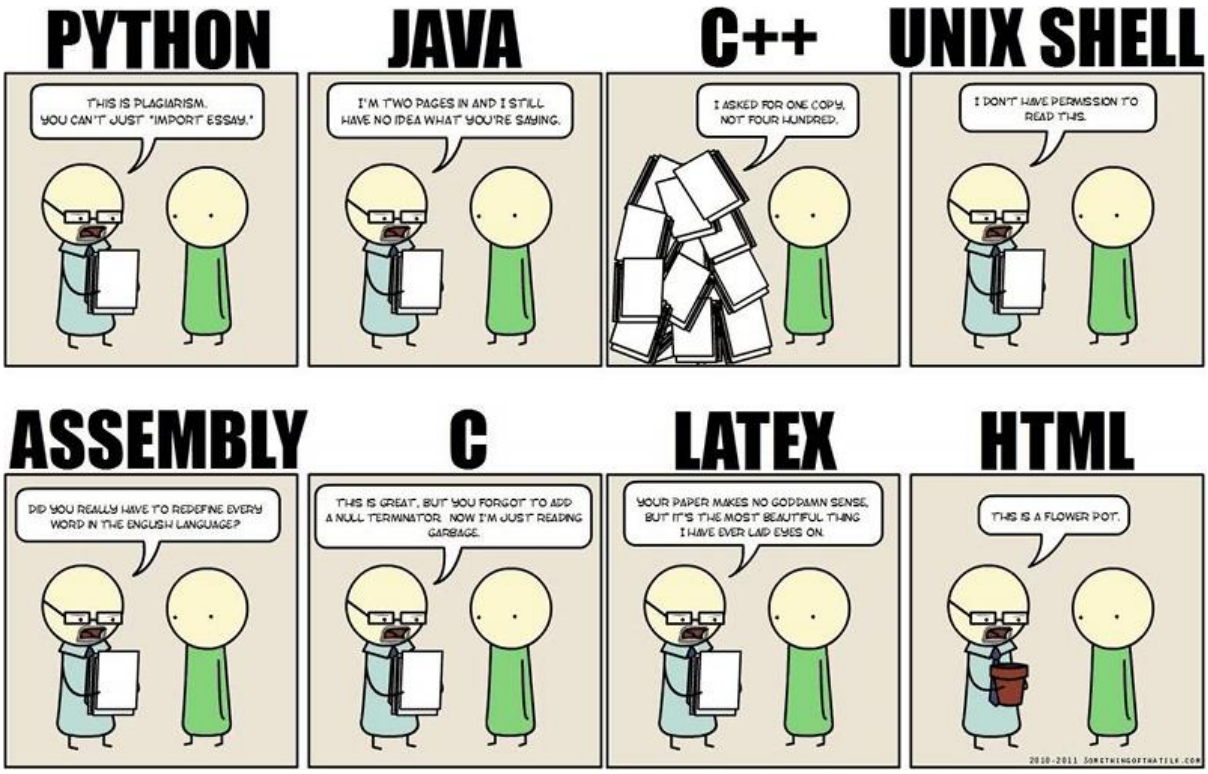


Galera, eu usei algumas dessas linguagens na faculdade! Por exemplo: usei FORTRAN na disciplina de Física Experimental e usei LISP na disciplina de Lógica Computacional. Quanto ao COBOL, eu nunca programei, mas tenho certeza que entre vocês tem alguém que trabalhou ou trabalha com algum sistema legado de bancos públicos, porque era a linguagem mais comum para mainframes.

Aliás, atualmente existe até COBOL orientado a objetos! Menção honrosa também ao ALGOL (ALGOrithmic Language), que é uma família de linguagens de programação de alto nível voltadas principalmente para aplicações científicas. Já na

década de 60 surgiu a APL, que era uma linguagem de programação destinada a operações matemáticas.

Surgiu também a Simula I, uma linguagem baseada em ALGOL 60, cuja versão posterior (Simula 67) foi a primeira linguagem de programação orientada a objetos, introduzindo os conceitos de classes e heranças. Surgiu ainda, em 1964, a linguagem BASIC (Beginner's All-purpose Symbolic Instruction Code), que foi criada com fins educativos. *Conheciam?* Eu também não!



