Estratégia Concursos

Aula 00

Prefeitura de Niterói-RJ - Seplag (APPGG - Analista de Políticas Públicas e Gestão Governamental - Área Gestão de Tecnologia) Desenvolvimento de software

Autor:
Paolla Ramos

01 de Fevereiro de 2024

Índice

APRESENTAÇÃO

PROF. PAOLLA RAMOS

FORMADA EM SISTEMAS DE INFORMAÇÃO PELA UNIVERSIDADE FEDERAL DE OURO PRETO,
PÓS-GRADUADA EM SISTEMAS DE INFORMAÇÃO
DIREITO TRIBUTÁRIO
DIREITO ADMINISTRATIVO
AUDITORA FISCAL ESPECIALISTA EM TI.

Olá, pessoal!! Meu nome é Paolla Ramos, sou Auditora Fiscal especialista em TI do ISS-Aracaju. Trabalhar nesse fisco incrível tem sido uma experiência fantástica!!

Pessoal, eu sou uma pessoa normal, assim como vocês. No início, achava que conquistar a aprovação em um concurso de alto nível era quase impossível, até que provei o contrário! Querem saber qual foi o segredo? Foi o hiper foco, galera! Não existe uma fórmula mágica, e eu nunca fui considerada "superinteligente" ou a primeira aluna na turma. No entanto, sempre fui MUITO DETERMINADA, PERSISTENTE.

A equipe de TI e eu estamos aprimorando nossas aulas de forma gradativa para oferecer o melhor conteúdo possível. Sabemos que o estudo pode ser complexo, especialmente por meio de livros eletrônicos, por isso, recomendo estudar em conjunto com as vídeo-aulas.

Além disso, informo que estamos trabalhando na atualização dos cursos neste exato momento! Estamos refazendo a formatação, adicionando questões e diagramas, entre outros aprimoramentos. Gradualmente, os cursos ficaram mais completos e aprofundados. E, para acompanhar as tecnologias mais recentes, novas videoaulas também estão a caminho.

Caso surja alguma demanda, não hesitem em contactar no fórum. Se preferirem, também podem entrar em contato pelo Instagram **@prof.paollaramos**. Eu amo ajudar os alunos e estou disponível para esclarecer quaisquer dúvidas que possam surgir.

A minha missão aqui é dar o meu melhor para ajudar cada um de vocês a conquistar a aprovação também! Podem contar comigo sempre que precisarem.

Então, minha ideia aqui é fazer o meu melhor para que você também consiga ser aprovado! Sempre que precisar, pode contar comigo. Meu instagram é:

ESTRATÉGIA FLASHCARDS

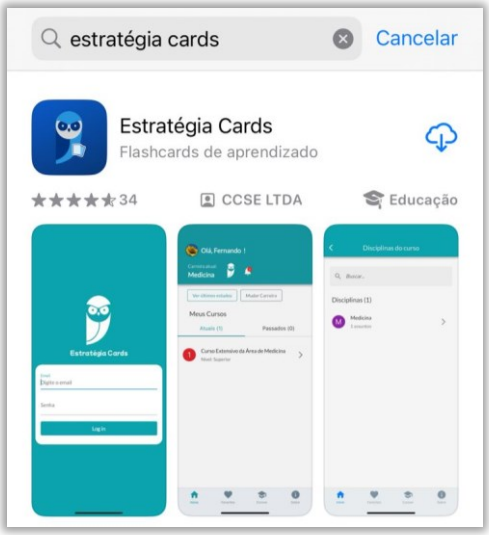
📚 Você tem dificuldade de estudar, memorizar e revisar os conteúdos que estuda em nossas aulas? Então nós temos a ferramenta perfeita para você!

Apresentamos o **Estratégia Cards**: app de flashcards que vai revolucionar sua forma de **estudar** e **revisar** conteúdos de provas de concurso público. Com nossa tecnologia inovadora e interface amigável, você dominará os tópicos mais complexos de maneira eficiente e divertida.

✨ Recursos do Estratégia Cards:

Curadoria de Flashcards	Flashcards criados e revisados por professores especializados em cada área, com qualidade e voltados para concursos públicos.
Flashcards Personalizados	Crie seus próprios flashcards, cobrindo os principais tópicos e matérias dos concursos públicos.
Repetição Espaçada	Técnica de aprendizagem que envolve revisar informações em intervalos crescentes para melhorar a retenção de longo prazo e combater o esquecimento.
Estatísticas Personalizadas	Visualize graficamente o percentual de acertos, erros ou dúvidas dos decks estudados.
Modo Offline	Estude em qualquer lugar, mesmo sem conexão à internet, fazendo o download dos decks.
Estudo por Áudio	*Está dirigindo ou fazendo esteira e quer continuar estudando?* Basta utilizar a opção "Escutar".
Decks Favoritos	Você pode escolher decks específicos como favoritos e visualizá-los em uma aba separada do app.
Opções de Estudo	Você poderá estudar todos os cards de um deck; ou apenas os que você errou; ou apenas os que você não estudou ainda; entre outras opções.

E como eu consigo baixar?



É muito fácil! Basta pesquisar por "Estratégia Cards" na loja oficial do seu smartphone.

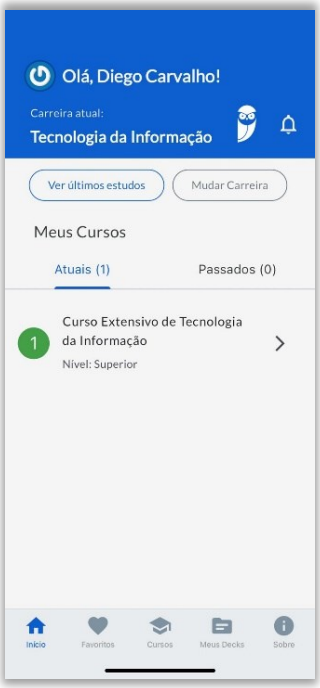
Se você tiver um Android, basta acessar a **Google Play**;

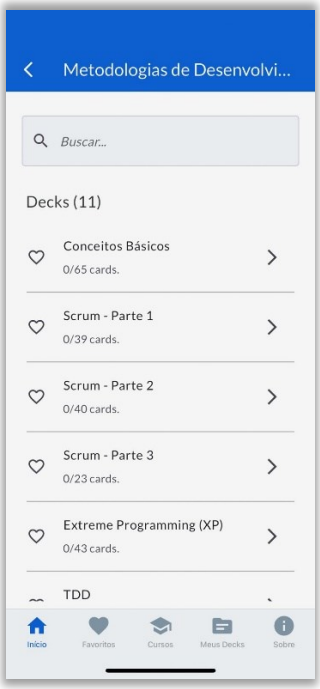
Se for tiver um iPhone, basta acessar a **App Store (iOS)**.

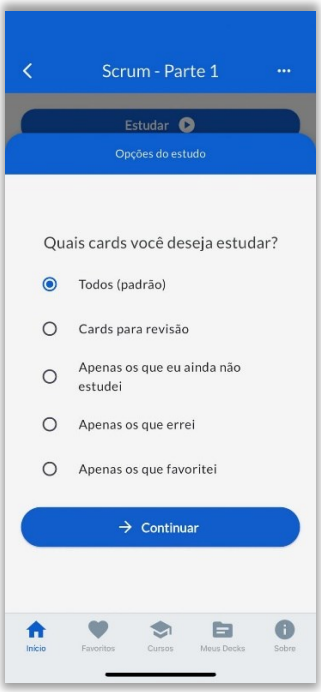
E para acessar?

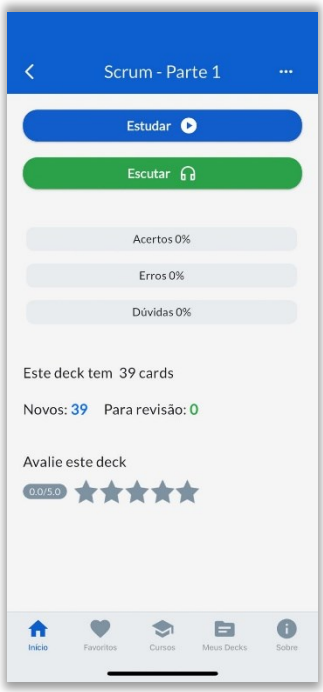
Para acessar, basta ter uma conta no Estratégia Concursos. Em seguida, utilize suas credenciais de login e senha para acessar o aplicativo. Por fim, acessa a carreira de Tecnologia da Informação.

Como utilizar o app:









LÓGICA DE PROGRAMAÇÃO

Conceitos Básicos

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**.

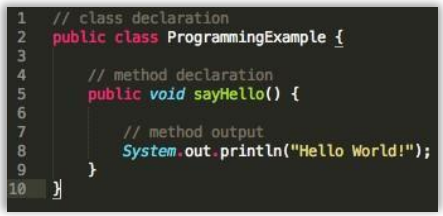
(Quadrix – 2022 – CRC-PR) A respeito dos diagramas de casos de uso, dos diagramas de classe, da análise essencial e da lógica de programação, julgue o item a seguir

Na lógica de programação, um algoritmo é conceituado como uma sequência estruturada e organizada de passos que tem por objetivo atingir um objetivo, seja ele definido ou indefinido.

Comentários: De fato, na lógica de programação, um algoritmo é uma sequência estruturada e organizada de passos para atingir um objetivo. Mas esse objetivo é sempre definido, e nunca indefinido. (Gabarito: Errado)

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**.



```
// class declaration
public class ProgrammingExample {

    // method declaration
    public void sayHello() {

        // method output
        System.out.println("Hello World!");
    }
}
```

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. Em um próximo tópico, vamos detalhar um pouco mais as principais linguagens de programação do mercado.

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
            <instru
    ções> senão
            <instru
    ções> 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.

(CESPE / CEBRASPE – 2022 – BANRISUL) Julgue o próximo item, a respeito de lógica de programação.

"O fluxograma é uma das formas de se representarem as instruções de um programa, utilizando-se de alguns comandos genéricos ou primitivos para a interpretação do algoritmo."

Comentários: Os comandos genéricos ou primitivos são utilizados em pseudocódigo, e não em fluxogramas. O fluxograma é uma das formas de se representar as instruções de programa, mas utiliza-se de formas gráficas, que enfatizam os passos individuais e o fluxo de execução de um algoritmo. (Gabarito: Errado)

Os principais símbolos de um Fluxograma são:

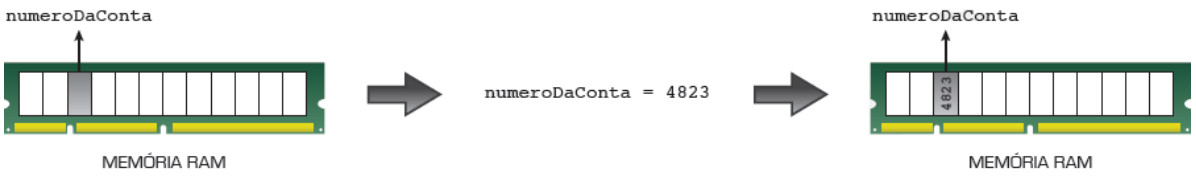
(CESPE/CEBRASPE – 2015 – TER-GO) Julgue os itens a seguir, relativos a lógica de programação.
Comumente usados em fluxogramas representativos de sistemas, os símbolos abaixo correspondem, respectivamente, a dados armazenados, processo, documento e entrada manual.

Comentários: Os símbolos em questão significam, respectivamente: dados externos, documento, processo e entrada manual. Portanto, está errado. (Gabarito: Errado).

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.

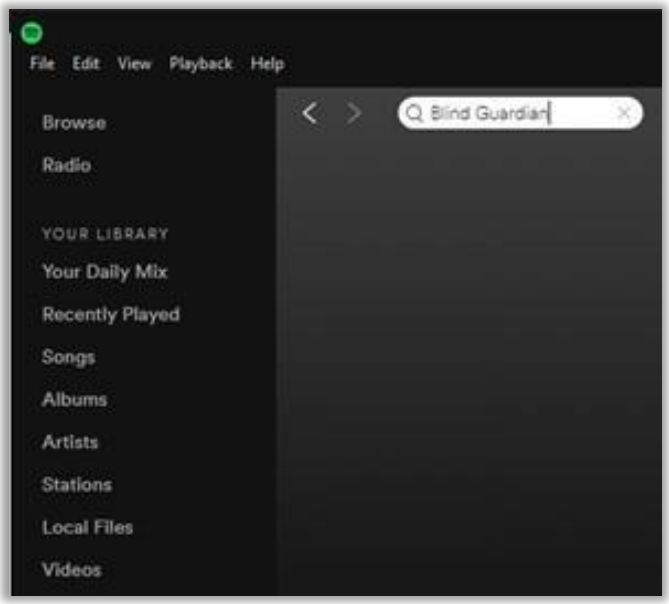


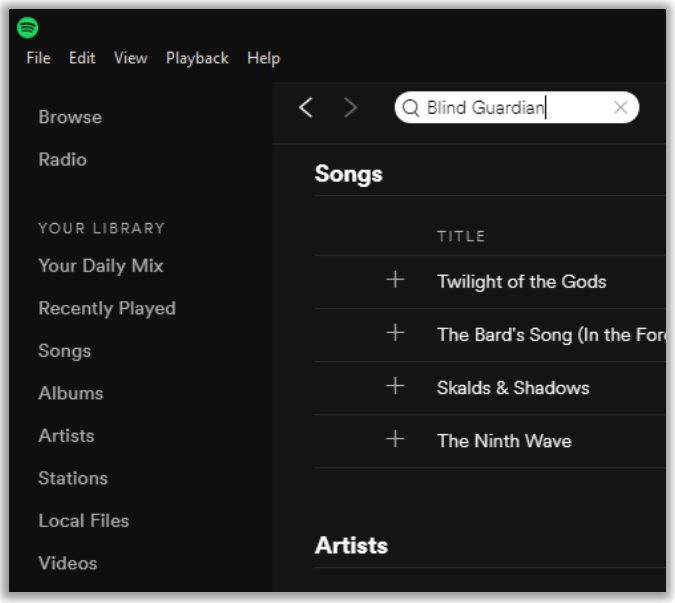
O conteúdo de uma variável pode ser alterado, consultado ou apagado diversas vezes durante a execução de um algoritmo, porém o valor apagado é perdido. Bacana, mas e a atribuição? Bem, trata-se de uma notação para associar um valor a uma variável, i.e., armazenar o conteúdo no endereço de memória específico.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

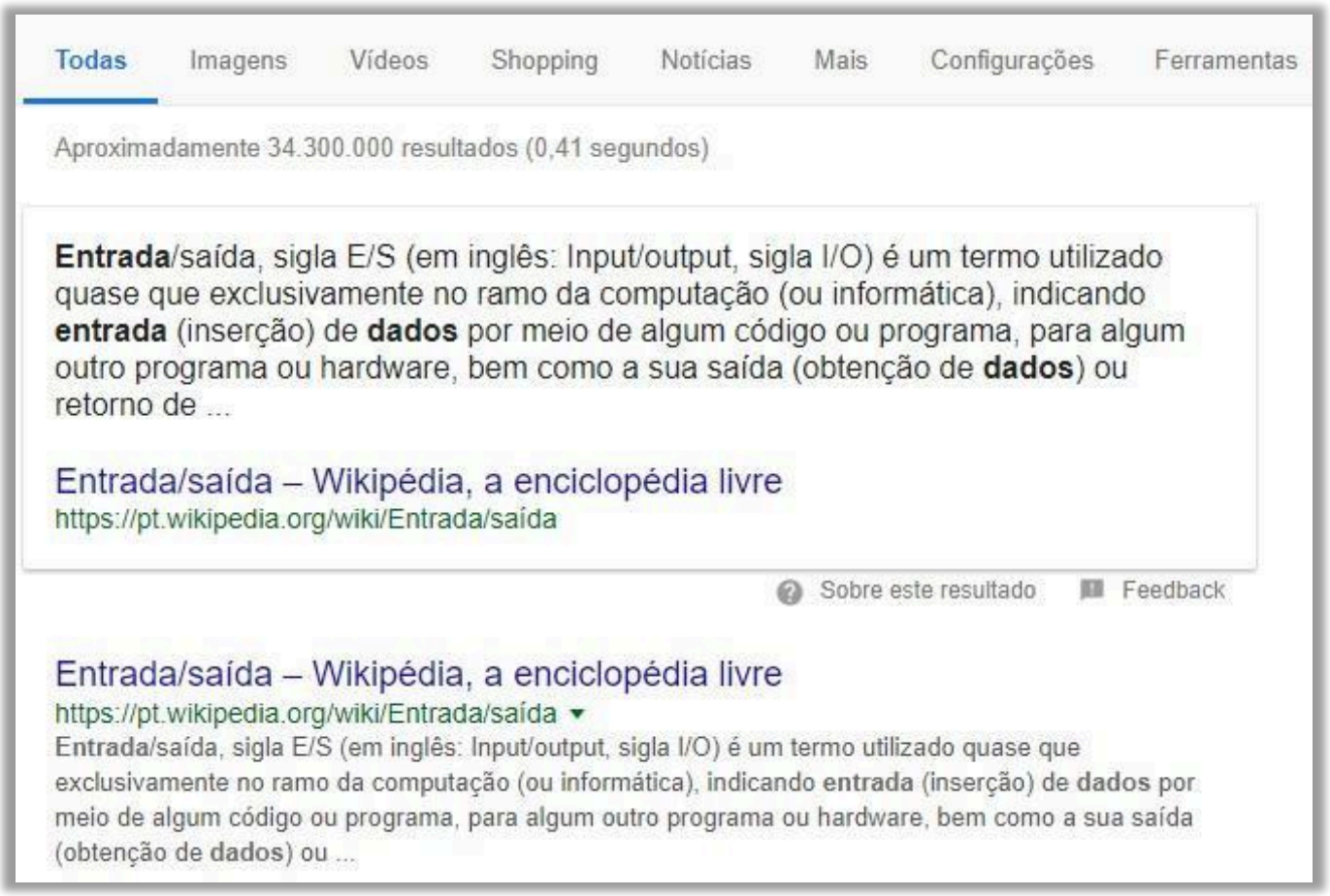
    - **Inteiro:** também conhecido como Integer, são similares aos números inteiros da matemática, i.e., sem parte fracionária. Podem ser positivos, negativos ou nulos. Ex: -2% de crescimento do PIB; 174 km de distância; 0 ºC de Temperatura; etc.
    - **Real:** também conhecido como Float (Ponto Flutuante), são similares aos números reais da matemática, i.e., possuem parte fracionária. Ex: 3,141592 é a constante de PI; 9,81 m/s² de Aceleração Gravitacional; raiz quadrada de 7; etc.
    - **Caractere:** também conhecido como Literal ou Char, são representações de letras, dígitos e símbolos. Quando colocadas em conjunto, formam um tipo estruturado chamado String ou Cadeia de Caracteres. Ex: ‘a’, ‘$’, ‘5’, ‘D’, etc.
    - **Lógico:** também conhecido como Boolean, são representações de valores lógicos – verdadeiro/falso, ligado/desligado, sim/não, etc. São extremamente importantes na programação, principalmente na verificação de condições.

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

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

- **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).



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 !=
Maior que	>
Menor que	<
Maior ou igual a	>=

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

Menor ou igual a	<=

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

(CESPE/CEBRASPE – 2019 – TJ-AM) A respeito de lógica, estrutura e linguagem de programação, julgue o item seguinte:

Os operadores lógicos e e ou possuem, respectivamente, as funções de conjunção e disjunção.

Comentários: Esta é a definição dos operadores lógicos e e ou.

Uma conjunção lógica ocorre quando os dois operadores são verdadeiros. Ou seja, um e.

Por outro lado, a disjunção lógica ocorre quando pelo menos um dos dois operadores são verdadeiros. Ou seja, um ou. (Gabarito: Correto)

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.

(CESPE/CEBRASPE – 2018 – ABIN) Julgue o item seguinte a respeito da construção de algoritmos, dos conceitos de variáveis e de bloco de comandos e das estruturas de controle.

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

Comentários: De fato, na lógica de programação, um bloco de comando é definido como um conjunto de ações para determinada função, mas seus delimitares são BEGIN e END; ou INÍCIO e FIM. (Gabarito: Errado)

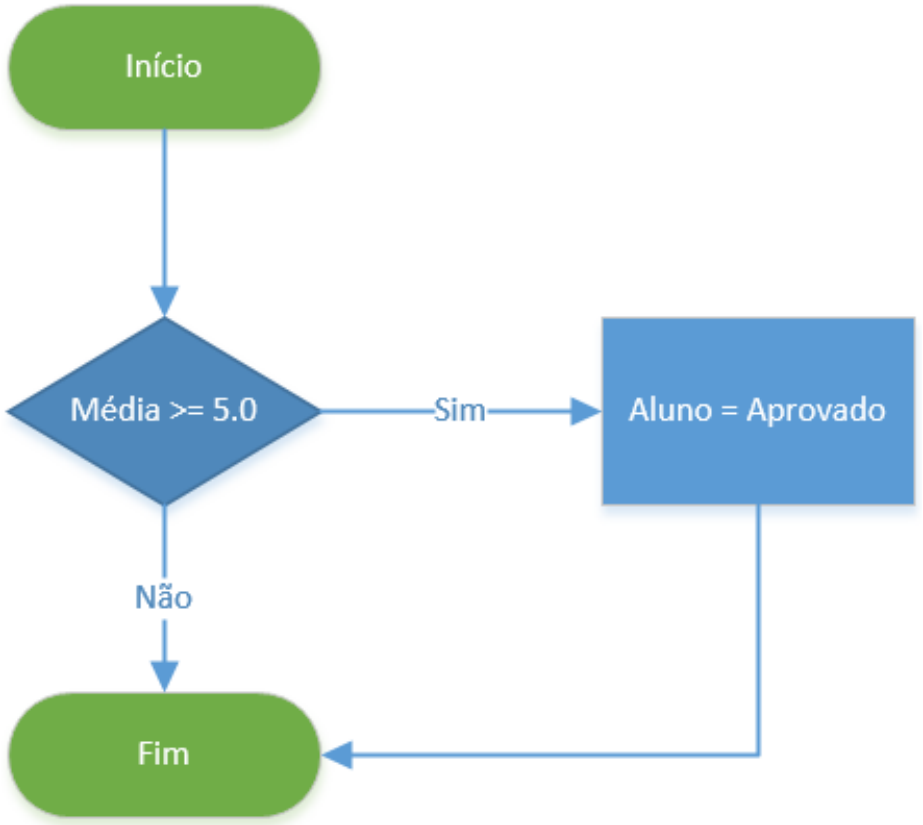
Estruturas de Controle: Decisão e Repetição

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
```



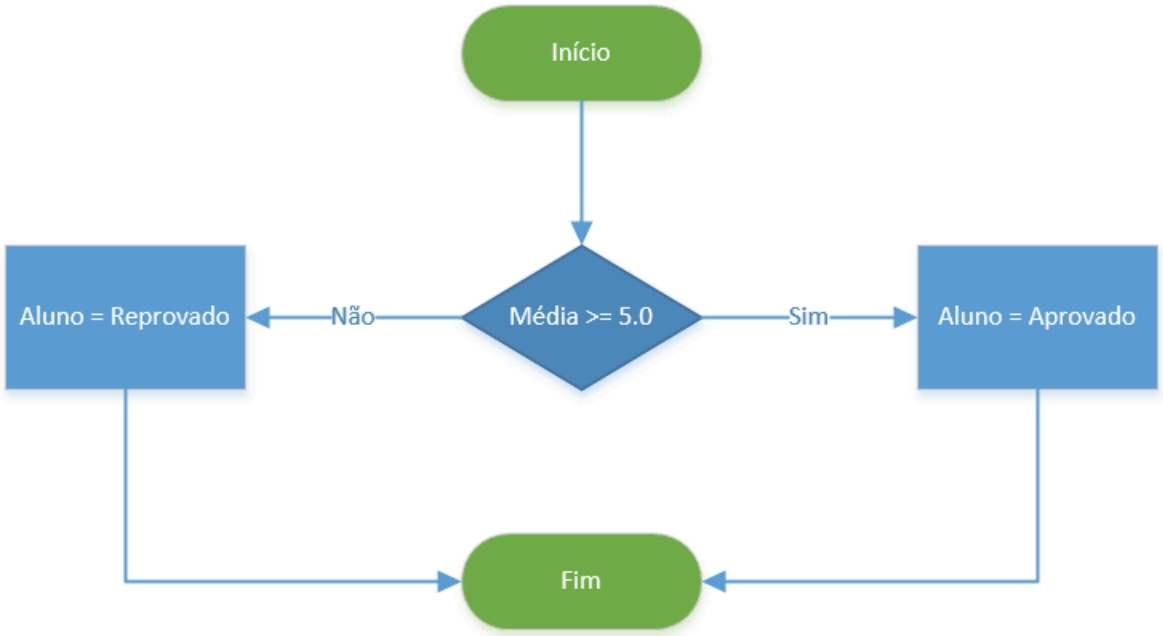
O primeiro caso, e mais simples, é uma estrutura de decisão com um teste **(Média >= 5.0).** O programa avalia se a variável Média tem o valor maior ou igual a 5.0, no caso de o teste ser verdadeiro a instrução "Aluno = Aprovado" é executada. Em caso negativo, o programa pula a instrução "Aluno = Aprovado"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

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
```

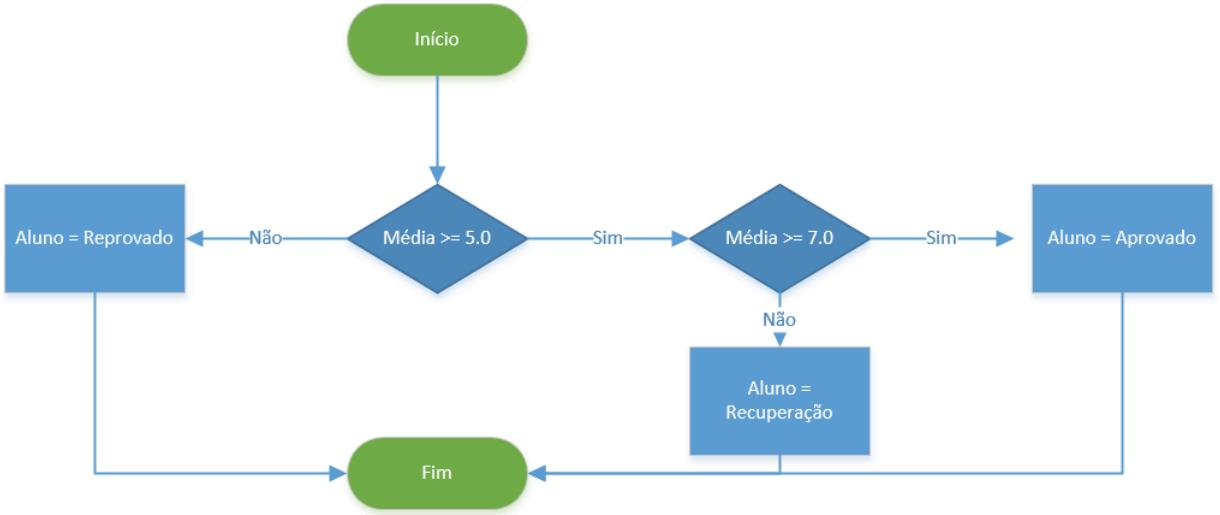


O segundo caso é só um pouquinho mais complexo, onde temos dois fluxos possíveis, um que passa pelo comando "Aluno = Aprovado" e outro que passa pelo "Aluno = Reprovado". Importante ressaltar que, nesse caso, sempre alguma dessas instruções serão executadas, pois, ou Média é maior ou igual a 5.0 ou é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

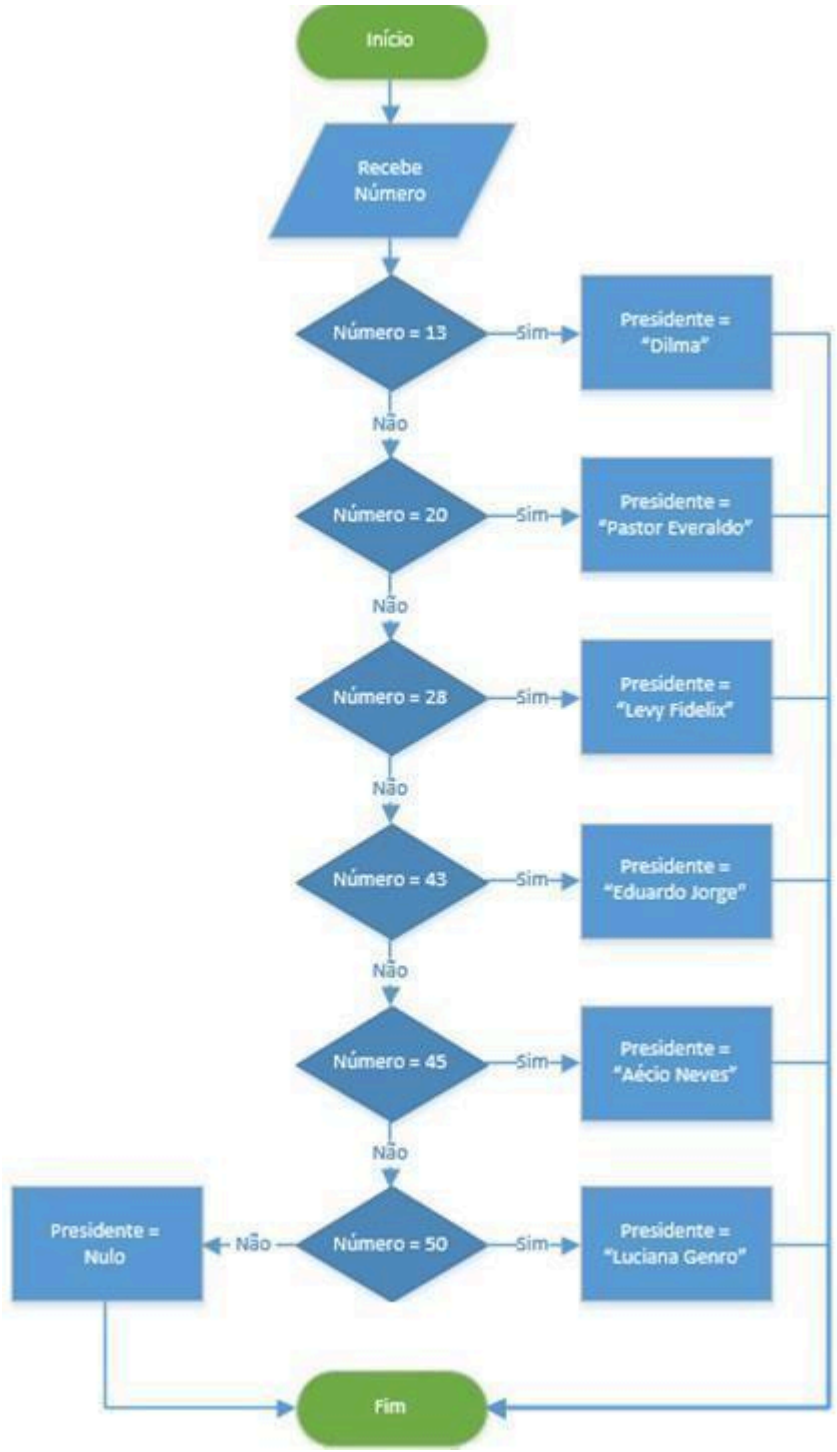
CASO 4:

```
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.

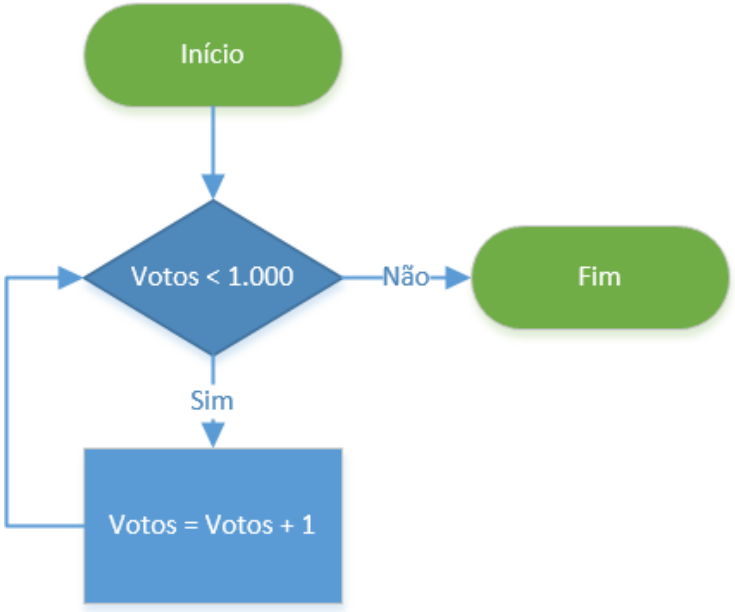
Temos, então, quatro Estruturas de Decisão (Seleção): Se-Então, Se-Então-Senão, Se-Então (Aninhados) e Caso-Selecione.3

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...



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:

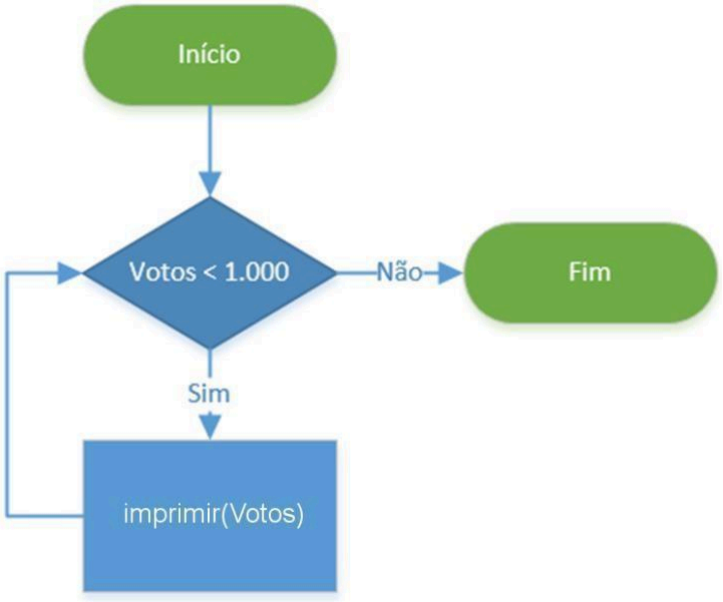
```
Votos = 500

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
```



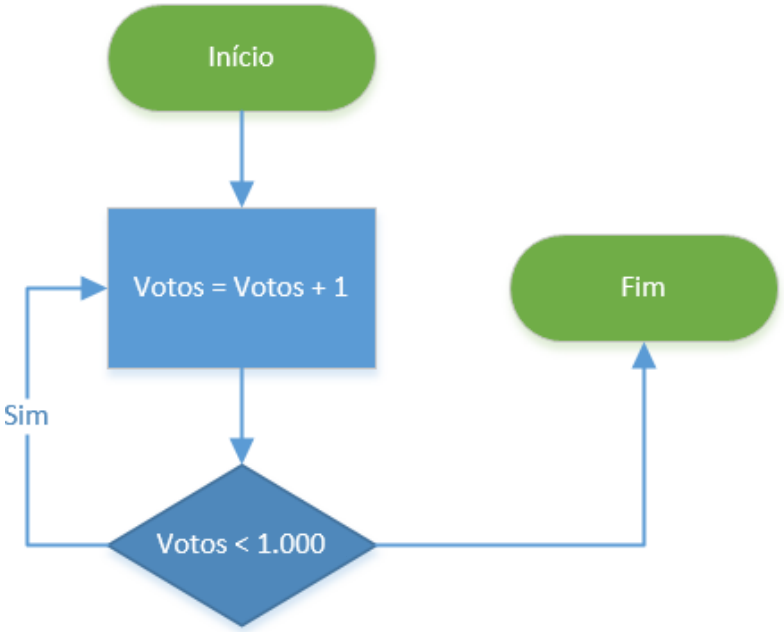
Esse tipo de estrutura é um pouco diferente da primeira, já que define de antemão quantas vezes será repetida as instruções dentro do bloco. No nosso exemplo, a instrução "imprimir (Votos)" será executada 1000 vezes, pois assim determina a expressão "Para Votos de 1 até 1000 Faça". Essa expressão é simplificada,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

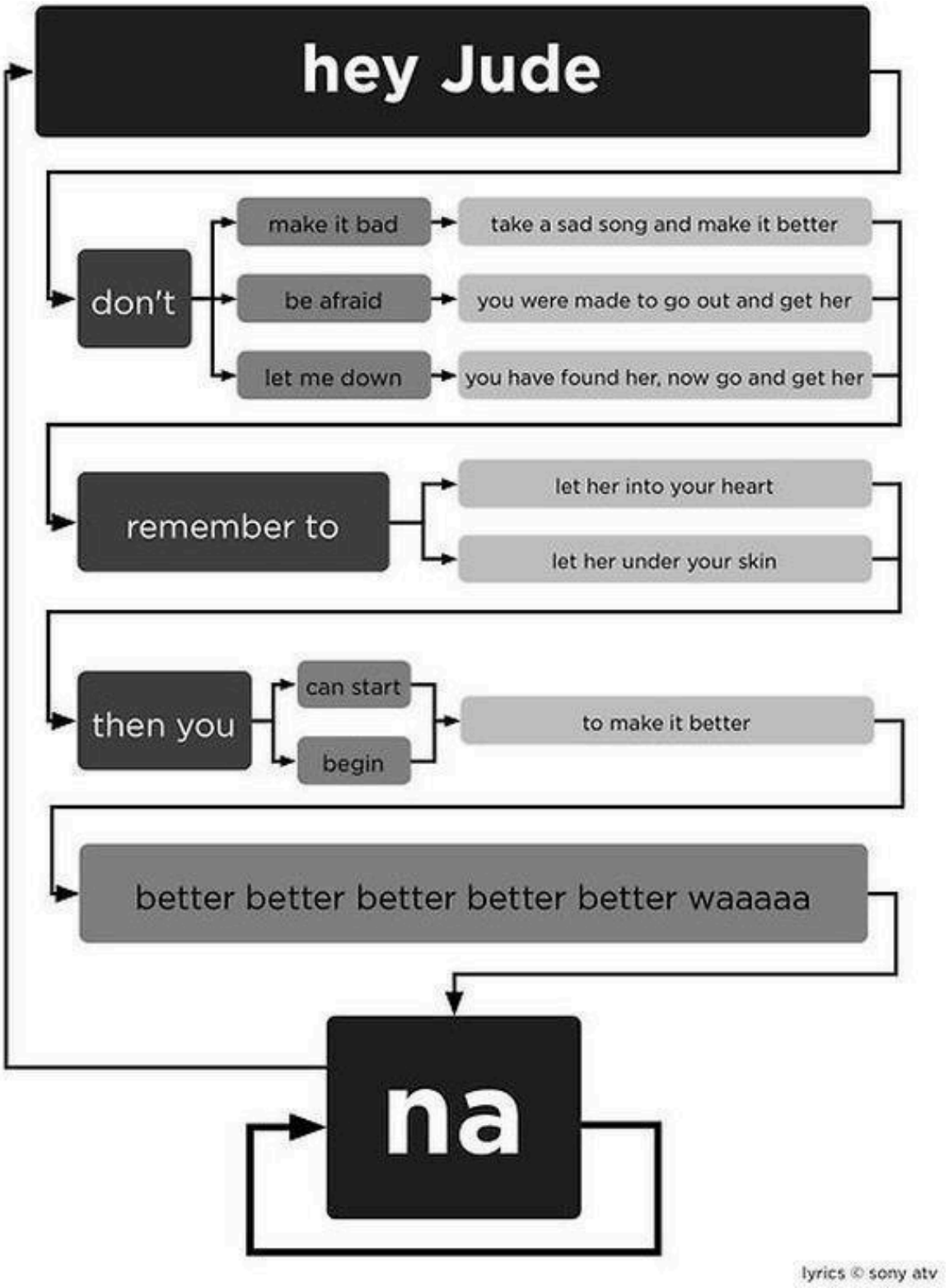
```
01 estrofe = 0
02
03 Enquanto (estrofe < 3) Repetir
04   estrofe == estrofe + 1
05   imprimir("Hey Jude, don't ")
06
07   Se (estrofe == 1) Então
08     imprimir("make it bad")
09     imprimir("take a sad song and make it better")
10   Senão Se (estrofe == 2) Então
11     imprimir("be afraid")
12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