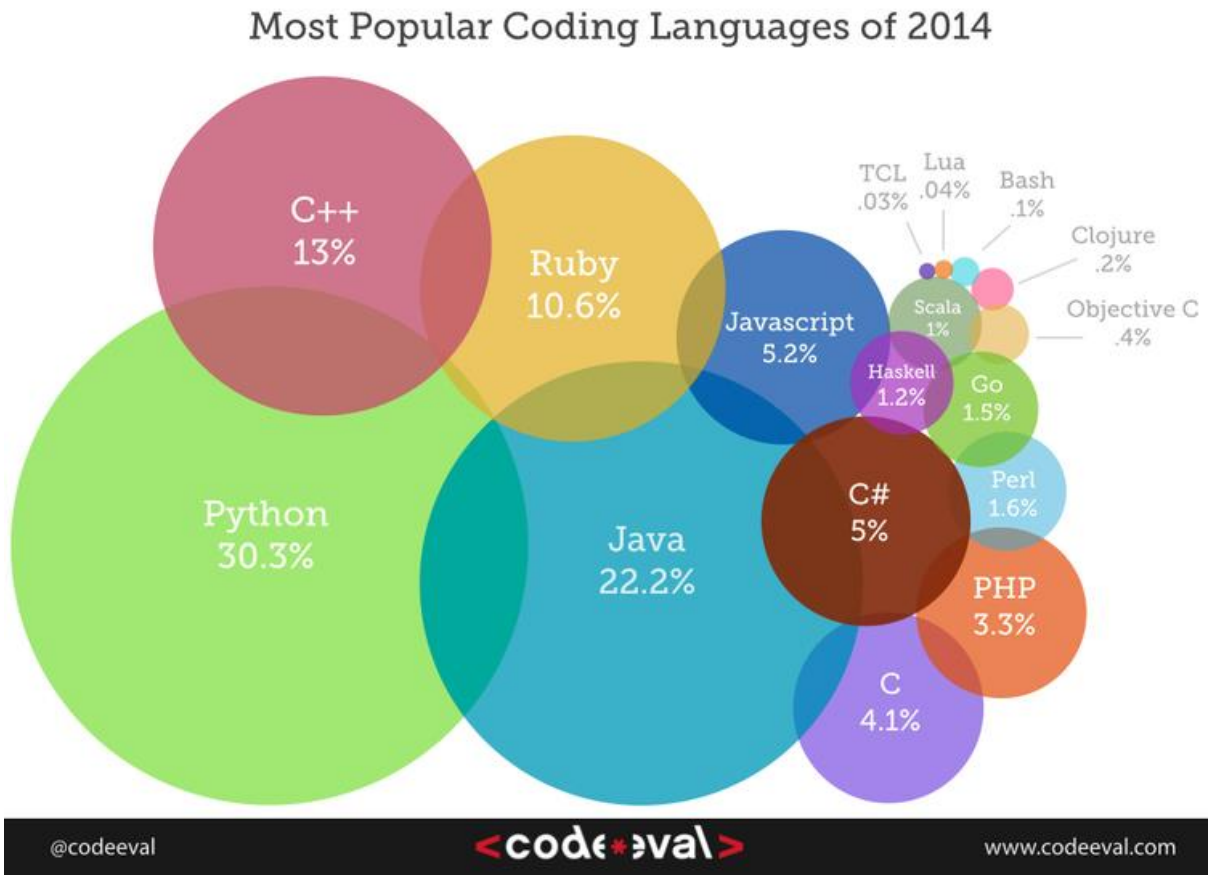
Todas estas linguagens foram criadas entre 1950 e 1960, e marcaram o início do desenvolvimento das linguagens de programação. Algumas destas linguagens, embora em versões mais recentes, são ainda utilizadas por vários programadores. O período compreendido entre o final dos anos 1960 à década de 1970 trouxe um grande florescimento de linguagens de programação.

A maioria dos principais paradigmas de linguagem utilizados atualmente foram inventados durante este período! A linguagem C foi uma das primeiras linguagens de programação de sistemas – desenvolvida pelos monstros Dennis Ritchie e Ken Thompson. SmallTalk forneceu uma base completa para o projeto de uma linguagem orientada a objetos.

Prolog foi a primeira linguagem de programação do paradigma lógico. Cada uma dessas línguas gerou toda uma família de descendentes, e linguagens mais modernas contam, pelo menos, com uma delas em sua ascendência. Nesse período, também surgiu o Pascal (eu aprendi a programar em Pascal) e SQL, que inicialmente era apenas uma linguagem de consulta.

A década de 80 veio mais para consolidar diversas linguagens de programação. O governo dos Estados Unidos padronizou a ADA, uma linguagem de programação de sistemas destinados à utilização por parte dos contratantes do ministério da defesa. No entanto, uma tendência nova e importante na concepção de linguagens foi o aumento do foco na programação de sistemas de larga escala.

Não só isso, mas com o uso de módulos! Na faculdade, eu fiz uma disciplina chamada Programação Estruturada cujo foco era justamente a modularização do código-fonte. Em 1983, veio o C++ (uma espécie de C orientado a objetos). Em 1987, veio o Perl, muito utilizada para administração de sistemas Linux e manipulação de strings. No final de 1988, eu nasci! =)



A década de 1990 não trouxe nenhuma grande novidade, no entanto fez uma combinação e maturação das ideias antigas. Uma filosofia de grande importância

era a produtividade do programador! Surgiram muitas linguagens para suportar o Desenvolvimento Rápido de Aplicações (RAD). Em geral, elas vieram com uma IDE, Garbage Collector, e tudo que aumentasse a produtividade do programador.

A grande maioria das linguagens já eram orientadas a objetos! Mais radicais e inovadoras do que as línguas RAD foram as novas linguagens de *scripting*. Estas não descenderam diretamente das outras linguagens e contaram com sintaxes novas e incorporação mais liberal de novas funcionalidades. As linguagens de *scripting* foram mais proeminentes utilizadas na programação web.

Em 1990, surgiu o Haskell, que era uma linguagem funcional de propósito geral. Em 1991, surgiu o Python, que é a queridinha de muita gente hoje em dia – utilizada no Google! Nesse mesmo ano, também veio o Java e revolucionou com o mundo todo da programação! Em 1993, veio o Ruby, que é outra queridinha atualmente dos programadores web.