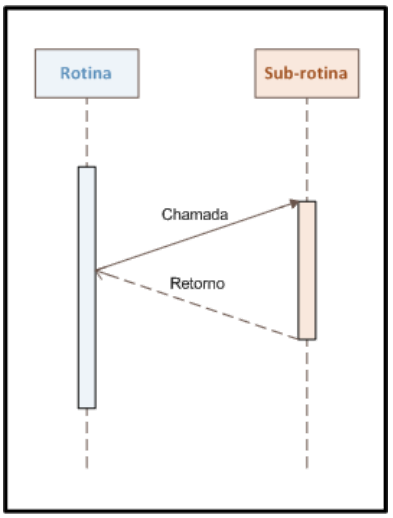

hey Jude
don't
make it bad
be afraid
let me down
take a sad song and make it better
you were made to go out and get her
you have found her, now go and get her
remember to
let her into your heart
let her under your skin
then you
can start
begin
to make it better
better better better better better waaaaa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

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?

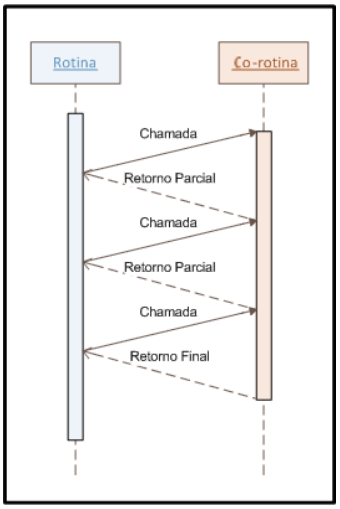
Quando uma sub-rotina é chamada, ocorre um desvio do programa para o ponto de entrada, e quando a sub-rotina atinge seu ponto de saída, o programa desaloca a sub-rotina, e volta para o mesmo ponto de onde tinha saído. Vejam a imagem ao lado!

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.

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 ' **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

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.

(CESPE / CEBRASPE – 2022 – BANRISUL)

Julgue o próximo item, a respeito de lógica de programação.

"Os laços usados em estruturas de repetição e teste podem ser feitos por meio de comandos como enquanto e repita."

Comentários: Os comandos enquanto e repita, de fato, são estruturas de repetição. Entretanto, não são estruturas de teste. (Gabarito: Errado)

(CESPE / CEBRASPE – 2022 – BANRISUL)

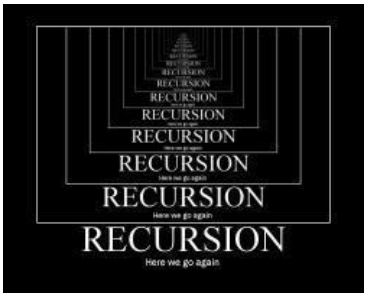
Julgue o próximo item, a respeito de lógica de programação.

"As estruturas se e senão são estruturas de repetição utilizadas nas situações em que, caso determinada condição seja alcançada, um comando é realizado, caso contrário, outro comando é executado."

Comentários: As estruturas se e senão são estruturas de seleção ou de decisão, e não estruturas de repetição. Fora este aspecto, o restante da afirmação está correta. (Gabarito: Errado)

Recursividade

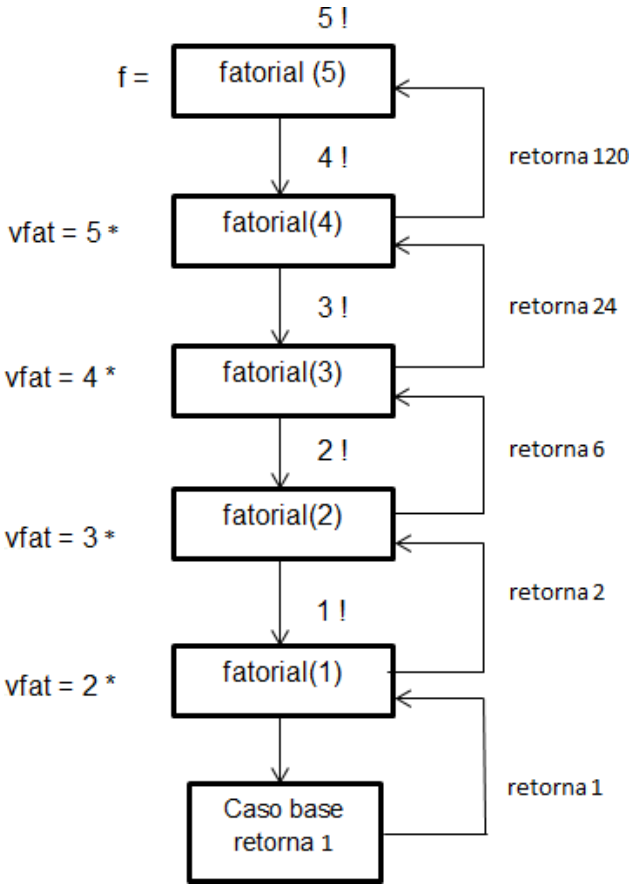
Por fim, vamos falar sobre recursividade! O que é isso, professor? Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por meio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

(FCC – 2010 – TJ-PA)

Considere a seguinte e somente a seguinte situação: Se um procedimento Px contiver uma referência a um outro procedimento Py que por sua vez contém uma referência direta ou indireta a Px, então

a) Px é subconjunto de Py.

b) Px é categorizado como indiretamente recursivo.

c) Py é subconjunto de Px.

d) Py é categorizado como diretamente recursivo.
e) Py é categorizado como indiretamente recursivo.

Comentários: Trata-se de uma questão sobre recursividade, pois os procedimentos possuem referência entre si.
Vamos comentar item a item.

a) Px é subconjunto de Py.

Px não é subconjunto de Py, já que os procedimentos elas se referenciam entre si. Falso.

b) Px é categorizado como indiretamente recursivo.

Esta é a correta.
Um procedimento X é considerado indiretamente recursivo se tiver referência a um outro procedimento Y, que, por sua vez, tiver uma referência direta ou indireta ao procedimento X.

c) Py é subconjunto de Px.

Py não é subconjunto de Px, já que as funções elas se referenciam entre si. Falso.

d) Py é categorizado como diretamente recursivo.

Incorreto, pois a referência a Px é direta ou indireta, e não direta.

Um procedimento X é considerado recursivo se tiver referência ao próprio procedimento X, e essa referência for explícita.

e) Py é categorizado como indiretamente recursivo.

Incorreto, pois a referência a Px é direta ou indireta, e não direta. (Gabarito: Letra B)

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).

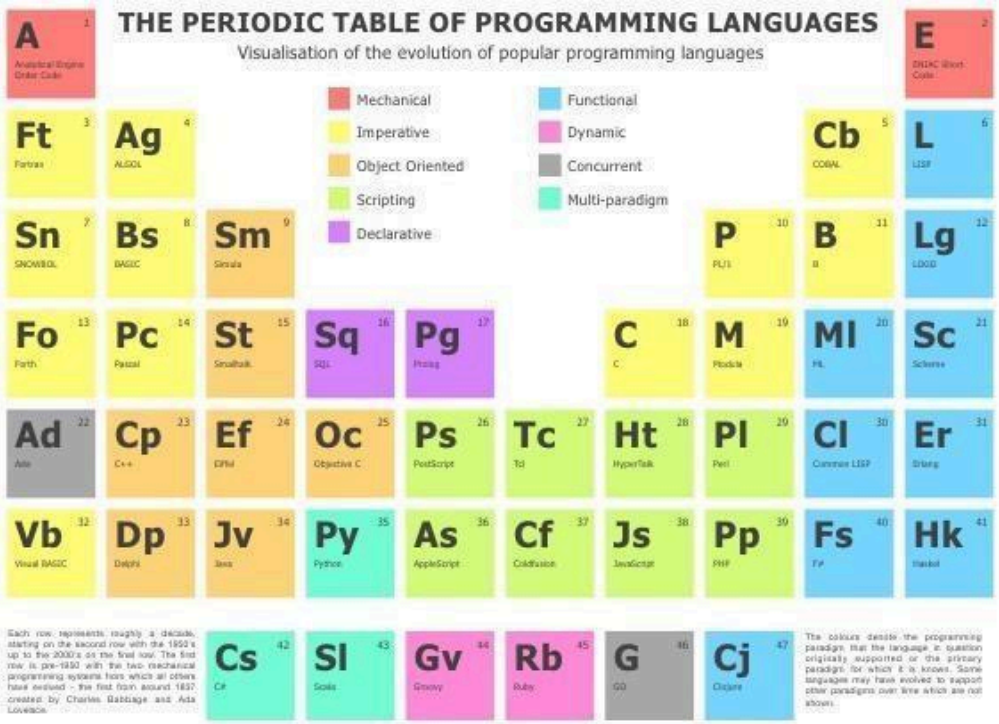
Linguagem de Alto Nível: linguagem com um nível de **abstração** relativamente **elevado,** longe do código de máquina e mais próximo à linguagem humana. Desse modo, as linguagens de alto nível não estão

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.

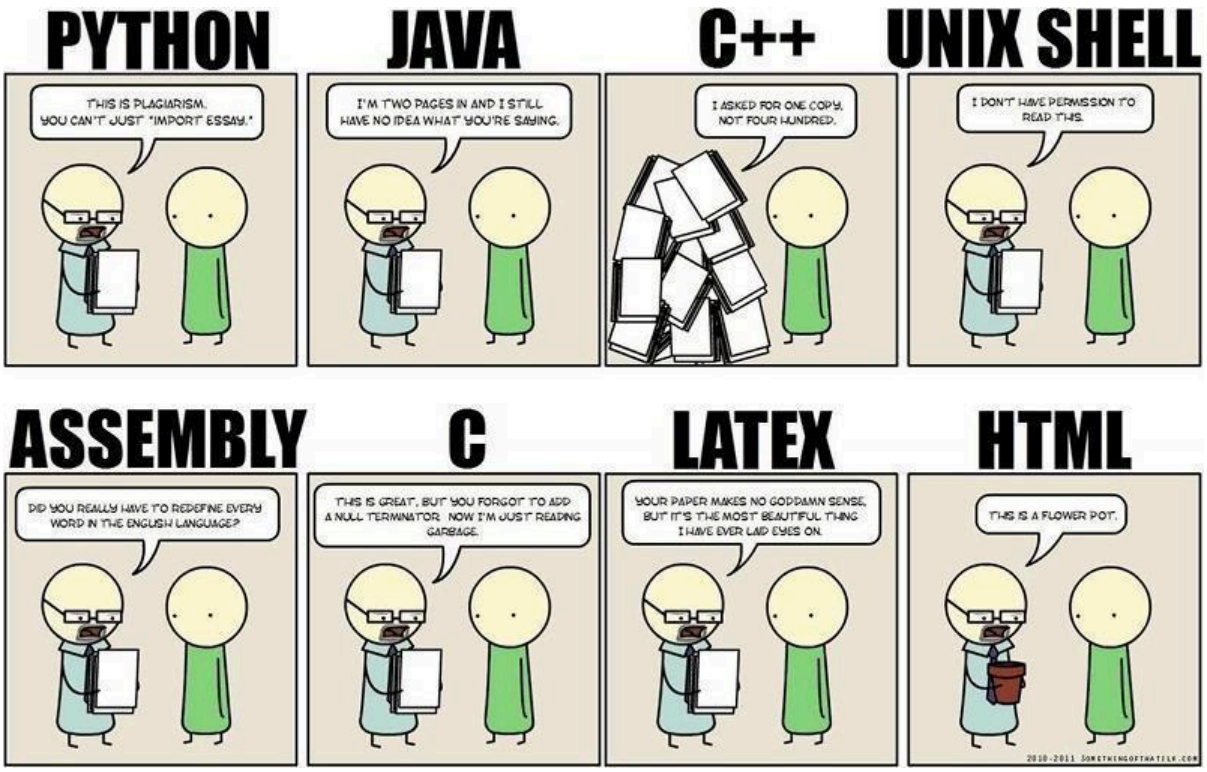
Com o objetivo de combater os problemas da programação em Assembly, John Backus criou, também na década de 1950, a linguagem FORTRAN (FORmula TRANslator), que é uma linguagem de alto nível e considerada uma das melhores da época. Ainda na mesma década, foram criados o LISP (LISt Processor) e o COBOL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

QUESTÕES COMENTADAS –

1. **(CESPE / CEBRASPE – 2022 – BANRISUL) Em um algoritmo, todo resultado de uma operação de entrada de dados é armazenado em uma posição na memória.**

Comentários:

Um algoritmo é uma sequência finita de instruções bem definidas que objetivam a resolução de um determinado problema.

O resultado de um algoritmo calculado a partir de um conjunto de dados de entrada necessariamente é armazenado em uma posição de memória para que posteriormente seja lido. **Gabarito: Certo.**

2. **(CESPE / CEBRASPE – 2022 – BANRISUL) Os laços usados em estruturas de repetição e teste podem ser feitos por meio de comandos como enquanto e repita.**

Comentários:

Os comandos **enquanto** e **repita**, de fato, são estruturas de repetição. Entretanto, não são estruturas de teste. **Gabarito: Errado.**

3. **(CESPE / CEBRASPE – 2022 – BANRISUL) O fluxograma é uma das formas de se representarem as instruções de um programa, utilizando-se de alguns comandos genéricos ou primitivos para a interpretação do algoritmo.**

Comentários:

Os comandos genéricos ou primitivos são utilizados em **pseudocódigo**, e não em **fluxogramas**.

O fluxograma é uma das formas de se representar as instruções de programa, mas utiliza-se de formas gráficas, que enfatizam os passos individuais e o fluxo de execução de um algoritmo. **Gabarito: Errado.**

4. **(CESPE / CEBRASPE – 2022 – BANRISUL) As estruturas se e senão são estruturas de repetição utilizadas nas situações em que, caso determinada condição seja alcançada, um comando é realizado, caso contrário, outro comando é executado.**

Comentários:

As estruturas se e senão são estruturas de seleção ou de decisão, e não estruturas de repetição. Fora este aspecto, o restante da afirmação está correta. **Gabarito: Errado.**

5. (CESPE/CEBRASPE – 2022 – POLITEC - RO) O autômato finito determinístico:

a) corresponde à função de transição que recebe um estado ou um símbolo de entrada que sempre retorna um conjunto de estados como resultado.
b) tem a capacidade de adivinhar algo sobre sua entrada ao testar valores.
c) pode, para cada entrada, transitar a partir do seu estado atual em um e somente um estado.
d) permite zero, uma ou n transições para os estados de entrada.
e) consegue estar em vários estados ao mesmo tempo.

Comentários:

Um autômato finito determinístico tem três características pelo seu próprio nome:

1. É um autômato.
2. É finito: ou seja, tem um estado inicial, um conjunto de estados de aceitação, um conjunto finito de estados intermediários e de símbolos de entrada.
3. É determinístico: significa que as regras de transição executadas pela função de transição entre os estados são bem determinadas. Por definição, gera-se um único ramo de computação para cada cadeia de entrada.

Vamos analisar os itens.

a) corresponde à função de transição que recebe um estado ou um símbolo de entrada que sempre retorna um conjunto de estados como resultado.

A função de transição irá retornar apenas um estado como resultado, devido ao fato do autômato finito determinístico ser determinístico. Significa que, para cada entrada, apenas transita por um conjunto de estados determinado. Falso

b) tem a capacidade de adivinhar algo sobre sua entrada ao testar valores.

Um autômato finito determinístico não adivinha informações sobre a entrada. Ele, de posse das informações de entrada, decide, por meio da função de transição, por quais estados percorrerá. Falso.

c) pode, para cada entrada, transitar a partir do seu estado atual em um e somente um estado.

Um autômato finito determinístico, pela definição de ser determinístico, somente pode transitar para um estado para cada entrada. Alternativa correta.

d) permite zero, uma ou n transições para os estados de entrada.

Um autômato finito determinístico, pela definição de ser determinístico, somente pode transitar para um estado para cada entrada. Portanto, ele não permite nem zero e nem n transições. Falso.

e) consegue estar em vários estados ao mesmo tempo. Somente um estado. Falso. **Gabarito: Letra C.**

6. (CESPE/CEBRASPE – 2022 – SEED-PR)

```
programa
{
    funcao inicio()
    {
        inteiro numero, atual = 1, valor = 1
        numero = 6
        enquanto (atual <= numero)
        {
            valor = valor * atual
            atual = atual + 1
        }
        escreva(valor)
    }
}
```

O resultado do algoritmo precedente é:

a) 2
b) 6
c) 24
d) 120
e) 720

Comentários:

A função calcula o fatorial de valor.
Vamos linha por linha:
inteiro numero, atual = 1, valor = 1

Nesta linha, são definidas 3 variáveis:

- numero, sem valor.
- atual, com valor 1
- valor, com valor 1 numero = 6

Define o valor da variável numero para 6.
enquanto (atual <= numero)
Laço de repetição com teste no início, que somente vai sair do laço quando atual <= numero for falso, ou seja, quando atual > numero.
enquanto (atual <= numero) {
valor = valor * atual

```
        atual = atual + 1
}
```

Em cada volta do laço, irá multiplicar o valor da variável valor pela variável atual. Depois, incrementa o valor da variável atual.