Uma coisa bem legal! Nesse mesmo ano, surgiu a Lua, que é uma linguagem de scripting brasileira. Ela foi criada no Rio de Janeiro para ser utilizada em um projeto da Petrobrás! Devido à sua eficiência, clareza e facilidade de aprendizado, passou a ser usada em diversos ramos da programação, como no controle de robôs, processamento de textos e desenvolvimento de jogos (Ex: World of Warcraft).

Ficou orgulhoso do Brasil agora, não é? Em 1995, surgiu o JavaScript, que é até hoje uma das principais linguagens de front-end. Surgiu também o PHP, que dispensa comentários. Em 2000, surgiu o C#, criada pela Microsoft! Galera, e a evolução não parou! Atualmente, as linguagens de programação continuam crescendo, tanto na indústria quanto na pesquisa.

EXPRESSÕES REGULARES

Galera, para entender expressões regulares, eu vou dar um depoimento pessoal. Eu sempre tive um pouco de TOC! Fico alucinado com coisas desalinhadas, tortas, desproporcionais ou fora do padrão. E aqui nós chegamos à palavra-chave para entender expressões regulares: padrão. Vejam só: há um tempo atrás, não havia Spotify! Quem quisesse ouvir música digital, deveria possuir o arquivo em MP3.

E daí, professor? E daí que não existia uma fonte única para fazer o download das músicas. Em geral, você fazia um download de dezenas de fontes diferentes e cada uma possuia um padrão de nomenclatura para as músicas.

Havia fontes que nomeavam as músicas da seguinte forma:

- Nome da Banda - Nome da Música
- Ex: Chet Baker - Autumn Leaves

Outras fontes nomeavam as músicas da seguinte forma:

- Nome da Banda - Nome do Álbum - Nome da Música
- Ex: Chet Baker - She Was Too Good To Me - Autumn Leaves

Já outras nomeavam as músicas da seguinte forma:

- Nome da Banda - Nome do Álbum - Número da Música - Nome da Música
- Ex: Chet Baker - She Was Too Good To Me - 01 - Autumn Leaves.

E agora? Como eu faço com meu TOC com tantas músicas diferentes com formatações de nomes diversas? Bem, eu podia escolher uma formatação padrão única e modificar o nome arquivo por arquivo de cada música para o padrão escolhido. Isso seria um trabalho braçal insano, uma vez que eu tinha milhares de músicas.

Uma outra maneira de resolver meu problema era utilizar algum software de renomeação de múltiplos arquivos. E foi isso que eu fiz! Eu busquei na internet um software, encontrei, instalei e executei. Inseri uma expressão regular (nós veremos

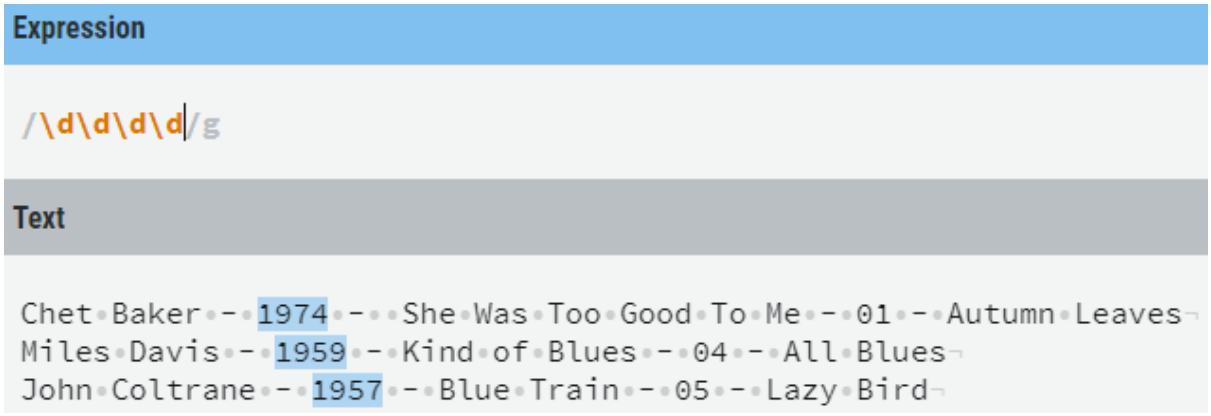
como mais para frente) que renomeava os arquivos de acordo com sua regra e renomeei todas as músicas da maneira que eu desejava.

No caso das minhas músicas, não me importava saber qual era o nome álbum ou número da música, então eu criava uma expressão regular que retirava tudo que não fosse "Nome da Banda - Nome da Música". *Mas afinal o que é uma expressão regular?* Galera, uma expressão regular é uma sequência de caracteres que definem um padrão de busca, geralmente utilizado para buscar padrões em strings.