Na primeira volta:
Teste: 1 <= 6. Verdadeiro, então entra no laço.
valor = 1*1 = 1
atual = 1 + 1 = 2
Na segunda volta:
Teste: 2 <= 6. Verdadeiro, então entra no laço.
valor = 1*2 = 2
atual = 2 + 1 = 3

Na terceira volta:
Teste: 3 <= 6. Verdadeiro, então entra no laço.
valor = 2*3 = 6
atual = 3 + 1 = 4
Na quarta volta:
Teste: 4 <= 6. Verdadeiro, então entra no laço.
valor = 6*4 = 24
atual = 4 + 1 = 5
Na quinta volta:
Teste: 5 <= 6. Verdadeiro, então entra no laço.
valor = 24*5 = 120
atual = 5 + 1 = 6
Na sexta volta:
Teste: 6 <= 6. Verdadeiro, então entra no laço.
valor = 120*6 = 720
atual = 6 + 1 =76
Na sétima volta:
Teste: 7 <= 6. Falso, então sai no laço.

Por fim, escreve o que está na variável valor, que é 720. **Gabarito: Letra E.**

7. (CESPE / CEBRASPE – 2021 – SEED-PR) Considerando a, b, c e d como variáveis com valores iniciais iguais a 5, 7, 3 e 9, respectivamente, assinale a opção correta.

a) O resultado da expressão (a != 3 || b < 10 || c == 5) é falso.
b) O resultado da expressão (d > 8 && c == 3 || a >= 10) é verdadeiro.
c) O resultado da expressão !(d == 12 && a != 10) é falso.
d) O resultado da expressão (c == 4 || d <=6) && (a >= 5 && b !=9) || (! (a < 5)) é falso.
e) O resultado da expressão (a == 3 || b > 10 || d == 8) é verdadeiro.

Comentários:

Consideremos a = 3, b = 7, c = 3 e d = 9.

Consideremos os significados dos sinais de operação:

Operadores de comparação	Significado
!=	Diferente (≠)
==	Igual (=)
>	Maior que (>)
>=	Maior ou igual a (≥)
<	Menor que (<)
<=	Menor ou igual a (≤)

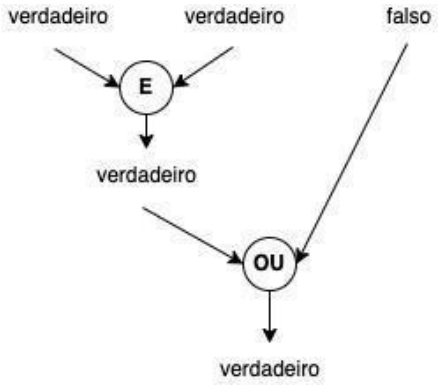
Operadores lógicos	Significado
&&	E
||	OU
!	Negação (NOT)

a) *"O resultado da expressão (a != 3 || b < 10 || c == 5) é falso."*

- a != 3: a variável **"a"**, que é **"5"**, é **diferente de "3"**, então é **verdadeiro**.
    - Pela tabela verdade, disso, já podemos deduzir que a expressão é verdadeira, pois, em uma expressão com operadores lógicos OU, se um dos elementos é verdadeiro, então a expressão inteira é verdadeira.
    - O item afirma que a expressão é falso. Então está errado.

b) "O resultado da expressão (d > 8 && c == 3 || a >= 10) é verdadeiro."

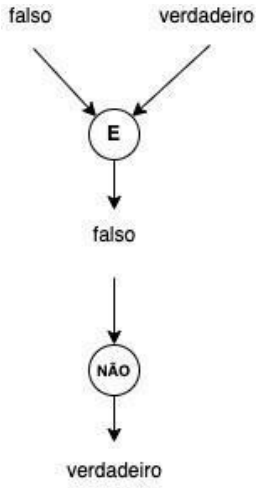
- d >8: a variável "d" é "9", o que é maior do que 8. Verdadeiro.
- c== 3: a variável "c" é "3", o que é igual a "3". Verdadeiro.
- a >= 10: a variável "a" é "3", o que não é maior ou igual a 10. Falso
- A expressão fica da seguinte maneira: (verdadeiro E verdadeiro OU falso). Pela tabela verdade:



- Portanto, este é o item correto! **Gabarito: Letra B.**

c) *"!(d == 12 && a != 10) é falso"*

- A expressão fica assim: NÃO (falso E verdadeiro). Pela tabela verdade:



- Portanto, o item está errado!

d) "(c == 4 || d <=6) && (a >= 5 && b !=9) || (! (a < 5)) é falso."

- Para resolver esse item bem rápido, observemos o último elemento da expressão:
  - !(a < 5): "a" não é menor do que 5. Então a expressão "a < 5" é falsa. Mas tem um operador de negação. Então a expressão "!(a < 5)" é verdadeira.
  - Se o último item da expressão é um "ou" e ele é verdadeiro, então a expressão é verdadeira.
- Portanto, o item está errado!

e) "O resultado da expressão (a == 3 || b > 10 || d == 8) é verdadeiro."

- Item por item:
  - a == 3: a variável "a" é "5", o que é diferente de "3". Falso.
  - b > 3: a variável "b" é "7", o que não é maior que "10". Falso.
  - d == 8: a variável "d" é "9", o que não é "8". Falso.
  - (falso OU falso OU falso) = falso.
- Portanto, o item está errado!

8. (CESPE / CEBRASPE – 2021 – SEED-PR)

```
1. var
2. valores: vetor [1..5] de inteiro
3. resultado, x: real
4.
5. inicio
6. para i de 1 ate 5 faca
7.    leia(valores)
8.    x<- x + valores
9. fimpara
10. resultado <- x / 5
11. escreva("Resultado: ", resultado)
12. fimalgoritmo
```

O resultado da lógica dos algoritmos precedente é a:

a) média dos valores da matriz vetor.
b) soma dos valores de 1 a 5, ou seja, 15.
c) média dos valores de 1 a 5, ou seja, 3.
d) soma dos valores da matriz vetor.
e) ordenação dos valores de 1 a 5.

Comentários:

Antes de mais nada, é importante observar que, no pseudocódigo, os valores do vetor não são explícitos, e, sim, pedidos para o usuário (linha 7). Portanto, os itens b), c) e e) estão eliminados.

```
6. para i de 1 ate 5 faca
7.    leia(valores)
8.    x<- x + valores
9. fimpara
```

Observa-se que, no laço acima, ele soma os valores de cada um dos itens do vetor e adiciona cada um numa variável.

```
10. resultado <- x / 5
```

Já na linha 10, ele divide o valor da soma de todos os itens do vetor por 5, que é a quantidade de itens do vetor.

Somar todos os itens de um vetor e dividir pela quantidade de itens no vetor é calcular a sua média. **Gabarito: Letra A.**

9. (CESPE / CEBRASPE – 2021 – SEED-PR)

```
1. var
2. num: inteiro
3. inicio
4. leia(num)
5. se (num % 2) =/= 0 entao
6. escreva("X")
7. senao
8. escreva("Y")
9. fimse
10. fimalgoritmo
```

Considerando-se a lógica do algoritmo anterior, é correto afirmar que, para todo valor de num (linha 4):

a) maior que 2, escreve Y.
b) menor que 2, escreve X.
c) igual a 2, escreve Y.
d) igual a 2, escreve X.
e) diferente de 2, escreve X.

Comentários:

O operador % significa “resto da divisão”, enquanto o operador =/= significa “diferente”.
O resto de uma divisão por 2 somente pode ser 0 ou 1. Em caso de números pares, o resto é 0, enquanto em números pares, o resto é 1.

Pela estrutura “se” e “então”:

- Se número for ímpar, o resto é 1, portanto entra no “se” escreve X.
- Se o número for par, resto é 0, portanto entra no “então, e escreve Y.

Analisando cada uma das alternativas:

a) maior que 2, escreve Y.
Existem números inteiros ímpares maiores que 2, e que, portanto, escrevem X. Errado.

b) menor que 2, escreve X.
Existem números inteiros pares menores do que 2, e que, portanto, escrevem Y. Errado.

c) igual a 2, escreve Y.
2 é um número par. Logo, escreve Y. Correto!

d) igual a 2, escreve X.
2 é um número par. Logo, escreve Y. Errado.

e) diferente de 2, escreve X.
Existem números inteiros ímpares diferentes de 2, e que, portanto, escrevem X. Errado. **Gabarito: Letra C.**

10. (CESPE / CEBRASPE – 2021 – SEED-PR)

```
var
valor: real
resultado: real
inicio
soma <- 0
escreva ("digite um valor")
leia (valor)
enquanto valor <> 0 faca
      resultado <- resultado + valor
   escreva ("digite um valor")
   leia (valor)
fimenquanto
escreva ("resultado: ", resultado)
fimalgoritmo
```

Considere que, na lógica do algoritmo apresentado, sejam inseridos sequencialmente os valores a seguir: 3 7 5 0 1 2

Nessa situação, o resultado da execução será

a) 5.
b) 18.
c) 375012
d) 15.
e) 16.

Comentários:

```
leia (valor)
enquanto valor <> 0 faca
      resultado <- resultado + valor
   escreva ("digite um valor")
   leia (valor)
fimenquanto
```

O laço de repetição "enquanto" vai rodar as instruções no seu interior até que a condição explicitada, no caso valor <> 0 (ou **valor diferente de 0**) ***não*** seja atendida. Daí ele sai do laço. O teste é sempre feito no início do bloco.

O trecho acima lê do usuário um valor, e vai somando esse valor na variável “resultado” para sempre... Desde que o valor seja diferente de zero. Ou seja, ele vai sair do loop se o valor for 0.
Então, o usuário digita 3, e o valor da variável resultado passa a ser 3.

Depois, pede novamente o valor do usuário, que digita 7. Ele volta para o “enquanto”, e, como esse valor não é 0, ele continua, e soma na variável resultado, que passar a ser 3 + 7 = 10.

Mais uma vez, o usuário digita um valor: 5. Ele volta para o “enquanto”, e, como esse valor ainda não é 0, ele continua, e soma na variável de resultado, que passa a ser 10 + 5 = 15.

Por fim, o usuário digita o valor 0. Ele volta para o “enquanto”. O valor é zero. Então ele sai do loop, e mostra o resultado, que é 15. **Gabarito: Letra D.**

11. (CESPE / CEBRASPE – 2021 – SEED-PR)

```
1. var
2. cont,n,resultado:inteiro
3. inicio
4. resultado <-1
5. leia(n)
6. para cont de 1 ate n passo 1 faca
7.    resultado <- resultado *cont
8. fimpara
9. escreva(resultado)
10. fimalgoritmo
```

Em relação à lógica do algoritmo precedente, caso o valor de n (linha 5) seja igual a:

a) 4, o resultado será 4.
b) 3, o resultado será 5
c) 2, o resultado será 12.
d) 6, o resultado será 30.
e) 5, o resultado será 120.

Comentários:

Este algoritmo está calculando o fatorial de n.

A variável resultado é definida inicialmente como 1.

```
6. para cont de 1 ate n passo 1 faca
7.    resultado <- resultado *cont
8. fimpara
```

Este loop roda n vezes, armazenando de 1 a 5 na variável cont a cada iteração. Além disso, ele multiplica na variável resultado o valor de cont.

Assim, se o valor de num for 4, ele vai multiplicar 1*1*2*3*4 = 24.
Essa é a definição de fatorial.
O fatorial de 5 é 1*2*3*4*5 = 120. **Gabarito: Letra E.**

12. (CESPE / CEBRASPE – 2021 – SEED-PR) Assinale a opção que apresenta a representação booleana equivalente mais simplificada para a função F(X, Y, Z) = X·Z + Z·(X' + XY).

a) Z + Y·Z
b) Z + X·Y
c) X·Z
d) X + Y·Z
e) Z

Comentários:

A questão apresenta uma expressão booleana F(X, Y, Z) = X·Z + Z·(X' + XY) e solicita a representação booleana equivalente mais simplificada. Para simplificar a expressão, é possível utilizar as propriedades da álgebra booleana, que incluem a distributiva, a associativa, a comutativa, a identidade e outras. Nessa notação, temos os seguintes operadores:

Operador	Significado
+	E
·	OU
'	Negação

Portanto, a expressão X·Z + Z·(X' + XY) pode ser traduzida como:
(X ou Z) e (Z ou (**não X e (X ou Z)**)

Para começar, podemos substituir o operador "·" por "+" e o operador "+" por "·", pois isso não afeta o resultado final. Então, a expressão fica assim: F(X, Y, Z) = XZ + Z(X' + XY).

Podemos aplicar a distributiva da seguinte forma: F(X, Y, Z) = XZ + ZX' + ZXY.

Em seguida, podemos usar a comutativa para mudar a ordem dos termos em ZXY: F(X, Y, Z) = XZ + ZX' + XYZ.

Podemos usar a distributiva novamente, desta vez para Z(X + X'): F(X, Y, Z) = XZ + ZX' + Z(X + Y).

Agora, podemos aplicar a distributiva de novo: F(X, Y, Z) = XZ + ZX' + ZX + ZY.

Podemos usar a associativa para agrupar os termos em Z: F(X, Y, Z) = XZ + ZX + ZX' + ZY.

Podemos usar a comutativa novamente para mudar a ordem dos termos em ZX': F(X, Y, Z) = XZ + ZX + X'Z + ZY.

Finalmente, podemos usar a distributiva mais uma vez: F(X, Y, Z) = Z(X + X') + XZ + YZ.

Como X + X' é sempre verdadeiro (pois é a negação de X ou a negação de não X, que são complementares), podemos simplificar a expressão para F(X, Y, Z) = Z + XZ + YZ.

Essa expressão representa a função F(X, Y, Z) de forma equivalente e mais simplificada, e a opção correta é a letra E, que apresenta apenas o termo Z.

Gabarito: Letra E.

13. (CESPE / CEBRASPE – 2021 – SEED-PR) (Adaptada) O fato de o complemento do produto ser igual à soma dos complementos, ou seja, (A·B) ' = A' + B', é justificado:

a) pela lei comutativa.
b) pela lei associativa.
c) pela propriedade distributiva.
d) pelo teorema de Morgan.
e) pelo teorema da identidade.

Comentários:

O teorema de Morgan define justamente a seguinte simplificação:
não (A ou B) = (não A e não B)
Gabarito: Letra D.

14. (CESPE / CEBRASPE – 2021 – PG-DF) O resultado do pseudocódigo precedente será 120.

```
função avaliar( a, b )
início
        ma <-a;
        se (ma < b) então ma <- b;
        me <-a;
        se (me > b) então me <- b;
    resultado <- ( ma % me );
    se (resultado = 0)
        então retorne me
        senão avaliar(me, ma)
fim

escreva avaliar (120,30);
```

Comentários:

A última linha chama a função avaliar definindo os parâmetros a = 120 e b = 30.
ma <-a;

Atribui à variável ma o valor de a, ou seja, 120.
se (ma < b) então ma <- b;

Realizada uma condicional. Como ma = 120 e b = 30, 120 < 30 é falso. Portanto, não entra na condicional.
me <-a;
Atribui à variável me o valor de a, ou seja, 120.
se (me > b) então me <- b;
Realiza uma condicional. Como me = 120 e b = 30, 120 > 30 é verdadeiro. Portanto, entra na condicional, e atribui a me o valor de b, ou seja, me <- 30.
resultado <- (ma % me);

O operador % significa resto da divisão. Portanto, atribui a resultado o resto da divisão entre ma e me. O resto da divisão de 120 por 30 é 0. Portanto, resultado <- 0
se (resultado = 0) então retorne me senão avaliar(me, ma

Realiza uma condicional. Como o resultado é 0, então resultado = 0 é verdadeiro. Portanto, **retorna o valor de me, que é 30**.

Como a questão afirma que o resultado é 120, a resposta é Errado. **Gabarito: Errado.**

15. (CESPE / CEBRASPE – 2021 – PG-DF) O resultado do pseudocódigo apresentado será 6.

```
início
   v <- vetor (2, 4, 6, 8 , 10 ,12 );
    escreva ( v[0] + " " +v[1] );
fim;
```

Comentários:

v <- vetor (2, 4, 6, 8 , 10 ,12);
Ao inicializar o vetor, seus valores são definidos da seguinte forma:
v[0] <- 2
v[1] <- 4
v[2] <- 6
v[3] <- 8
v[4] <- 10
v[5] <- 12
escreva (v[0] + " " +v[1]);

Nesta linha, ele escreve v[0] + " " +v[1], ou seja, 2 + " " + 4. Como há uma string com um único caractere no meio da soma, deduzimos que o que está ocorrendo ao acionar o operador + não é uma operação de soma aritmética, mas, sim, uma concatenação de texto.

Portanto, concatena-se o texto "2"com o texto " " e, por fim, com o texto "4". O resultado é um texto "2 4".
Gabarito: Errado.

16. (CESPE / CEBRASPE – 2021 – SEED-PR) (4 > 2) xor (5 = 3) and (4 > 2) or (5 = 5)

Assinale a opção que apresenta o resultado da expressão anterior.

a) 2
b) 3
c) 4
d) Falso
e) Verdadeiro

Comentários:

xor significa operador “ou exclusivo”.

Consideremos as tabelas-verdade:

A	B	A e B
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

A	B	A ou B
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

A	B	A xor B
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

Analisemos individualmente as preposições entre parênteses.

- 4 > 2: 4 é maior do que 2. Verdadeiro.
- 5 = 3: 5 é igual a 3. Falso.
- 4 > 2: 4 é maior que 2. Verdadeiro.
- 5 = 5: 5 é igual a 5. Verdadeiro.

Agora, vamos substituir na expressão:
(4 > 2) xor (5 = 3) and (4 > 2) or (5 = 5)

Vamos simplificando o sublinhado:
verdadeiro xor falso and verdadeiro or verdadeiro
verdadeiro xor falso é verdadeiro.
verdadeiro and verdadeiro or verdadeiro

verdadeiro

Gabarito: Letra E.

17. (CESPE/CEBRASPE – 2021 – SEED-PR) Assinale a opção que apresenta o resultado do algoritmo apresentado.

```
programa {
    funcao inicio() {
        inteiro vetor[] = { 81, 37, 51, 77, 19 }
        inteiro naosei
        logico achou = verdadeiro
        enquanto (achou)
        {
            achou=falso
            para (inteiro i = 0; i <4; i++)
            {
                se (vetor[i] > vetor[i+1])
                {
                    naosei = vetor[i]
                    vetor[i] = vetor[i+1]
                    vetor[i+1] = naosei
                    achou = verdadeiro
                }
            }
        }
        para (inteiro i = 0; i < 5; i++)
        {
            escreva (vetor[i]+ "\n")
        }
    }
}
```

a) 81
37
51
77
19

b) 81
51
37
17
19

c) 19
37
51
77
81

d) 51
81
37
77
19

e) 19
77

37
81
51

Comentários:

O que este algoritmo faz é ordenar os números do vetor, o *Bubble Sort*. É um algoritmo clássico e a melhor forma de resolver essa questão é conhecendo-o previamente, para já responder sem precisar nem passar passo-a-passo no algoritmo. Basta informar a opção em que o vetor está ordenado, que é a letra C.