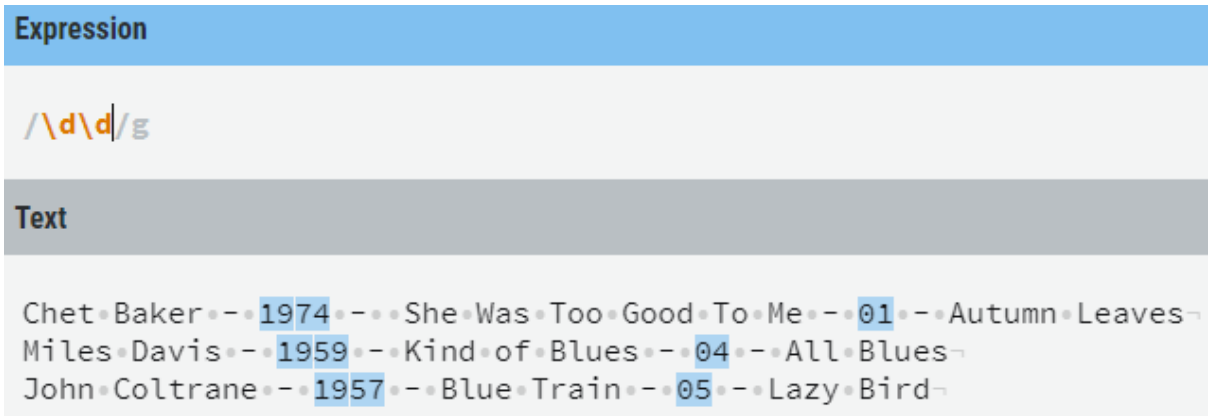
Vocês já utilizaram o *find-and-replace* do Word? É mais ou menos igual! Ok, muita falação e pouca prática! Vamos ver como funciona de verdade. Imaginem que eu tenha uma planilha com o nome da banda, ano do álbum, nome do álbum, número da música e nome da música, como no exemplo abaixo (um exemplo simplificado com apenas três músicas).

```
Chet Baker - 1974 -  She Was Too Good To Me - 01 - Autumn Leaves
Miles Davis - 1959 - Kind of Blues - 04 - All Blues
John Coltrane - 1957 - Blue Train - 05 - Lazy Bird
```

Galera, se eu quiser buscar apenas o ano dos álbuns, eu posso criar uma expressão regular como essa: **\d\d\d\d**. *O que ela significa?* Bem, **\d** é uma classe de dígitos (0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9). Vejam na imagem abaixo que a minha expressão regular encontrou todos os conjuntos de quatro dígitos possíveis (1974, 1959 e 1957). *Viram que legal?*



*S*e eu quiser retirar o ano, eu posso simplesmente substituir tudo que for encontrado por essa expressão regular por nada (vazio). Vamos ver o que acontece se minha expressão regular for apenas **\d\d**. Notem na imagem abaixo que ele encontrou todos os conjuntos de dois dígitos possíveis, logo encontrou também 01, 04 e 05. *Interessante, não?*



Galera, esses são exemplos simples, mas esse assunto é gigantesco. Há como fazer expressões regulares complicadíssimas. Vamos ver alguns outros exemplos:

- Um único caractere: a, b ou c:
- [abc]

- Qualquer caractere exceto: a, b ou c;
- [^abc]

- Um caractere entre a-z:
- [a-z]

- Qualquer caractere que não esteja entre a-z:
- [^a-z]

- Um caractere entre a-z ou A-Z:
- [a-zA-Z]

- Um único caractere:
- .

- Qualquer caractere de espaço em branco:
- \s

- Qualquer caractere que não seja espaço em branco:
- \S

- Qualquer dígito (já vimos):
- \d

- Qualquer caractere que não seja um dígito:

- \D
- Match entre a ou b:
- (a|b)

- Zero ou um caractere a:
- a?

- Zero ou mais caracteres a:
- a*

- Um ou mais caracteres a:
- a+

- Exatamente três caracteres a:
- a{3}

- Três ou mais caracteres a:
- a{3,}

- Entre três e seis caracteres a:
- a{3,6}

- Início de uma string:
- ^

- Fim de uma string:
- $

- Exatamente três caracteres a:
- a{3}

Galera, há muuuuuuitas outras possibilidades! Quem quiser brincar com isso (recomendo bastante), há os sites abaixo:

https://regex101.com

https://regexr.com

Por fim, é importante dizer que – por meio de expressões regulares – é possível especificar o que uma linguagem de programação aceita e o que ela não aceita.

1. (COPERVE – 2018 – UFSC – Tecnologia da Informação) Considere o texto abaixo e a expressão regular POSIX a seguir:

   Então olhou para mim. Pensava que olhava para mim pela primeira vez. Mas então, quando se virou por trás do abajur, e eu continuava sentindo sobre o ombro, nas minhas costas, seu escorregadio e oleoso olhar, compreendi que era eu quem a olhava pela primeira vez.

   Fonte: Olhos de cão azul - Gabriel García Márquez

Expressão regular: o+[a-z]*o

Assinale a alternativa que indica a quantidade de palavras do texto que têm correspondência com a expressão regular.

a) 1
b) 5
c) 2
d) 4
e) 6

Comentários:

Galera, pela expressão regular, a palavra procurada deve começar com o caractere o, depois deve ter nada ou qualquer quantidade de caracteres entre [a-z] e, por fim, outro caractere o. Logo, temos: olhou, ombro, escorregadio e oleoso. Agora notem a pegadinha: a questão pede a quantidade de palavras e só há duas palavras completas: ombro e oleoso.

Gabarito: C

2. (CESGRANRIO – 2010 – BNDES – Tecnologia da Informação) O diretório de um servidor Linux contém os arquivos a seguir.

AmorGlobal
InvestimentoBrasil
NSanders
PlanetasMarteNetuno
TransporteUrbano

Observe a expressão regular do comando.

du *[a-e]???o*

Considerando que a expressão regular não contém espaços em branco, quantos arquivos serão incluídos na contabilização do comando?

a) 1
b) 2
c) 3
d) 4
e) 5

Comentários:

Primeiro, lembremos que du é um comando para saber o espaço utilizado em disco, por pastas ou arquivos, de maneira rápida e fácil, diretamente no terminal.