De toda forma, vamos mostrar o passo-a-passo deste algoritmo:
Inicializa o vetor = [81,37,51,77,19]
Declara a variável auxiliar naosei.
Inicializa a variável achou = true
Testa o "enquanto (achou)".
Como achou = true, então entra no "enquanto".
Define achou = false.
Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.
Para i = 0:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 81 > 37
naosei = vetor[i] = vetor[0] = 81
Agora, troca os valores de vetor[0] e vetor[1].
vetor[0] = vetor[i + 1] = vetor[1] = 37
vetor[1] = naosei = 81
Define achou = true.
Valores do vetor: [37,81,51,77,19].

Para i = 1:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 81 > 51
naosei = vetor[i] = vetor[1] = 81
Agora, troca os valores de vetor[1] e vetor[2].
vetor[1] = vetor[i + 1] = vetor[2] = 51
vetor[2] = naosei = 81
Define achou = true.
Valores do vetor: [37,51,81,77,19].

Para i = 2:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 81 > 77
naosei = vetor[i] = vetor[2] = 81
Agora, troca os valores de vetor[2] e vetor[3].
vetor[2] = vetor[i + 1] = vetor[3] = 77
vetor[3] = naosei = 81
Define achou = true.
Valores do vetor: [37,51,77,81,19].

Para i = 3:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 81 > 19
naosei = vetor[i] = vetor[3] = 81
Agora, troca os valores de vetor[3] e vetor[4].
vetor[3] = vetor[i + 1] = vetor[4] = 19
vetor[4] = naosei = 81
Define achou = true.
Valores do vetor: [37,51,77,19,81].

Volta para o enquanto.
Como achou = true, então entra no "enquanto".
Define achou = false.
Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.
Para i = 0:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 37 > 51
Falso. Não entra no if.
Valores do vetor: [37,51,77,19,81].

Para i = 1:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 51 > 77
Falso. Não entra no if.
Valores do vetor: [37,51,77,19,81].

Para i = 2:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 77 > 19
naosei = vetor[i] = vetor[2] = 77
Agora, troca os valores de vetor[2] e vetor[3].
vetor[2] = vetor[i + 1] = vetor[3] = 19
vetor[3] = naosei = 77
Define achou = true.
Valores do vetor: [37,51,19,77,81].

Para i = 3:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81
Falso. Não entra no if.
Valores do vetor: [37,51,19,77,81].

Volta para o enquanto.
Como achou = true, então entra no "enquanto".
Define achou = false.
Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.
Para i = 0:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 37 > 51
Falso. Não entra no if.
Valores do vetor: [37,51,19,77,81].

Para i = 1:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 51 > 19

naosei = vetor[i] = vetor[1] = 51
Agora, troca os valores de vetor[1] e vetor[2].
vetor[1] = vetor[i + 1] = vetor[2] = 19
vetor[2] = naosei = 51
Define achou = true.
Valores do vetor: [37,19,51,77,81].

Para i = 2:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 51 > 77
Falso. Não entra no if.
Valores do vetor: [37,19,51,77,81].

Para i = 3:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81
Falso. Não entra no if.
Valores do vetor: [37,19,51,77,81].

Volta para o enquanto.
Como achou = true, então entra no "enquanto".
Define achou = false.
Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.
Para i = 0:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 37 > 19
naosei = vetor[i] = vetor[0] = 37
Agora, troca os valores de vetor[0] e vetor[1].
vetor[0] = vetor[i + 1] = vetor[1] = 19
vetor[1] = naosei = 37
Define achou = true.
Valores do vetor: [19,37,51,77,81].

Para i = 1:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 37 > 51
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Para i = 2:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 51 > 77
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Para i = 3:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Volta para o enquanto.
Como achou = true, então entra no "enquanto".

Define achou = false.
Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.
Para i = 0:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 19 > 37
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Para i = 1:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 37 > 51
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Para i = 2:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 51 > 77
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Para i = 3:
Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81
Falso. Não entra no if.
Valores do vetor: [19,37,51,77,81].

Volta para o enquanto.
Como achou = false, então sai do "enquanto".
Valor final do vetor = [19,37,51,77,81].

Gabarito: C

18. (CESPE/CEBRASPE – 2020 – TJPA) Assinale a opção que apresenta o comando que tem a função de implementar desvios incondicionais no programa, mas que é de uso proibido na programação estruturada.

a) IF-THEN-ELSE
b) CASE
c) GOTO
d) WHILE
e) REPEAT

Comentários:

Talvez a melhor palavra para a função GOTO não seja “proibido”, mas, sim, “altamente não recomendada” em se tratando de programação estruturada. Como nenhuma das outras opções corresponde a uma instrução cujo uso é desencorajado, a resposta correta somente pode ser, por eliminação, o GOTO.

O comando GOTO permite que o fluxo salte para um outro ponto qualquer do algoritmo, marcado com o comando LABEL.

Seu uso indiscriminado pode gerar problemas de legibilidade no programa, dificultando, especialmente, a sua compreensão, e, consequentemente, sua manutenibilidade.

Portanto, seu uso deve ser evitado. **Gabarito: Letra C.**

19. (CESPE/CEBRASPE – 2019 – TJ-AM) Os operadores lógicos 'e' e 'ou' possuem, respectivamente, as funções de conjunção e disjunção.

Comentários:

Esta é a definição dos operadores lógicos e e ou.

Uma conjunção lógica ocorre quando os dois operadores são verdadeiros. Ou seja, um e.

Por outro lado, a disjunção lógica ocorre quando pelo menos um dos dois operadores são verdadeiros. Ou seja, um ou. **Gabarito: Correto.**

20. (CESPE / CEBRASPE – 2019 – TJ-AM]

As variáveis A e B estão definidas no programa TROCA_VALORES com escopo global, e a variável Y está definida com escopo local na área de dados da memória; dessa forma, as variáveis A e B somente são visíveis quando a sub-rotina TROCA é executada.

```
programa TROCA_VALORES
var
  A, B : inteiro
procedimento TROCA
var
     Y : inteiro
inicio
     Y <- A
     A <- B
     B <- Y
fim
inicio
  leia A, B
  TROCA
  escreva A, B
fim
```

Comentários:

Como as variáveis A e B têm escopo global, elas são visíveis em qualquer ponto do algoritmo. **Gabarito: Errado.**

21. (CESPE / CEBRASPE – 2019 – TJ-AM]

Se as variáveis A e B tivessem sido definidas novamente dentro da sub-rotina TROCA, elas seriam novas variáveis e teriam escopo global para a sub-rotina TROCA

```
programa TROCA_VALORES
var
  A, B : inteiro
procedimento TROCA
var
     Y : inteiro
inicio
     Y <- A
     A <- B
     B <- Y
fim
inicio
  leia A, B
  TROCA
  escreva A, B
fim
```

Comentários:

Se as variáveis A e B tivessem sido definidas novamente dentro da sub-rotina TROCA, elas, de fato, seriam novas variáveis, mas teriam um escopo **local**, e não **global**, para a sub-rotina TROCA. **Gabarito: Errado.**

22. (CESPE/CEBRASPE – 2018 – ABIN) Julgue o item subsequente, relativo à lógica de programação.

A expressão a seguir especifica que: 1 será adicionado a x, se x for maior que 0; 1 será subtraído de x, se x for menor que 0; o valor de x será mantido, se x for igual a zero.
Se (x > 0) então x++; senão if (x < 0) x-- ;

Comentários:

O operador ++ significa operador incremental. x++ é o mesmo que x = x + 1.
Já o operador – significa operador decremental. x-- é o mesmo que x = x – 1.

Portanto, o código pode ser lido da seguinte forma:

“Se o valor de x for maior do que 0, então adiciona 1 ao x. Senão, e se o valor de x for menor do que 0, então

No caso de x ser igual a zero, nada ocorre.

Isso é uma forma diferente de falar a mesma coisa que está sendo dita no enunciado. Portanto, está correto.
Gabarito: Correto.

23. (CESPE/CEBRASPE – 2018 – ABIN) Na lógica de programação, um bloco de comando é definido como um conjunto de ações para determinada função e tem como delimitadores as palavras reservadas INPUT e OUTPUT.

Comentários:

De fato, na lógica de programação, um bloco de comando é definido como um conjunto de ações para determinada função, mas seus delimitares são BEGIN e END; ou INICIO e FIM. **Gabarito: Errado.**

24. (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:

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**

25. (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**

26. (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**

27. (CESPE/CEBRASPE – 2016 – TCE/PA) A passagem de parâmetro em uma rotina pode ocorrer de duas maneiras: por valor ou por referência. Em se tratando da passagem por valor, alteram-se os valores dos parâmetros que foram passados para a função.

Comentários:

As passagens de parâmetro em funções ocorrem de duas maneiras:

- Por valor: os valores das variáveis usadas como parâmetros passados para a função não são alterados.
- Por referência: os valores das variáveis usadas como parâmetros passados para a função são alterados.

O item diz que, na passagem por valor, os valores se alteram. Na verdade, isso ocorre na passagem por referência. **Gabarito: Errado.**

28. (CESPE/CEBRASPE – 2016 – TCE/PA) Em se tratando de linguagens procedimentais, os dados são globais e, portanto, acessíveis a todos os procedimentos.

Comentários:

Os dados das variáveis declaradas vão atender ao seu escopo, delimitado por um Início e Fim, ou *Begin* e *End*. Dessa forma, os dados somente serão globais se forem declaradas em escopo global; do contrário, serão restritos aos escopos dos procedimentos. **Gabarito: Errado.**

29. (CESPE/CEBRASPE – 2015 – TER-GO) Comumente usados em fluxogramas representativos de sistemas, os símbolos abaixo correspondem, respectivamente, a dados armazenados, processo, documento e entrada manual.

Comentários:

Os símbolos em questão significam, respectivamente: dados externos, documento, processo e entrada manual. Portanto, está errado. **Gabarito: Errado.**

30. (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.

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**

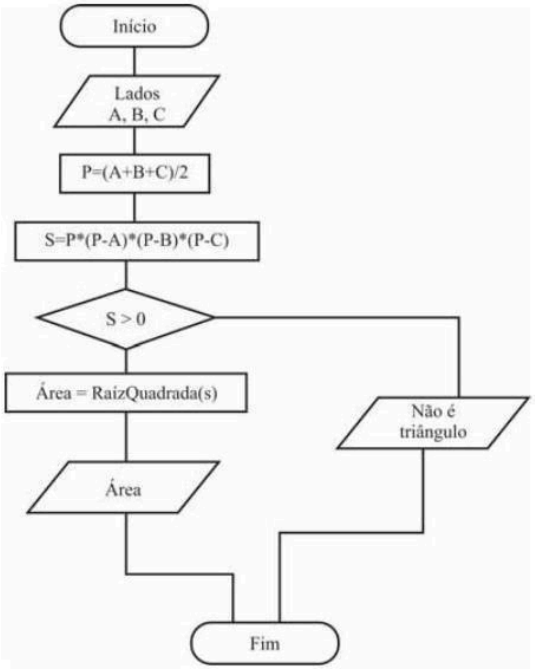
31. (CESPE - 2013 – CPRM - Analista de Sistemas) Na implementação de recursividade, uma das soluções para que se evite o fenômeno de terminação do programa – que possibilita a ocorrência de um looping infinito – é definir uma função ou condição de terminação das repetições.

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**

32. (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**

33. (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**

34. (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.

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**

35. (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**

36. (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**

37. (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**

QUESTÕES COMENTADAS – FCC

1. (FCC – 2022 – TST) Considere o trecho de um algoritmo em pseudocódigo que mostra comandos condicionais (se) aninhados com início e fim delimitados por { }:

```
se  (B1)
então     {     Comando1
                Comando2
          }
senão     {  se (B2)
             então    {   Comando3
                      }
             senão
                      {   Comando4
                      }
          }
Comando5;
```

Analisando este trecho, é correto afirmar que

a) se B1 for falso, o Comando3 e o Comando4 serão executados.
b) se B2 for verdadeiro, somente o Comando3 será executado.
c) o Comando5 poderá ser o único comando a ser executado.
d) o Comando4 sempre será executado, uma vez que o comando B2 é sempre falso.
e) o Comando5 sempre será executado.

Comentários:

Analisemos item a item.

a) se B1 for falso, o Comando3 e o Comando4 serão executados.

Se B1 for falso, verificará B2, e daí se B2 for verdadeiro, será executado o Comando3; se B2 for falso, será executado o Comando4. Falso.

b) se B2 for verdadeiro, somente o Comando3 será executado.

Se B2 for verdadeiro, mas B1 for verdadeiro, ele entrará na primeira condicional, e executará Comando1 e Comando2; não executará Comando3.

c) o Comando5 poderá ser o único comando a ser executado.

Se B1 for verdadeiro, os comandos Comando1 e Comando2 serão executados. Se B1 for falso, daí o Comando3 ou o Comando4 serão executados, a depender do valor de B2. O Comando5 sempre será executado, mas nunca sozinho. Falso.

d) o Comando4 sempre será executado, uma vez que o comando B2 é sempre falso.

B2 pode ser verdadeiro ou falso. Não está explícito no programa que B2 é falso. Falso.

e) o Comando5 sempre será executado.

Sim, o Comando5 sempre será executado, pois está fora das condicionais. Gabarito: E

2. (FCC – 2022 – DPE-RS) Considere o seguinte algoritmo em pseudocódigo:

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

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

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

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

a) I deve ser preenchida com numero>=1900 e numero<=2017
b) II deve ser preenchida com leia(vet[indice])
c) I deve ser preenchida com numero<1900 ou numero>2017
d) II deve ser preenchida com imprima ("Valor valido = ", vetor[indice])
e) I deve ser preenchida com numero>=1900 ou numero<=2017

Comentários:

Analisemos item a item.

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

O contrário. Como o valor do ano deve ser entre 1900 e 2017, a condição deve ser “numero < 1900 o numero > 2017”, porque a ideia é que entre no “enquanto” somente se o valor não for válido! E que fique ali até que seja válido.

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

A ideia é que os valores sejam impressos, e não lidos. Falso.

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

Correto. Porque a condição deve ser verdadeira apenas se o valor do ano for inválido. Se a condição for "numero < 1900 ou numero > 2017", o loop while será executado apenas se o valor do ano for menor que 1900 ou maior que 2017. Isso significa que o loop while será executado até que o usuário digite um valor de ano válido, ou seja, um valor entre 1900 e 2017..

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

Não. Deve ser preenchido com imprima ("Valor valido = ", vet[indice]). O erro está no nome da variável, que é vet, e não vetor. Valso.

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

Não. Porque a condição do loop while deve ser verdadeira apenas se o valor do ano for inválido. Se a condição for "numero >= 1900 ou numero <= 2017", o loop while será executado apenas se o valor do ano for válido, ou seja, um valor entre 1900 e 2017. Isso significa que o loop while nunca será executado, pois o valor do ano sempre será válido. Gabarito: C

3. (FCC – 2019 - TRF-4) Considere o programa em pseudocódigo abaixo, que não apresenta erros.

```
        var  var1=1, var2=2: inteiro

        funcao1()
        inicio
            var var1=100, var2=100: inteiro
            imprima("Variaveis dentro da funcao1(): var1= ", var1, " var2=", var2)
        fim

        funcao2()
        inicio
           var1 = var1 +1;
           var2 = var2 +2;
           imprima("Variaveis dentro da funcao2(): var1= ", var1, " var2=", var2)
        fim

inicio
   imprima("Variaveis antes de chamar a funcao1(): var1= ", var1, " var2=", var2)
   funcao1()
   imprima("Variaveis depois de chamar a funcao1():var1= ", var1, " var2=", var2)
   funcao2()
   imprima("Variaveis depois de chamar a funcao2():var1= ", var1, " var2=", var2)
fim
```

O pseudocódigo, ao ser executado, imprimirá

a) Variaveis antes de chamar a funcao1(): var1=0 var2=0
b) Variaveis dentro da funcao1(): var1=1 var2=2
c) Variaveis dentro da funcao2(): var1=101 var2=102
d) Variaveis depois de chamar a funcao1(): var1=1 var2=2
e) Variaveis depois de chamar a funcao2(): var1=101 var2=102

Comentários:

Antes de mais nada, é importante observar que existem dois tipos de escopos de variáveis neste pseudocódigo.

Existem as variáveis globais, definidas no início do programa, e com valores var1 = 1 e var2 = 2. E existem as variáveis de dentro da funcao1, var1 = 100 e var2 = 200. Essas variáveis não são as mesmas das variáveis globais, e só são variáveis locais consideradas dentro da funcao1.

Dito isso, vamos seguir e ver o que vai imprimindo...
imprima("Variaveis antes de chamar a funcao1(): var1= ", var1, " var2=", var2)
O primeiro comando. Vai usar os valores iniciais das variáveis globais, ou seja, var1 = 1 e var2 = 2.
Assim, vai imprimir: *"Variaveis antes de chamar a funcao1(): var1= 1, var2 = 2"*
funcao1()
Chama a funcao1.
inicio
    var var1=100, var2=100: inteiro
    imprima("Variaveis dentro da funcao1(): var1= ", var1, " var2=", var2)
fim

Neste caso, irá utilizar os valores de var1 e var2 de dentro da funcao1. Ou seja, vai imprimir: "Variaveis dentro da funcao1(): var1= 100, var2 = 100".

As variáveis globais não vão ser alteradas aqui porque as variáveis usadas são as locais do escopo da funcao1.
imprima("Variaveis depois de chamar a funcao1():var1= ", var1, " var2=", var2)

Neste caso, irá utilizar novamente as variáveis globais, que não foram mudadas. Então, vai ser: "Variaveis depois de chamar a funcao1():var1= 1 var2= 2".
funcao2()
Executa a funcao2.
inicio

```
    var1 = var1 +1;
    var2 = var2 +2;
    imprima("Variaveis dentro da funcao2():
    var1= ", var1, " var2=", var2)
```

fim

Neste caso, irá utilizar as variáveis globais, já que não tem nenhuma variável de escopo declarada nesta função. A função vai alterar o valor das variáveis globais. var1 = 1 + 1 = 2, e var2 = 2 + 2 = 4. E vai imprimir o seguinte: "Variaveis dentro da funcao2(): var1= 2 var2 = 4"
imprima("Variaveis depois de chamar a funcao2():var1= ", var1, " var2=", var2)

Por fim, executa este comando usando as variáveis globais, e imprime: "Variáveis depois de chamar a funcao2(): var1 = 2 var2 = 4.

Dentre as opções, a correta é, portanto, d). Gabarito: D

4. (FCC – 2019 – AFAP) No âmbito dos sistemas de numeração computacionais, o número decimal 132 tem sua respectiva correspondência aos seguintes em binário e hexadecimal:

a) 1110 0111 e 84
b) 0100 0010 e C3.
c) 1000 0100 e 84.
d) 1110 0100 e E4.
e) 1000 0100 e 78

Comentários:

Vamos item a item.

Para converter decimal em binário, temos que dividir por 2 e pegar o resto.

Dividir por 16	Quociente	Resto
(132)/2	66	0
(66)/2	33	0
(33)/2	16	1

(2)/2	1	0

(1)/2	0	1

Agora, pegamos os valores dos restos ao contrário e juntamos: (132)10 = (1000 0100)2
Agora a gente já sabe que a resposta certa é a C, mas vamos converter hexadecimal também. No caso do hexadecimal, temos que dividir por 16, e pegar o resto. Porém, temos que saber que A = 10, B = 11, C = 12, D = 13 e E = 14.

Dividir por 16	Quociente	Resto
(132)/16	8	4
(8)/16	0	8

Agora, pegamos os valores de resto ao contrário e juntamos: $(132)_{10} = (84)_{16}$. Gabarito: C

5. (FCC – 2019 – AFAP) A soma do hexadecimal 1C5 com o binário de mais baixa ordem 1101, terá como resultado o decimal

a) 434.
b) 466.
c) 737.
d) 479.
e) 482

Comentários:

O melhor a se fazer nestes casos é converter os valores para decimal e somar normalmente.
Em caso de hexadecimal, temos que considerar A = 10, B = 11, C = 12, D = 13 e E = 14.

No caso, para converter 1C5 para decimal, temos que considerar:
1 = 1
C = 12
5 = 5

E utilizar a fórmula de conversão:

$(1C5)_{16} = (1 \times 16^2) + (12 \times 16^1) + (5 \times 16^0) = (453)_{10}$
Para decimal, também devemos aplicar a fórmula de conversão. No caso:
$(1101)_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = (13)_{10}$
Portanto, $(1C5)_{16} + (1101)_2 = (453)_{10} + (13)_{10} = (466)_{10}$
Gabarito: B

6. (FCC – 2018 – SABESP) Antes de se escrever um programa em uma linguagem de programação, uma prática recomendada é apresentar a lógica de programação usando uma pseudolinguagem. Considere o algoritmo em pseudolinguagem apresentado abaixo, em que o

```
Programa Sabesp
Var ano, n: inteiro
Início
          imprima("Digite o ano com 4 digitos: ")
          leia (ano)

          se (ano > 1949 e ano<=1999)
              então
                    n ← ano mod 100

                    I
                .....

              senão
                  imprima("Ano inválido")
          fim se
Fim
```

Um Estagiário, ao analisar o algoritmo acima, conclui corretamente que

a) o operador lógico e está errado e deve ser substituído pelo operador lógico ou na instrução se.
b) a lacuna I deve ser preenchida com a instrução imprima ("O ano é do século XX pois inicia-se com ", n).
c) a lacuna I deve ser preenchida com a instrução imprima ("Os dois últimos dígitos do ano = ", n).
d) se for fornecido 1950 para ano será impresso Ano inválido. (
e) o usuário pode digitar apenas valores de ano com 4 dígitos, positivos e menores ou iguais ao ano atual (2018).

Comentários:

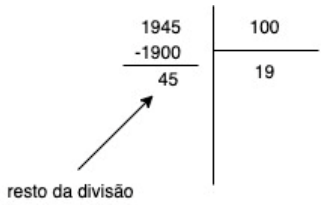
Vamos comentar item a item:

a) o operador lógico e está errado e deve ser substituído pelo operador lógico ou na instrução se.

Não há nada de errado com o operador lógico mod. Falso.

b) a lacuna I deve ser preenchida com a instrução imprima ("O ano é do século XX pois inicia-se com ", n).

Ao calcular o resto de uma divisão inteira de um ano por 100, você terá os dois últimos dígitos do ano. Observe:



Portanto, falso.

c) a lacuna I deve ser preenchida com a instrução imprima ("Os dois últimos dígitos do ano = ", n).

Esta é a correta, pelo mesmo motivo explicado no item anterior.

d) se for fornecido 1950 para ano será impresso Ano inválido.

Somente será inválido se a condição (ano > 1949 e ano<=1999) for falsa; no caso, se o ano for menor ou igual a 1949 ou maior do que 1999, o que não é o caso. Falso.

e) o usuário pode digitar apenas valores de ano com 4 dígitos, positivos e menores ou iguais ao ano atual (2018).

Não há esta regra no algoritmo. Falso. Gabarito: C

7. (FCC – 2018 – SABESP) Considere, por hipótese, que a SABESP utiliza diferentes preços de tarifas para os serviços de abastecimento de água e/ou coleta de esgoto para o município de São Paulo. Para a categoria Residencial/Favela as tarifas são:

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

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

a. o tipo básico inteiro para ambas as variáveis.
b. o tipo básico caracter para consumo e o tipo básico inteiro para valor.
c. a instrução leia para ler o valor que o usuário deverá pagar.
d. as instruções se então senão de forma aninhada para avaliar as diferentes faixas de consumo.
e. a instrução escolha caso para avaliar os diferentes tipos de valor que poderão ser pagos pelo usuário.

Vamos comentar item a item:

a. o tipo básico inteiro para ambas as variáveis.

O tipo básico inteiro não pode ser usado para o valor da tarifa, visto que há casas decimais. Falso.

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

O valor do consumo deve ser um inteiro. Mas o tipo básico inteiro não pode ser usado para o valor da tarifa, visto que há casas decimais.

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

A leitura é feita do valor do consumo, e não o valor que ele deve pagar – que deve ser calculado. Falso.

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

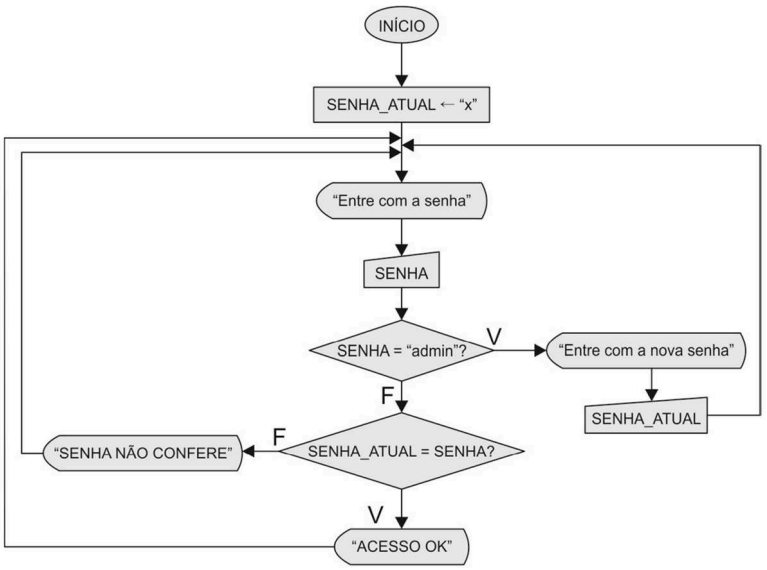
É possível criar instruções se.. então para avaliar as diferentes faixas de consumo. Exemplo: se consumo >= 0 e consumo <= 10 então valor <- 6,25; unidade <- 'mês';
se consumo >= 11 e consumo <= 20 então valor <- 0,71; unidade <- 'm3';
se consumo >= 21 e consumo <= 20 então valor <- 2,36; unidade <- 'm3';
se consumo >= 31 e consumo <= 50 então valor <- 7,14; unidade <- 'm3';
se consumo > 50 então valor <- 7,89; unidade <- 'm3';
Verdadeiro.

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

Não é o adequado usar a instrução escolha..caso porque os diferentes tipos de valor são definidos com por um intervalo, e não por valores discretos. Falso. Gabarito: D

8. (FCC – 2017 – ARTESP) Considere o fluxograma abaixo.

De acordo com a lógica expressa pelo fluxograma, conclui-se que:

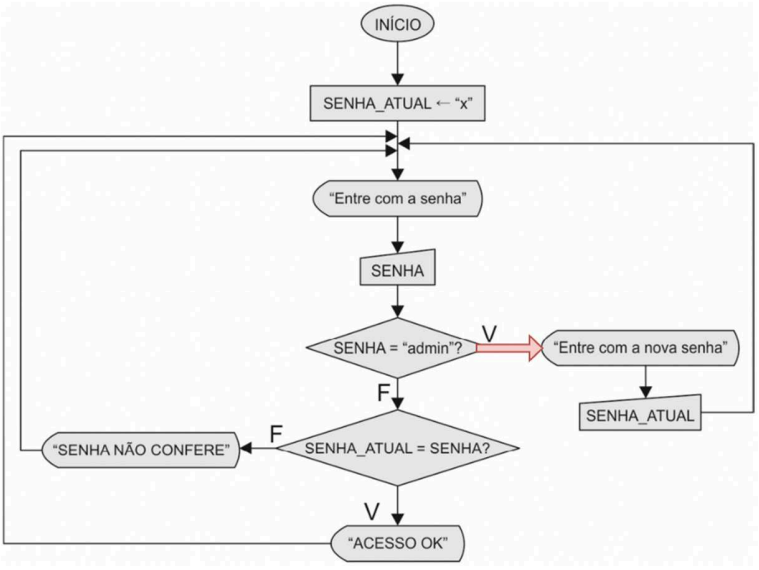


a) a solicitação da senha é encerrada quando o usuário fornece a senha admin.
b) somente o usuário com a senha admin consegue alterar a variável SENHA.
c) o usuário com a senha admin avaliada como verdadeira nunca chega ao comando que exibe ACESSO OK.
d) quando a SENHA_ATUAL não confere, esta é inicializada com "x".
e) após acessar o comando que exibe ACESSO OK a estrutura de repetição finaliza.

Comentários:

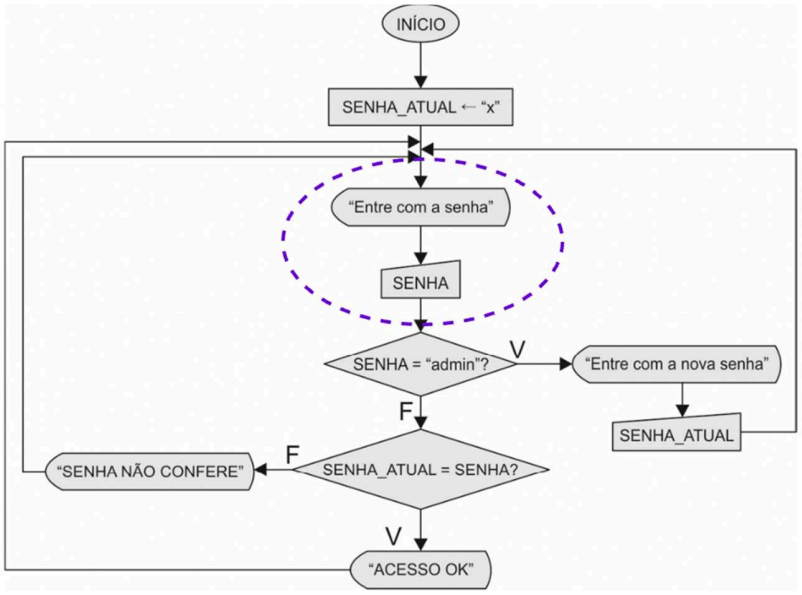
Temos nessa questão um fluxograma de alteração de senha. Vamos explicar item a item.

a) a solicitação da senha é encerrada quando o usuário fornece a senha admin.



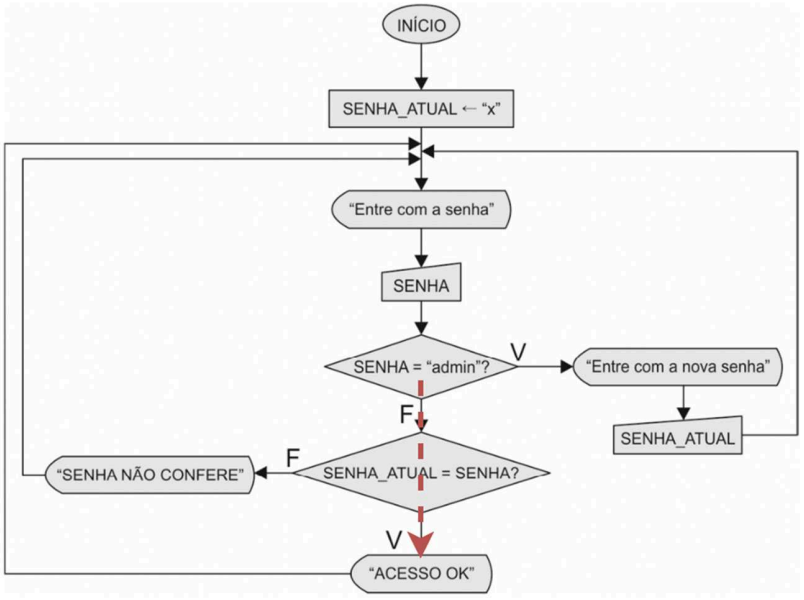
Neste caso, ele segue o caminho indicado em vermelho. Ou seja, não encerra, mas, sim, pede para entrar nova senha. Falso.

b) somente o usuário com a senha admin consegue alterar a variável SENHA.



Como o passo pra informar a variável SENHA ocorre antes da verificação da "SENHA = admin", então qualquer usuário pode trocar a variável da senha (embora somente os usuários com senha "admin" possam efetivamente entrar com uma senha nova).

c) o usuário com a senha admin avaliada como verdadeira nunca chega ao comando que exibe ACESSO OK.



Para chegar no "ACESSO OK", é necessário passar pelo caminho indicado em vermelho. Ou seja, é necessário que o teste SENHA = "admin" seja falso. Ou seja, se sempre for verdadeira, nunca chegará no "ACESSO OK". Verdadeiro.

d) quando a SENHA_ATUAL não confere, esta é inicializada com "x".

À variável SENHA_ATUAL é sempre atribuído o valor “x”, sendo este o segundo passo do fluxograma. Falso.

e) após acessar o comando que exibe ACESSO OK a estrutura de repetição finaliza.

Não. Ele volta para o “Entre com a senha”, pois há uma seta ligando “Acesso OK” com “Entre com a senha”. Falso. Gabarito: C

10.(FCC – 2017 – ARTESP) Considere o algoritmo em pseudocódigo abaixo.

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

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

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

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

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

Este algoritmo

a) não poderia usar a categoria 0 no comando escolha, nem atribuir zero ao valor do pedágio.
b) apresenta erro de lógica na condição do comando condicional se.

c) teria que usar uma condição no comando enquanto (verdadeiro) faça, pois este não pode avaliar apenas o valor lógico verdadeiro.
d) tem erro de sintaxe, pois o comando escolha deveria estar dentro da cláusula senão do comando condicional se.
e) tem erro de sintaxe, pois o comando escolha deveria ter a cláusula senão, que é obrigatória.

Comentários:

Vamos de item em item:

a) não poderia usar a categoria 0 no comando escolha, nem atribuir zero ao valor do pedágio.

```
escolha(categoria)
  caso 0:    pedagio ← 0
```

Pode, pois, na opção de escolha da categoria, há a opção caso 0. Falso.

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

Na linha, há a seguinte condição: se (categoria < 0 e categoria > 8). Não tem como um valor ser menor do que zero e maior do que 8 ao mesmo tempo. Verdadeiro!

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

Não precisa, pois existe a linha “então vá para FINALIZA” permite direcionar a marcação “FINALIZA”, que está depois do fim do enquanto. Falso.

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

Não faz diferença. Se estivesse no “senão”, o programa funcionaria da mesma forma, já que o comando “então vá para o FINALIZA” joga o fluxo da execução pra fora do “enquanto”. Falso.

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

O “senão” não é obrigatório no “escolha”. Só se o programador quiser. Falso. Gabarito: B

11. (FCC – 2017 – TRE-SP) Considere as duas funções, abaixo, escritas em pseudocódigo, que implementam uma lógica.

```
função f1 (N: inteiro): real
Início
        se (N<=1)
              então    retorna  1
              senão    retorna (N  * f1 (N - 1))
        fim se
Fim

função f2 (N: inteiro): real
Var     i: inteiro
        result: real
Início
     result ← 1
     para (i←2 até N passo 1) faça
          result  ← result * i
     fim para
     retorna result
Fim
```

A função

a) f1 e a função f2 recebem uma variável real e retornam um valor inteiro.
b) f1 é executada apenas uma vez, já que em seu corpo existe apenas um comando condicional.
c) f2 é executada N-2 vezes.
d) recursiva faz cálculos e apresenta resultados totalmente diferentes da função iterativa.
e) iterativa e a função recursiva retornam 1 para valores de N=0 e N=1.

Comentários:

Vamos de item em item:

a) f1 e a função f2 recebem uma variável real e retornam um valor inteiro.

O contrário: recebem um parâmetro inteiro e retornam um real. Falso.

b) f1 é executada apenas uma vez, já que em seu corpo existe apenas um comando condicional.

A f1 é uma função recursiva. Caso receba valores maiores do que 1, será executada mais de uma vez, pois o fluxo cairá, pelo menos uma vez, no senão, onde ela é chamada novamente.

c) f2 é executada N-2 vezes.

Não. É executada apenas uma vez, visto que é uma função iterativa, e não recursiva. Ela não é chamada dentro dela mesma. Falso.

d) recursiva faz cálculos e apresenta resultados totalmente diferentes da função iterativa.

É possível implementar de maneira iterativa funções recursivas, e retornar o mesmo resultado. A f1 é recursiva, enquanto a f2 é iterativa. As duas realizam o mesmo cálculo: o fatorial de N. Falso.

e) iterativa e a função recursiva retornam 1 para valores de N=0 e N=1.

Correto. Na f1, se for 0, entra no "se", e retorna 1. Se for 1, entra no "se" também, e retorna 1. Na f2, se for 0 ou 1, nem sequer entra no "para... faça"; e retorna o valor de result, que é 1. Gabarito: E

12. (FCC – 2017 – TRE-SP) Considere a lógica do algoritmo, abaixo, expressa em pseudocódigo.

De acordo com a lógica apresentada,

```
Var
   tipo V= vetor [0..4] inteiro
   var  j, voto: inteiro
   votos: V

Início
   para (j ← 0 até 4 passo 1) faça
             votos[j] ← 0
   fim para

   enquanto  (verdadeiro)
       imprima ("Digite o voto (1,2,3 ou 0 (branco) -1 finaliza): ")
       leia (voto)
       se (voto = -1)
          então   vá para  RESULT
       fim se
       se (voto < 0 OU voto > 3)
           então   votos[4] ← votos[4] + 1
           senão   votos[voto] ← votos[voto] + 1
       fim se
   fim enquanto
   RESULT:
    para (j ← 1 até 3 passo 1) faça
             imprima ("O candidato ", j," obteve ", votos[j], " votos")
    fim para

    imprima ("Número de votos em branco= ", .....I.....)
    imprima (".....II.....", votos[4])
Fim
```

a) a instrução se (voto < 0 OU voto > 3) deveria utilizar o operador lógico E ao invés do OU.
b) a lacuna I deve ser preenchida com votos[1]
c) a lacuna II deve ser preenchida com Número de votos nulos =
d) para saber o número total de eleitores basta percorrer o vetor e somar todas as posições de 1 a 3.
e) logo após RESULT:, a instrução para deveria se iniciar em 0 e ir até 4.