Bem, estamos buscando nenhum ou qualquer quantidade de caracteres, seguido de um caractere entre 'a' e 'e', i.e., 'a', 'b', 'c', 'd' ou 'e', seguido de quaisquer três caracteres quaisquer, seguido de um caractere 'o' e seguido por nenhuma ou qualquer quantidade de caracteres. Logo, é obrigatório ter um caractere 'o', então está descartada a opção NSanders.

AmorGlobal
InvestimentoBrasil
~~NSanders~~
PlanetasMarteNetuno
TransporteUrbano

Vamos analisar as outras: antes do 'o', podemos ter quaisquer três caracteres quaisquer e antes devemos ter necessariamente um caractere entre 'a' e 'e', logo:

- AmorGlobal: observem que 'o' antes do 'rGl' não está entre [a-e];
- InvestimentoBrasil: observem que 'm' antes do 'ent' não está entre [a-e];
- PlanetasMarteNetuno: observem que 'e' antes do 'tun' está entre [a-e];
- TransporteUrbano: observem que 'a' antes do 'nsp' está entre [a-e].

Logo, a resposta é PlanetasMarteNetuno e TransporteUrbano.

Gabarito: B

ACERTEI	ERREI

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
LÓGICA DE PROGRAMAÇÃO

1. (CESGRANRIO – 2014 – EPE – Tecnologia da Informação) Analise o algoritmo abaixo, onde a%b representa o resto da divisão de a por b.

```
inicio
    inteiro x,y,i,r
    ler x
    ler y
    para i de 1 até x
      se (x%i=0) e (y%i=0) então
          r <- i
      fim se
    próximo
    escrever r
fim
```

Qual será a resposta, caso as entradas sejam 128, para x, e 56, para y?

a) 2
b) 8
c) 56
d) 64
e) 128

2. (CESGRANRIO – 2014 – PETROBRÁS - Analista de Sistemas) Analise o algoritmo abaixo em português estruturado.

```
algoritmo segredo;
variáveis
  x,y,z : inteiro;
fim-variáveis
início
   x:=15;
   y:=10;
   z:=0;
   enquanto y>0 faça
      z:=z+x;
      y:=y-1;
   fim-enquanto
   imprima(z);
fim
```

Que número seria impresso caso esse programa executasse?

a) 0
b) 10
c) 15
d) 100
e) 150

3. (CESGRANRIO – 2014 – BASA - Analista de Sistemas) A saída do algoritmo apresentado abaixo para as entradas 100 e 20, respectivamente, é

```
inicio
inteiro X, Y
Ler X
Ler Y
Enquanto X ≥ Y - 1 faz
X < - X – 1
Y < - Y + 2 Fim Enquanto
Escrever "saída =" , Y – X
Fim
```

a) -5
b) -2
c) 1
d) 4
e) 7

4. (CESGRANRIO – 2010 – PETROBRÁS – Técnico em Informático) Relacionado à programação de computadores, um algoritmo, seja qual for a sua complexidade e a linguagem de programação na qual será codificado, pode ser descrito por meio da:

a) reografia.
b) criptografia.
c) linguagem de marcação.
d) engenharia estruturada.
e) pseudolinguagem.

5. (FCC - 2010 - DPE-SP - Agente de Defensoria - Analista de Sistemas) É utilizada para avaliar uma determinada expressão e definir se um bloco de código deve ou não ser executado. Essa é a definição da estrutura condicional:

a) For
b) If...Then...Else
c) While
d) Do...While
e) Next

6. (FCC - 2010 – TRT/SE - Analista de Sistemas) Objeto que se constitui parcialmente ou é definido em termos de si próprio. Nesse contexto, um tipo especial de procedimento (algoritmo) será utilizado, algumas vezes, para a solução de alguns problemas. Esse procedimento é denominado:

a) Recursividade.
b) Rotatividade.
c) Repetição.
d) Interligação.
e) Condicionalidade.

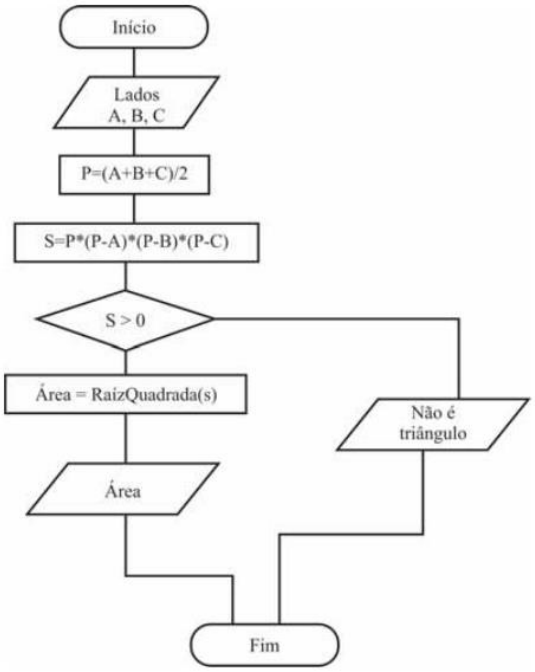
7. (FCC - 2009 – TJ/SE - Analista de Sistemas) A recursividade na programação de computadores envolve a definição de uma função que:

a) apresenta outra função como resultado.
b) aponta para um objeto.
c) aponta para uma variável.
d) chama uma outra função.
e) pode chamar a si mesma.