Comentários:

Da análise do algoritmo, podemos deduzir que:

- Há um vetor de quatro posições, cada posição inicializada com zero.
- O usuário deve preencher os valores 1, 2 e 3 para declarar votos nos candidatos 1, 2 e 3; 0 caso o voto seja em branco; e -1 para finalizar e mostrar os resultados.

- Usando o número informado (0, 1, 2 ou 3), armazenado na variável voto, o sistema incrementa o valor em vetor[voto]. Se o valor informado for menor que zero, ou maior que 3, ele incrementa o valor de vetor[4].
- Dessa forma:
  - Nas posições 1, 2 e 3 são adicionados os votos referentes aos candidatos 1, 2 e 3.
  - Na posição 0, são adicionados os votos em branco.
  - Na posição 4, são adicionados os votos nulos (já que não são 0, 1, 2 ou 3).

Vamos de item em item:

a) a instrução se (voto < 0 OU voto > 3) deveria utilizar o operador lógico E ao invés do OU.

Se utilizasse o operador lógico E, seria inválida, já que não é possível que o voto seja menor do que zero E maior do que 3 ao mesmo tempo. Falso.

b) a lacuna I deve ser preenchida com votos[1]

O número de votos em branco é adicionado na posição 0 do vetor, ou seja, vetor[0]. Portanto, deveria ser preenchida com votos[0]. Falso.

c) a lacuna II deve ser preenchida com Número de votos nulos =

A lacuna II imprime o valor de votos[4]. Portanto, este é justamente o valor dos votos nulos. Correto.

d) para saber o número total de eleitores basta percorrer o vetor e somar todas as posições de 1 a 3.

Não, pois existem as posições 0 e 4, referentes aos votos em branco e nulos, respectivamente. Falso.

e) logo após RESULT:, a instrução para deveria se iniciar em 0 e ir até 4.

Não, pois o objetivo do loop é mostrar os votos válidos nos candidatos 1, 2 e 3. Gabarito: C

13.(FCC – 2017 – TRT-24) 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:
Vamos seguir o algoritmo.

leia(v1, v2, v3)

v1 = 3, v2 = 3 e v3 = 4
exiba(v1)
Exibirá o valor 3.
enquanto v3>1 faça
v3 > 1, 4 > 3. Verdadeiro. Então entra no enquanto.
v1 <- v1*v2 = 3*3 = 9
v3 <- v3 – 1 = 4 – 1 = 3
v1 = 9, v3 = 3
Então volta para o enquanto. 3 > 1, continua no enquanto.
v1 <- v1*v2 = 9*3 = 27
v3 <- v3 – 1 = 3 – 1 = 2
Novamente volta para o enquanto. 2 > 1, continua no enquanto. v1 <- v1*v2 = 27*3 = 81
v3 <- v3 – 1 = 2 – 1 = 1
Novamente volta para o enquanto. 1 > 1. Falso. Sai do enquanto. exiba(v1)
Irá mostrar o valor de v1 atual, que é 81. Gabarito: D

14. (FCC – 2017 – ARTESP) Considere o algoritmo em pseudocódigo abaixo:

```
função  digitos (n: inteiro): inteiro
início
      se (n < 10)
         então          retorna 1
         senão          retorna (1+ digitos (n/10))
      fim se
fim

início

   imprima(" Para 654321 o resultado da função digitos = ", digitos (654321))

fim.
```

Considerando que o operador / realiza a divisão inteira, ao executar o algoritmo acima será impresso: Para 654321 o resultado da função digitos =

a) 21
b) 123456
c) 654321
d) 100000
e) 6

Comentários:

Vamos seguir o algoritmo.

imprima(" Para 654321 o resultado da função digitos = ", digitos (654321))
Chama a função dígitos passando o parâmetro n = 654321. Só vai imprimir depois que retornar o valor da função.
se (n < 10)
654321 < 10? Falso, então vai para o “senão”.
senão retorna (1+ digitos (n/10))
Aqui ele vai pegar 1 + dígitos(n/10). É uma função recursiva. n/10 = 654321/10 = 65432 (sem as casas decimais, pois se trata de um inteiro).
Vamos contando a quantidade de vezes que roda a função. Essa é a primeira vez.
Passando então o novo valor de n = 65432. Temos:
se (n < 10)
65432 < 10? Falso, então vai para o “senão”.
senão retorna (1+ digitos (n/10))
Novamente a função recursiva. n/10 = 65432/10 = 6543. Rodemos novamente digitos(6543). Segunda vez.
se (n < 10)
6543 < 10? Falso, então vai para o “senão”.
Novamente a função recursiva. n/10 = 6543/10 = 654. Rodemos novamente digitos(654). Terceira vez.
se (n < 10)
654 < 10? Falso, então vai para o “senão”.
Novamente a função recursiva. n/10 = 654/10 = 65. Rodemos novamente digitos(65). Quarta vez.
se (n < 10)

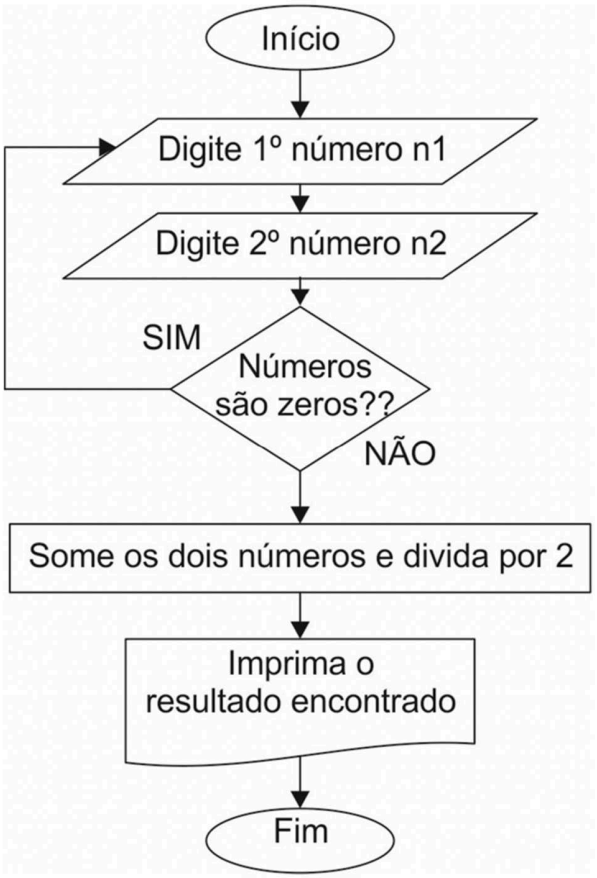
Novamente a função recursiva. n/10 = 65/10 = 6. Rodemos novamente digitos(6). Quinta vez.
se (n < 10)
6 < 10? Verdadeiro, então vai para o “então”.
então retorna 1
Agora ele retornou um valor.
Note que a função rodou 5 vezes, então o resultado vai ser e como o seguinte:

digitos(6) = 1
digitos(65) = 1 + digitos(6) = 1 + 1 = 2
digitos(654) = 1 + digitos(65) = 1 + 2 = 3
digitos(6543) = 1 + digitos(654) = 1 + 3 = 4
digitos(65432) = 1 + digitos(6543) = 1 + 4 = 5
digitos(654321) = 1 + digitos(65432) = 1 + 5 = 6

Dessa forma:
imprima(" Para 654321 o resultado da função digitos = ", digitos (654321)) Vai

imprimir Para 654321 o resultado da função digitos = 6. Gabarito: E

15.(FCC – 2016 – CREMESP) Considere o diagrama abaixo:



Analisando o raciocínio lógico e as estruturas lógicas utilizadas no diagrama, é correto afirmar que:

a) o losango com a inscrição "Números são zeros??" indica que há uma estrutura condicional do tipo escolha-caso.
b) há um comando de repetição do tipo enquanto (condição) faça sendo "Números são zeros??" a condição.
c) a lógica implementa a solução de cálculo da média de 2 números diferentes de zero.
d) se um dos números digitados for zero, o comando de repetição para e nada é impresso.
e) se os dois números digitados na primeira vez forem zero, os dois serão somados e divididos por 2.

Comentários:

Temos nessa questão um fluxograma que poderia der traduzido para o seguinte pseudocódigo:

```
repita
        ler(n1)
        ler(n2)
até (n1 != 0 e n1 != 0)
resultado <- (n1 + n2)/2
imprime(resultado)
```

Dito isso, analisemos as opções:

a) o losango com a inscrição "Números são zeros??" indica que há uma estrutura condicional do tipo escolha-caso.

Não, é um repita..até. Falso.

b) há um comando de repetição do tipo enquanto (condição) faça sendo "Números são zeros??" a condição.

Não. É um repita..até. Falso.

c) a lógica implementa a solução de cálculo da média de 2 números diferentes de zero.

Isso mesmo. Ele pede ao usuário que digite n1 e n2. Se os dois forem zero, volta e pede de novo pra digitar. Isso até serem diferentes de zero. Daí, soma os dois e divide por 2, o que é calcular a média dos dois números. Correto!

d) se um dos números digitados for zero, o comando de repetição para e nada é impresso.

Não. Neste caso, ele pede novamente ao usuário que digite n1 e n2. Falso.

e) se os dois números digitados na primeira vez forem zero, os dois serão somados e divididos por 2.

Não, isso somente ocorrerá se os dois números forem diferentes de zero. Falso. Gabarito: C

16.(FCC – 2015 – DPE-SP) Considere o algoritmo em pseudocódigo no qual DIV calcula o quociente da divisão inteira e MOD o resto da divisão inteira:

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

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

Fim.
```

O algoritmo em pseudocódigo acima

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

Comentários:

Antes de mais nada, existem alguns operadores aqui que precisam ser explicados:

- DIV: obtém o quociente da divisão de dois números inteiros.
- MOD: obtém o resto da divisão números inteiros.

Dito isso, o algoritmo ele funciona da seguinte maneira:

- Lê uma variável taxa.
- Obtém o quociente da divisão de taxa por 5, e armazena na variável “quociente”.
- Obtém o resto da divisão da taxa por 5, e armazena na variável “resto”.
- Define as variáveis “tres” e “cinco” como 0.

- Dependendo do resto da divisão calculada, ele vai definir novos valores para as variáveis “cinco” e “tres”.
- Imprime os valores de “taxa”, “cinco” e “tres”.

Note que, embora o programa peça para digitar um valor maior ou igual a 8, ele não testa se o valor da “taxa” realmente foi igual ou maior a 8.
Posto isso, analisemos item a item:

a) garante que o valor de entrada seja maior ou igual a 8 para que seja possível dividir a taxa por 5 e por 3.

Em momento algum ele realiza este teste. Falso.

b) para o valor inicial da taxa = 22 finaliza com cinco= 2 e tres=4.

Se a taxa for 22, o quociente da divisão é 4 e o resto é 2.
Portanto, executa:
cinco ← quociente - 2 = 4 – 2 = 2
tres ← 4
Portanto, cinco = 2 e três = 4. Correto!

c) determina o maior número de 5 e de 3 unidades cuja soma dá o valor da taxa.

Não é o que o programa faz, pois a soma dos valores “cinco” e “tres” não resulta no valor da taxa. Falso.

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

Se for 15, então o quociente da divisão é 3 e o resto é 2.
Portanto, executa:
cinco ← quociente - 2 = 3 – 2 = 1
tres ← 4
Portanto, cinco = 1 e três = 4

e) sempre finaliza com valores da variável cinco maiores ou igual a 1, mas a variável tres pode ter valor 0.

Não, pois, contra a vontade do programa, se eu digitar um número para taxa menor do que 5, o quociente será 0, e o valor de cinco não tem como ser maior do que 1. Falso. Gabarito: B

17. (FCC – 2015 – DPE-SP) Considere o algoritmo a seguir, na forma de pseudocódigo:

```
Var n, i, j, k, x: inteiro
Var v: vetor[0..7] inteiro
Início
  v[0] ← 12
  v[1] ← 145
  v[2] ← 1
  v[3] ← 3
  v[4] ← 67
  v[5] ← 9
  v[6] ← 45
  n ← 8
  k ← 3
  x ← 0
  Para j ← n-1 até k passo -1 faça
    v[j] ← v[j - 1];
  Fim_para
  v[k] ← x;
Fim
```

Este pseudocódigo

a) exclui o valor contido na posição x do vetor v.
b) insere o valor de x entre v[k-1]e v[k] no vetor v.
c) exclui o valor contido na posição k do vetor v.
d) tentará, em algum momento, acessar uma posição que não existe no vetor.
e) insere o valor de k entre v[x] e v[x+1] no vetor v.

Comentários:

Esta é uma questão um pouco complicada.

O algoritmo começa inicializando os valores do vetor, e depois os valores de n, k e x.
No loop para..até..faça, ele vai armazenar, inicialmente, na variável j o valor de n – 8. Como a instrução “passo” indica o valor “-1”, significa que o valor da variável j será decrementado em uma unidade a cada iteração. Ou seja, ele vai iterar de 7 até k, de maneira decrescente.

Depois, ele executa v[j] ← v[j - 1];

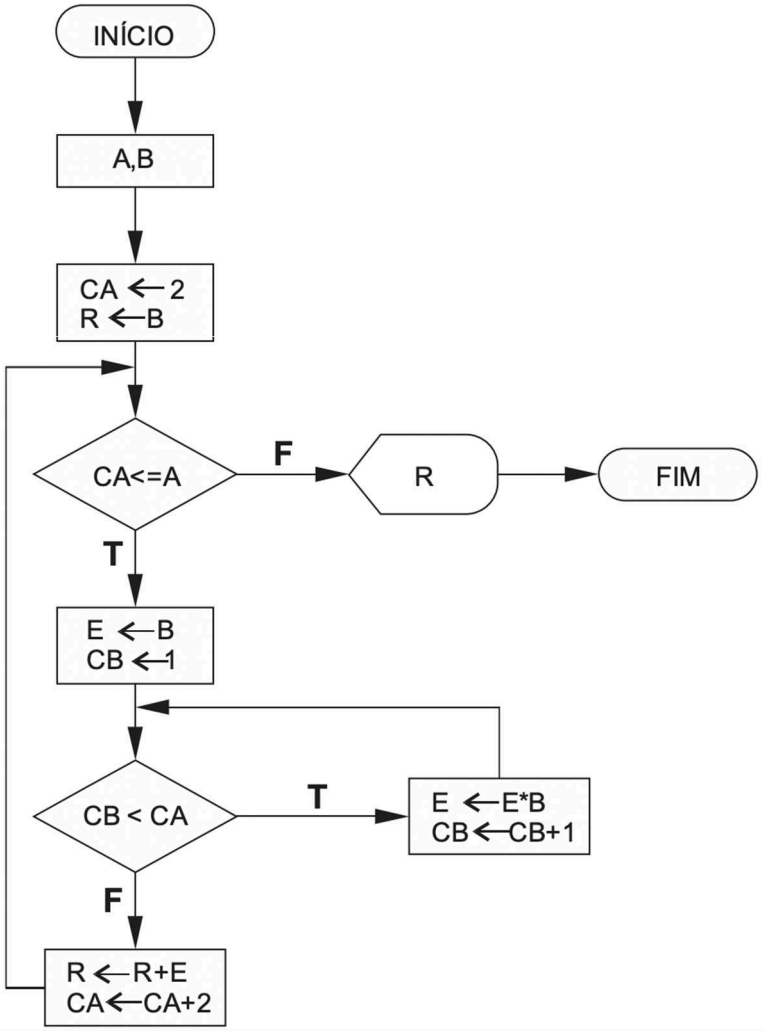
Isso significa que ele passa para a posição seguinte o valor da posição anterior no vetor.
Por exemplo:
j = 8
Então ele coloca em v[8] o valor de v[7].
j = 7
Então ele coloca em v[7] o valor de v[6]
E assim por diante. Até chegar em “k”. Suponha que k = 6.

Até aí, note que ele transpôs para frente os valores do vetor, certo? E daí vem a seguinte instrução: v[k] ← x;

Significa que ele coloca em v[6] o valor de x.

Os valores de v[1], v[2], v[3], v[4] e v[5] se mantém inalterados. O valor de v[6] vira o valor de k. E os valores seguintes são transpostos. Na prática, ele está inserindo o valor de x entre v[k-1] e v[k].
Gabarito: B

18.(FCC – 2015 - MANAUSPREV) Considere o fluxograma abaixo:



Se forem lidos para as variáveis A e B, respectivamente, os valores 4 e 4 será exibido o valor
a) 47994.
b) 276.
c) 1338.
d) 4372.
e) 20.

Comentários:

Vamos seguir o fluxograma, considerando T = true = verdadeiro, F = false = falso. A = 4, B = 4
Definem-se os valores iniciais. CA <- 2
R <- B
CA = 2 e R = 4
CA <= A
CA é menor ou igual a A? CA = 2, A = 4. Verdadeiro. Então passa para o fluxo da seta com “T”. E <- B
CB <- 1
E = 4 e CB = 1
CB < CA
Realiza-se o teste. CB é menor que CA? CB = 1 e CA = 2. Verdadeiro. Passa pro passo da seta com “T”.
E <- E*B
CB <- CB + 1
Inicialmente E = 4 e B = 4. Portanto E <- 4*4 = 16. Então, E = 16.
E CB = CB + 1 = 1 + 1 = 2. Portanto, CB = 2. E volta para o teste.
CB < CA
Testa novamente. CB = 2 e CA = 2. Falso. Vai pra seta o falso.
R <- R + E
CA <- CA + 2
Inicialmente, R = 4 e E = 16. Então, R = 4 + 16. R = 20.
CA = CA + 2 = 2 + 2 = 4
Volta para o teste.
CA <= A
CA = 4 e A = 4. Verdadeiro. Segue pra seta do “T”.
E <- B
CB <- 1
E = 4 e CB =1.
CB < CA
CB é menor que CA? CB = 1 e CA = 4. Verdadeiro. Vai pra seta do “T”. E <- E*B
CB <- CB + 1
Inicialmente E = 4 e B = 4. Portanto E <- 4*4 = 16. Então, E = 16. CB = 1 + 1 = 2. Volta para o teste.
CB < CA
CB é menor que CA? CB = 2 e CA = 4. Verdadeiro. Vai pra seta do “T”. E <- E*B
CB <- CB + 1
Inicialmente E = 16 e B = 4. Portanto E <- 16*4 = 64. Então, E = 64. CB = 2 + 1 = 3. Volta para o teste.
CB < CA
CB é menor que CA? CB = 3 e CA = 4. Verdadeiro. Vai pra seta do “T”. E

Inicialmente E = 4 e B = 4. Portanto E <- 64*4 = 256. Então, E = 256. CB = 3 + 1 = 4. Volta para o teste.
CB < CA
CB é menor que CA? CB = 4 e CA = 4. Falso. Vai para a seta do “F”. R <- R + E
CA <- CA + 2
R = 20 + 256 = 276
CA = 4 + 2 = 6
E volta para o teste:
CA <= A
CA é menor ou igual a A? CA = 6 e A = 4. Verdadeiro.
Finalmente, imprime o valor de R, que é 276. Gabarito: B

19.(FCC – 2014 – TRT-16) Para responder às questões de números 43 e 44, considere o algoritmo em pseudo-código abaixo.