8. (CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos) Uma estrutura de repetição possibilita executar um bloco de comando, repetidas vezes, até que seja encontrada uma dada condição que conclua a repetição.

9. (CESPE - 2010 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Se um trecho de algoritmo tiver de ser executado repetidamente e o número de repetições for indefinido, então é correto o uso, no início desse trecho, da estrutura de repetição Enquanto.

10. (CESPE - 2013 - CNJ - Programador de computador) No fluxograma abaixo, se A = 4, B = 4 e C = 8, o resultado que será computado para Área é igual a 32.



11. (CESPE - 2011 - TJ-ES - Analista Judiciário - Análise de Banco de Dados - Específicos) Em uma estrutura de repetição com variável de controle, ou estrutura PARA, a verificação da condição é realizada antes da execução do corpo da sentença, o que impede a reescrita desse tipo de estrutura por meio de estrutura de repetição pós-testada.

12. (CESPE - 2010 – DETRAN/ES - Analista de Sistemas) O método de recursividade deve ser utilizado para avaliar uma expressão aritmética na qual um procedimento pode chamar a si mesmo, ou seja, a recursividade consiste em um método que, para que possa ser aplicado a uma estrutura, aplica a si mesmo para as subestruturas componentes.

13. (CESPE - 2013 – CPRM - Analista de Sistemas) Na implementação de recursividade, uma das soluções para que se evite o fenômeno de terminação do programa – que possibilita a ocorrência de um looping infinito – é definir uma função ou condição de terminação das repetições.

14. (CESPE - 2014 – ANATEL - Analista de Sistemas) A recursividade é uma técnica que pode ser utilizada na implementação de sistemas de lógica complexa, com a finalidade de minimizar riscos de ocorrência de defeitos no software.

15. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Tanto a recursividade direta quanto a indireta necessitam de uma condição de saída ou de encerramento.

16. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo.

```
Atribuir 13 a X;
Repetir
      Atribuir X – 2 a X;
      Imprimir (X);
Até que X < -1;
```

A estrutura será executada até que X seja igual ao seguinte valor:

a) – 1
b) – 3

17. (CONSULPLAN - 2012 - TSE - Programador de computador) Observe o trecho de pseudocódigo, que mostra o emprego da estrutura de controle enquanto ... faça ...

```
atribuir 0 a n;
enquanto n < 7 faça
   início
      imprimir (n);
      atribuir n+1 a n;
   fim;
```

A opção que utiliza a estrutura para ... faça ... correspondente, que gera o mesmo resultado, é:

a) Para n de 0 até 6 faça imprimir(n);
b) Para n de 0 até 7 faça imprimir(n);

18. (FEPESE - 2010 - SEFAZ-SC - Auditor Fiscal da Receita Estadual - Parte III - Tecnologia da Informação) Assinale a alternativa correta a respeito das variáveis e constantes, utilizadas em diversas linguagens de programação.

a) O número de constantes deve ser menor ou igual ao número de variáveis em um programa.

b) O número de constantes deve ser menor ou igual ao número de procedimentos em um programa.

c) O número de constantes deve ser igual ao número de variáveis em um programa.

d) O número de constantes independe da quantidade de variáveis em um programa.

e) O número de constantes deve ser igual ao número de procedimentos em um programa.

19. (NUCEPE - 2015 – SEDUC/PI - Analista de Sistemas) O código abaixo é usado para calcular o fatorial de números. Assinale a alternativa CORRETA sobre esse código:

```
função fatorial(n)
{
  se (n <= 1)
    retorne 1;
  senão
    retorne n * fatorial(n-1);
}
```

a) Este é um exemplo de procedimento.

b) O comando retorne pode ser retirado do código e a função terá o mesmo efeito.

c) Exemplo clássico de recursividade.

d) Não é possível chamar a função fatorial dentro dela mesma.

e) O resultado da função sempre retornará um valor elevado a ele mesmo (valor ^ valor).

20. (IADES - 2011 – PG/DF - Analista Jurídico - Analista de Sistemas) Os algoritmos são compostos por estruturas de controle de três tipos: sequencial, condicional e de repetição. Assinale a alternativa que apresenta apenas um tipo de estrutura de controle:

a) ...
escreva (“Digite seu nome: ”)
leia (nome)

```
escreva (“Digite sua idade: ”)
leia (idade)
limpe a tela
escreva (“Seu nome é:”, nome)
escreva (“Sua idade é:”, idade)
se (nome = “João”) entao
  se (idade > 18) entao
     escreva (nome, “ é maior de 18 anos!”)
  fim se
fim se
...
```

b) ...

```
escreva (“Pressione qualquer tecla para começar...”)
leia (tecla)
mensagem ← “Não devo acordar tarde...”
numero ← 0
enquanto (numero < 100)
     escreva (mensagem)
     numero ← (numero + 1)
fim enquanto
escreva (“Pressione qualquer tecla para
terminar...”)
leia (tecla)
escreva (“Tecla digitada: ”)
escreva (tecla)
...
```

c) ...

```
leia (nome)
escreva (“nome digitado: ”)
escreva (nome)
se (nome = “Wally”) entao
    escreva (“Encontrado o Wally!”)
senao
    cont ← 5
    enquanto (cont > 0)
      escreva (“Não é Wally”...”)
      cont ← (cont – 1)
    fim enquanto
```

```
    fim se
    ...
```

d) ...

```
    var
      nome: literal
      num: inteiro
    inicio
      escreva (“Digite seu nome: ”)
      leia (nome)
      num ← 0
      se (nome = “José”) entao
          num ← (num + 1)
      fim se
     escreva (“Quantidade de João encontrados:
    ”)
     escreva (num)
    ...
```

e) ...

```
    var
       nome: literal
       idade: inteiro
    inicio
      escreva (“Digite seu nome: ”)
      leia (nome)
      escreva (“Digite sua idade: ”)
      leia (idade)
      limpe a tela
      escreva (“Seu nome é:”)
      escreva (nome)
      escreva (“Sua idade é:”)
      escreva (idade)
    fim algoritmo
    ...
```

21. (IADES - 2011 – TRE-PA - Programador de Computador)

```
VAR
N1, N2 : INTEIRO;
N1 ← 2;
```

```
N2 ← 30;
INICIO
ENQUANTO N1<N2 FAÇA
      N2 ← N2 + N1;
      N1 ← N1 * 3;
FIM ENQUANTO;
N1 ← N2 + 11;
FIM
```

Dado o algoritmo escrito em pseudocódigo, quais os valores de N1 e N2, respectivamente, ao final da execução?

a) 162 e 110.
b) 110 e 121.
c) 110 e 162.
d) 121 e 110.
e) 173 e 110.

22. (CESPE – 2017 – TRE/BA – Analista Judiciário – Analista de Sistemas) Assinale a opção que apresenta a saída resultante da execução do algoritmo antecedente.

```
var a = 0, b = 1, f = 1;
      for (var i = 2; i <= 6; i++) {
            f = a + b;
            a = b;
            b = f;
      document.write(b);
      }
```

a) 123456
b) 12121
c) 12345
d) 0112
e) 12358

23. (CONSULPLAN - 2012 - TSE – Técnico – Programação de Sistemas) Analise o pseudocódigo, que ilustra o uso de uma função recursiva.

```
programa PPRRGG;
variáveis
   VERDE, AZUL : numérica;
função FF(AUX:numérica):numérica;
início
   atribuir VERDE+1 a VERDE;
   se AUX <= 2
   então atribuir 5 a FF
   senão atribuir AUX*FF(AUX-1) a FF;
fim; { fim da função FF }
início
  atribuir 0 a VERDE;
  atribuir FF(4) a AZUL;
  escrever(VERDE,AZUL);
fim.
```

O valor de retorno de FF e a quantidade de vezes que a função será executada serão, respectivamente,

a) 5 e 1.
b) 15 e 2.
c) 60 e 3.
d) 300 e 4.

24. (CESPE – 2017 – TRE/BA - Analista de Sistemas)

```
var i = 0;
 while (i < 5) {
   i++;
   if (i == 3) {
         continue;
   }
 document.write(i);
 }
```

Assinale a opção que apresenta a saída resultante da execução do algoritmo antecedente.

a) 12345
b) 1245
c) 3
d) 0124
e) 1234

25. (VUNESP – 2014 – SP/URBANISMO – Analista Administrativo) Analise o algoritmo a seguir, apresentado na forma de uma pseudolinguagem (Português Estruturado). Esse algoritmo deverá ser utilizado para responder às questões.

```
Início
  Inteiro x1, x2, x3, i;
  Leia x1, x2, x3;
  Para i de 1 até x1 faça
  [
    x2 ← x1 + x3;
    x3 ← x1 - x2;
  ]
Imprima (x2 + x3);
Fim
```

Considere que os valores lidos para x1, x2 e x3 tenham sido, respectivamente, 5, 4 e 3. É correto afirmar que o valor impresso ao final da execução do algoritmo é igual a:

a) –3
b) 0
c) 5
d) 8
e) 11

26. (CESPE – 2017 – TRT 7ª Região – CE – Técnico Judiciário – Tecnologia da Informação)

```
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

27. (QUADRIX – 2017 – SEDF/DF – Professor – Informática) É correto afirmar que o uso de algoritmos eficientes está relacionado ao emprego de estruturas de dados adequadas.

28. (IF/CE – 2017 – IF/CE – Técnico de Tecnologia da Informação) Observe a seguinte lógica de programação.

```
Inicio algoritmo
var
        N: inteiro
Inicio
        para N de 1 ate 9 faca
                se N mod 2 = 1 entao
                        escreva(N)
                fimse
        fimpara
fim algoritmo
```

Este algoritmo escreve a saída:

a) 3, 5, 7, 9
b) 1, 3, 5, 7, 9
c) 2, 4, 6, 8
d) 1, 2, 4, 6, 8
e) 1, 3, 5, 7, 8

29. (IF/PE – 2017 – IF/PE – Técnico de Laboratório - Informática para Internet) No que diz respeito a algoritmos, analise as proposições a seguir:

I. Algoritmo é uma sequência de procedimentos que são executados sequencialmente com o objetivo de resolver um problema específico.

II. O comando CASE não deve ser utilizado caso já exista no programa um comando IF.

III. Um algoritmo não representa, necessariamente, um programa de computador, e sim os passos necessários para realizar uma tarefa.

IV. Diferentes algoritmos não podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros.

V. Serve como modelo para programas, pois sua linguagem é intermediária à linguagem humana e às linguagens de programação, funcionando como uma boa ferramenta na validação da lógica de tarefas a serem automatizadas.

Estão CORRETAS as proposições:

a) I, IV e V.
b) II, III e IV.
c) I, III e V.
d) II, IV e V.
e) I, II e III.

30. (CESPE– 2017 – SEDF – DF – Professor de Educação Básica - Informática) Considere o algoritmo a seguir:

```
Inteiro x=1, y=4, z=5;
Enquanto (x<y) faça
        z=z+y%x;
        y=y-1;
        x=x+1;
Fim Enquanto
Imprima(z);
```

A operação % representa o resto da divisão entre dois inteiros. Assinale a alternativa que indica o valor que será impresso:

a) 5.
b) 6.
c) 7.
d) 8.
e) 9.

31. (FCC– 2017 – TRT 24ª Região – MS – Técnico Judiciário - Tecnologia da Informação) Considere o algoritmo em pseudocódigo abaixo.

```
var v1, v2, v3: inteiro
início
  leia (v1, v2, v3)
  exiba(v1)
  enquanto v3>1 faça
     v1 ← v1 * v2
     v3 ← v3 - 1
     exiba(v1)
  fim_enquanto
fim
```

Se forem lidos para as variáveis v1, v2 e v3, respectivamente, os valores 3, 3 e 4, o último valor exibido será:

a) 729
b) 243
c) 27
d) 81
e) 128

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)

PASSAGENS DE PARÂMETROS

1. (FCC - 2012 – TJ/RJ – Analista Judiciário) O seguinte trecho de pseudo-código representa a definição de uma função (sub-rotina) f com um único argumento x.

```
f(x)
x ← x + 1
devolva x
```

Considere agora o seguinte trecho de código que invoca a função f definida acima.

```
a ← 0
escreva a
escreva f(a)
escreva a
```

A execução do trecho de código acima resultaria na escrita de:

a) 0, 1 e 0 no caso de passagem de parâmetros por valor e.
0, 1 e 0 no caso de passagem de parâmetros por referência.

b) 0, 1 e 1 no caso de passagem de parâmetros por valor e.
0, 1 e 0 no caso de passagem de parâmetros por referência.

c) 0, 1 e 0 no caso de passagem de parâmetros por valor e.
0, 1 e 1 no caso de passagem de parâmetros por referência.

d) 0, 1 e 1 no caso de passagem de parâmetros por valor e.
0, 1 e 1 no caso de passagem de parâmetros por referência.

e) 0, 0 e 0 no caso de passagem de parâmetros por valor e.
0, 1 e 1 no caso de passagem de parâmetros por referência.

2. (FCC - 2012 – ARCE – Analista Judiciário) Há duas maneiras de se passar argumentos ou parâmetros para funções: por valor e por referência. Todas as afirmativas sobre passagem de parâmetros estão corretas, EXCETO:

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

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

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

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

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

3. (VUNESP – 2015 – TCE/SP – Analista de Sistemas) Um usuário implementou uma rotina de um programa, denominada Fatorial, e passou para essa rotina um parâmetro com o valor 6, mas deseja receber, após a execução da rotina, nesse mesmo parâmetro, o valor 6! (seis fatorial). Para isso, a passagem de parâmetro deverá ser por:

a) escopo.
b) hashing.
c) módulo.
d) referência.
e) valor.

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
EXPRESSÕES REGULARES

1. **(COPERVE – 2018 – UFSC – Tecnologia da Informação)** Considere o texto abaixo e a expressão regular POSIX a seguir:

   Então olhou para mim. Pensava que olhava para mim pela primeira vez. Mas então, quando se virou por trás do abajur, e eu continuava sentindo sobre o ombro, nas minhas costas, seu escorregadio e oleoso olhar, compreendi que era eu quem a olhava pela primeira vez.

   Fonte: Olhos de cão azul - Gabriel García Márquez

Expressão regular: o+[a-z]*o

Assinale a alternativa que indica a quantidade de palavras do texto que têm correspondência com a expressão regular.

a) 1
b) 5
c) 2
d) 4
e) 6

2. **(CESGRANRIO – 2010 – BNDES – Tecnologia da Informação)** O diretório de um servidor Linux contém os arquivos a seguir.

AmorGlobal
InvestimentoBrasil
NSanders
PlanetasMarteNetuno
TransporteUrbano

Observe a expressão regular do comando.

du *[a-e]???o*

Considerando que a expressão regular não contém espaços em branco, quantos arquivos serão incluídos na contabilização do comando?

a) 1
b) 2
c) 3
d) 4
e) 5

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
LÓGICA DE PROGRAMAÇÃO

1	2	3	4	5	6	7	8	9	10
B	E	D	E	B	A	E	C	C	E
11	12	13	14	15	16	17	18	19	20
E	C	C	E	C	B	A	D	C	E
21	22	23	24	25	26	27	28	29	30
D	E	C	B	C	C	C	B	C	B
31	32	33	34	35	36	37	38	39	40
D									

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
PASSAGEM DE PARÂMETROS

1	2	3	4	5	6	7	8	9	10
C	C	D							

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
EXPRESSÕES REGULARES

1	2	3	4	5	6	7	8	9	10
C	B								

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.