```
tipo V= vetor [1..10] inteiro
var   i, k: inteiro
      a: logico
      vet: V

inicio

      leia (k)
      i ← 1
      a ← falso
      enquanto (i<=10 e a=falso) faça
      inicio
            se (vet[i] = k)
              entao
                    imprima(“Sucesso”)
                    a ← verdadeiro

              senão
                    imprima(“Insucesso”)
            fim se
            i ← i+1

      fim enquanto

fim
```

Considere que o vetor vet possua os seguintes valores: 6, 5, 1, 9, 0, 1, 4, 2, 3 e 7. É correto afirmar:

a) O algoritmo não funciona quando há elementos repetidos no vetor, portanto, o vetor não pode ter elementos repetidos.
b) Se for fornecido 1 para k, será impresso “Insucesso” duas vezes.
c) Somente se o valor de k não for encontrado no vetor será impresso “Insucesso”.
d) Faltou o comando a ← falso depois de imprima(“Insucesso”) no senão do comando se para esta frase ser impressa uma única vez na pesquisa sequencial malsucedida no vetor.
e) Se for fornecido 1 para k, será impresso “Sucesso” duas vezes.

Comentários:

Da análise do algoritmo, nota-se que seu comportamento é o seguinte:

- Temos um vetor de 10 posições.
- Lê o valor de k, e define valores iniciais i = 1 e a = falso.
- Enquanto o for i <= 10 e a = falso, ele vai rodar um loop:
    - No fim do loop, ele executa i <- i + 1, ou seja, incrementa o valor de i. Significa que o loop vai rodar:
        - De 1 a 10, em cada uma das iterações do loop;
        - Ou até que o valor de "a" seja verdadeiro.
    - Em cada uma das iterações do loop, ele verifica se o vetor na posição i é igual ao valor k, que foi lido.
        - Se for, então imprime "Sucesso", e define a = verdadeiro.
        - Se não for, imprime "Insucesso", e segue o fluxo.
- Na prática, o que o algoritmo faz é percorrer o vetor posição a posição. Vai imprimir "Insucesso" em cada uma das posições em que o valor não corresponder a "k". Quando finalmente encontrar o valor de "k", ele imprime "Sucesso" e sai do loop.
- Logo, só vai imprimir "Sucesso" uma vez, pra primeira vez que o valor tiver no vetor.

Verifiquemos item a item:

a) O algoritmo não funciona quando há elementos repetidos no vetor, portanto, o vetor não pode ter elementos repetidos.

Não há problema algum com o algoritmo se tiver valores repetidos. Vai sair do loop no primeiro que encontrar. Falso.

b) Se for fornecido 1 para k, será impresso "Insucesso" duas vezes.

O primeiro valor "1" está na terceira posição. Portanto, será impresso "Insucesso" duas vezes, e "Sucesso" uma vez. Correto.

c) Somente se o valor de k não for encontrado no vetor será impresso "Insucesso".

Não. Basta que o valor no vetor não seja o primeiro.

d) Faltou o comando a ← falso depois de imprima("Insucesso") no senão do comando se para esta frase ser impressa uma única vez na pesquisa sequencial malsucedida no vetor.

Não é necessário, pois o valor de a já é falso desde o início, e só muda se entrar no "Sucesso".

e) Se for fornecido 1 para k, será impresso "Sucesso" duas vezes.

Não. É o contrário. Será impresso "Insucesso" duas vezes, e "Sucesso" uma vez. Gabarito: B

20. (FCC – 2014 – TRT-16) No algoritmo há:

```
tipo V= vetor [1..10] inteiro
var   i, k: inteiro
      a: logico
      vet: V

inicio

      leia (k)
      i ← 1
      a ← falso
      enquanto (i<=10 e a=falso) faça
      inicio
            se (vet[i] = k)
              entao
                    imprima("Sucesso")
                    a ← verdadeiro

              senão
                    imprima("Insucesso")
            fim se
            i ← i+1

      fim enquanto

fim
```

a) diferentes estruturas de dados, de diferentes tipos básicos, que são encontrados em linguagens de programação procedurais.
b) tipos estruturados de dados que são indivisíveis, como inteiro e lógico.
c) um tipo abstrato de dados, que pode ser visto como um modelo matemático, acompanhado das operações definidas sobre o modelo.
d) um tipo estruturado homogêneo de dados, de um tipo básico que pode ser encontrado em diversas linguagens de programação.
e) a implementação de uma lógica particular a uma classe de linguagens de programação, o que limita sua implementação a linguagens orientadas a objetos.

Comentários:
Item a item:

a) diferentes estruturas de dados, de diferentes tipos básicos, que são encontrados em linguagens de programação procedurais.

Não há estruturas de dados como pilhas, listas, filas, etc. Falso.

b) tipos estruturados de dados que são indivisíveis, como inteiro e lógico.

O inteiro é um valor divisível. 4 divide por 2, por exemplo. Falso.

c) um tipo abstrato de dados, que pode ser visto como um modelo matemático, acompanhado das operações definidas sobre o modelo.

Não há tipo abstrato de dados é uma especificação de um conjunto de dados possíveis para uma variável. Não há isso. Falso.

d) um tipo estruturado homogêneo de dados, de um tipo básico que pode ser encontrado em

Sim, há um tipo homogêneo de dados, que é o vetor. No caso, é implementado um vetor de inteiros. Correto.

e) a implementação de uma lógica particular a uma classe de linguagens de programação, o que limita sua implementação a linguagens orientadas a objetos.

Uma classe é utilizada em linguagem orientada a objetos. O pseudocódigo mostrado é procedural. Falso. Gabarito: D

21.(FCC – 2013 – TRT-9) Analise o algoritmo em pseudocódigo abaixo:

```
início
real: n1, n2;
      imprima("Digite a primeira nota: ");
      leia(n1);
      imprima("Digite a segunda nota: ");
      leia(n2);
      se [                                    ]
      então
          media ←(n1+n2)/2;
          imprima ("A media das notas é ", media);
      senão
         imprima("Alguma nota fornecida é inválida.");
      fim se;
fim.
```

Considerando que uma nota válida deve possuir valores entre 0 e 10 (inclusive), a lacuna que corresponde à condição do comando SE é corretamente preenchida por

a) n1 >= 0 OU n1 <=10 OU n2 >= 0 OU n2 <=10
b) (n1 >= 0 E n1 <=10) OU (n2 >= 0 E n2 <=10)
c) (n1 >= 0 OU n1 <=10) E (n2 >= 0 OU n2 <=10)
d) n1 >= 0 E n1 <=10 E n2 >= 0 E n2 <=10
e) n1 > 0 E n1 < 0 E n2 < 10

Comentários:

Vamos item a item.

a) n1 >= 0 OU n1 <=10 OU n2 >= 0 OU n2 <=10

Isso vai permitir que qualquer número n1 e n2 maior do que 0 (inclusive maior do que 10) seja aceito. Falso.

b) (n1 >= 0 E n1 <=10) OU (n2 >= 0 E n2 <=10)

Isso vai permitir que n1 esteja entre 0 e 10, ou que n2 esteja entre 0 e 10. Queremos que os dois estejam entre 0 e 10. Falso.

c) (n1 >= 0 OU n1 <=10) E (n2 >= 0 OU n2 <=10)

Isso também vai permitir que n1 ou n2 sejam maiores do que 0, e inclusive maior do que 10. Falso.

d) n1 >= 0 E n1 <=10 E n2 >= 0 E n2 <=10

Correto. Isso vai permitir somente que n1 e n2 sejam entre 0 e 10. Verdadeiro! e)

n1 > 0 E n1 < 0 E n2 < 10

Isso é uma instrução inválida e vai dar sempre falso, pois não tem como n1 > 0 e n1 < 0.
Gabarito: D

21.(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

22. (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

23.(FCC - 2012 – BANESE) Dadas as variáveis reais (K e M), inteiras (X e Y) e lógicas (W e Z), produz um resultado correto o comando de atribuição:

a) X ← K = M
b) W ← X > Y
c) Y ← M
d) K + M ← 7.5
e) Z ← X + Y

Comentários:

É importante, neste caso, que os resultados dos dois lados da atribuição possuam o mesmo tipo, ou que pelo menos sejam compatíveis.
Vamos, portanto, comentar item a item:

a. X ← K = M

X é inteiro. K é real. M é real.
A expressão “K = M”, ou seja, “K igual a M” pode ser verdadeira ou falsa, portanto, é uma expressão lógica.
Não é possível atribuir a um inteiro (X) o resultado de uma expressão lógica (K = M).
Falso.

b. W ← X > Y

W é lógico. X é inteiro e Y é inteiro. “X > Y”, ou seja, “X é maior do que Y” pode ser verdadeiro ou falso, portanto, é uma expressão lógica.
É possível atribuir a uma variável lógica (W) um valor de uma expressão lógica (X > Y). Verdadeiro.

c. Y ← M

Y é inteiro. M é real. Não é possível, a rigor, atribuir um valor real a um inteiro, sem perder os dados das casas decimais. Perdendo os dados das casas decimais, é possível. Como a alternativa B é totalmente certa, vamos considerar esta falsa. É comum que em questões de concursos tenhamos que selecionar as “mais certas”.
O contrário seria totalmente possível, pois o conjunto dos inteiros está dentro do conjunto dos reais; mas o conjunto dos reais não está dentro do conjunto dos inteiros.

d. K + M ← 7.5

Não é possível atribuir um valor fixo a uma expressão. O lado esquerdo da atribuição deve ser uma única variável. Falso.

e. Z ← X + Y

Z é lógico. X é inteiro. Y é inteiro. X + Y, ou seja, “X mais Y”, resulta em um inteiro.

24. (FCC – 2010 – TJ-PA) Considere a seguinte e somente a seguinte situação: Se um procedimento Px contiver uma referência a um outro procedimento Py que por sua vez contém uma referência direta ou indireta a Px, então

a) Px é subconjunto de Py.
b) Px é categorizado como indiretamente recursivo.
c) Py é subconjunto de Px.
d) Py é categorizado como diretamente recursivo.
e) Py é categorizado como indiretamente recursivo.

Comentários:

Trata-se de uma questão sobre recursividade, pois os procedimentos possuem referência entre si. Vamos comentar item a item.

a) Px é subconjunto de Py.

Px não é subconjunto de Py, já que os procedimentos elas se referenciam entre si. Falso.

b) Px é categorizado como indiretamente recursivo.

Esta é a correta.
Um procedimento X é considerado indiretamente recursivo se tiver referência a um outro procedimento Y, que, por sua vez, tiver uma referência direta ou indireta ao procedimento X.

c) Py é subconjunto de Px.

Py não é subconjunto de Px, já que as funções elas se referenciam entre si. Falso.

d) Py é categorizado como diretamente recursivo.

Incorreto, pois a referência a Px é direta ou indireta, e não direta.
Um procedimento X é considerado recursivo se tiver referência ao próprio procedimento X, e essa referência for explícita.

e) Py é categorizado como indiretamente recursivo.

Incorreto, pois a referência a Px é direta ou indireta, e não direta. Gabarito: B

25. (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

d) Do...While

e) Next

Comentários:

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

26.(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

27.(FCC – TRT – 2011) Em relação à programação de computadores, considere:

I. Métodos de passagem de parâmetros permitem que parâmetros sejam transmitidos entre o programa principal e os subprogramas, sendo que, na passagem de parâmetros por valor, o valor real é passado e uma variável local é criada para armazená-lo; nesse processo sempre será efetuada a cópia dessa variável.

II. Pilha é uma estrutura de dados com acesso restrito aos seus elementos ordenados pelo princípio FIFO; a pilha pode ser usada na avaliação de expressões numéricas, na recursividade e pelos compiladores, na passagem de parâmetros para as funções.

III. Prototipação é uma abordagem que envolve a produção de versões iniciais de um sistema futuro com a qual pode-se realizar verificações e experimentações para se avaliar algumas de suas qualidades antes que o sistema venha realmente a ser construído.

IV. Registro é uma estrutura básica que permite guardar coleções de dados de diferentes tipos, sendo normalmente utilizado quando um objeto tem diferentes atributos.

É correto o que consta APENAS em

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

Comentários:

Vamos explicar item a item.

I. Métodos de passagem de parâmetros permitem que parâmetros sejam transmitidos entre o programa principal e os subprogramas, sendo que, na passagem de parâmetros por valor, o valor real é passado e uma variável local é criada para armazená-lo; nesse processo sempre será efetuada a cópia dessa variável.

Correto, definição perfeita.

II. Pilha é uma estrutura de dados com acesso restrito aos seus elementos ordenados pelo princípio FIFO; a pilha pode ser usada na avaliação de expressões numéricas, na recursividade e pelos compiladores, na passagem de parâmetros para as funções.

Falso. FIFO significa ***"First in... First out"***, ou seja, "primeiro a entrar é o primeiro a sair". Porém, numa pilha, o correto é LIFO, que é ***"Last in... First out"***, ou seja, "o último a entrar é o primeiro a sair". O resto está correto. Falso.

III. Prototipação é uma abordagem que envolve a produção de versões iniciais de um sistema futuro com a qual pode-se realizar verificações e experimentações para se avaliar algumas de suas qualidades antes que o sistema venha realmente a ser construído.

Definição correta.

IV. Registro é uma estrutura básica que permite guardar coleções de dados de diferentes tipos, sendo normalmente utilizado quando um objeto tem diferentes atributos.

Definição correta. Gabarito: D

28.(FCC – 2010 – TRE-AM) Formalização de algoritmo proposto em 1936, universalmente conhecido e aceito. Trata-se de um mecanismo simples, que formaliza a ideia de uma pessoa que realiza cálculos, denominado:

a) Recursividade de Bird
b) Máquina de Redução.
c) Máquina de Turing.

e) Máquina com Pilhas.

Comentários:

Vamos explicar item a item.

a) Recursividade de Bird

Trata-se de um formalismo criado por R. Bird para tratar o conceito de recursão por meio de definições recursivas. Falso.

b) Máquina de Redução.

Envolve um algoritmo de mapeamento entre linguagens cujo objetivo é traduzir os problemas. Falso.

c) Máquina de Turing.

É um modelo abstrato de computador, restrito somente aos aspectos lógicos do seu funcionamento. É o que está sendo referenciado na questão, sendo, portanto, a formalização do algoritmo. Verdadeiro!

d) Sistema de Post.

Criado por Emil Post, trata-se de um sistema para manipular strings que iniciam com um conjunto finito de palavras, transformando-as em um conjunto finito de regras, criando, portanto, uma linguagem formal. Falso.

e) Máquina com Pilhas.

Foi formalizado na década de 60. Sua memória de entrada é separada das memórias de trabalho e de saída, com estruturas auxiliares do tipo pilha. Falso. Gabarito: C

29.(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

QUESTÕES COMENTADAS –

1. (FGV – 2018 – Câmara de Salvador-BA) Expressões lógicas são frequentemente utilizadas em linguagens de programação. Por exemplo, um comando if com a expressão

if not (A and B)

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

a) A or B
b) not A or not B
c) not A or B
d) not (not A or not B)
e) A and B

Comentários:

Neste caso, utilizamos a propriedad que é aplicar a negação às proposições, e inverter disjunção para conjunção, ou o inverso:

não (A e B) = não A ou não B

não (A ou B) = não A e não B

Portanto, not (A and B) = not A or not B. Gabarito: Letra B.

2. (FGV – 2018 – Câmara de Salvador-BA) Observe o trecho de pseudocódigo exibido a seguir.

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

Numa hipotética execução desse código, os valores exibidos seriam:

a) 2, 5, 7;
b) 6, 13, 15;
c) 6, 13, 19;
d) 7, 15, 21;
e) 7, 17, 23

Comentários:

Inicia o programa definindo os valores para as variáveis: a = 1, b = 3, c = 5

Chega no while.

Iteração: 1

Realiza o teste: b <> a and c < 20

3 <> 1 and 5 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Realiza o teste: 1 > 5

Falso. Entra no "else".

Realiza o teste: if a + b < c. 1 + 3 < 7.

Verdadeiro. Entra dentro do if.

a := b - a = 3 - 1 = 2 b

:= b + 2 = 3 + 2 = 5

Fim do loop. Valores: a = 2, b = 5, c = 7

Iteração: 2

Realiza o teste: b <> a and c < 20

5 <> 2 and 7 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Realiza o teste: 2 > 7

Falso. Entra no "else".

Realiza o teste: if a + b < c. 2 + 5 < 9.

Verdadeiro. Entra dentro do if.

a := b - a = 5 - 2 = 3 b

:= b + 2 = 5 + 2 = 7

Fim do loop. Valores: a = 3, b = 7, c = 9

Iteração: 3

Realiza o teste: b <> a and c < 20

7 <> 3 and 9 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Falso. Entra no "else".

Realiza o teste: if a + b < c. 3 + 7 < 11.

Verdadeiro. Entra dentro do if.

a := b - a = 7 - 3 = 4 b

:= b + 2 = 7 + 2 = 9

Fim do loop. Valores: a = 4, b = 9, c = 11

Iteração: 4

Realiza o teste: b <> a and c < 20

9 <> 4 and 11 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Realiza o teste: 4 > 11

Falso. Entra no "else".

Realiza o teste: if a + b < c. 4 + 9 < 13.

Falso. Não entra dentro do if.

Fim do loop. Valores: a = 4, b = 9, c = 13

Iteração: 5

Realiza o teste: b <> a and c < 20

9 <> 4 and 13 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Realiza o teste: 4 > 13

Realiza o teste: if a + b < c. 4 + 9 < 15.

Verdadeiro. Entra dentro do if.

a := b - a = 9 - 4 = 5

b := b + 2 = 9 + 2 = 11

Fim do loop. Valores: a = 5, b = 11, c = 15

Iteração: 6

Realiza o teste: b <> a and c < 20

11 <> 5 and 15 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Realiza o teste: 5 > 15

Falso. Entra no "else".

Realiza o teste: if a + b < c. 5 + 11 < 17.

Verdadeiro. Entra dentro do if.

a := b - a = 11 - 5 = 6

b := b + 2 = 11 + 2 = 13

Fim do loop. Valores: a = 6, b = 13, c = 17

Iteração: 7

Realiza o teste: b <> a and c < 20

13 <> 6 and 17 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Falso. Entra no "else".

Realiza o teste: if a + b < c. 6 + 13 < 19.

Falso. Não entra dentro do if.

Fim do loop. Valores: a = 6, b = 13, c = 19

Iteração: 8

Realiza o teste: b <> a and c < 20

13 <> 6 and 19 < 20

Verdadeiro. Entra dentro do loop.

if a > c {

Realiza o teste: 6 > 19

Falso. Entra no "else".

Realiza o teste: if a + b < c. 6 + 13 < 21.

Verdadeiro. Entra dentro do if.

a := b - a = 13 - 6 = 7

b := b + 2 = 13 + 2 = 15

Fim do loop. Valores: a = 7, b = 15, c = 21

Realiza o teste: b <> a and c < 20

15 <> 7 and 21 < 20. Falso

Fim do programa. Valores: a = 7, b = 15, c = 21

Gabarito: Letra D.

3. (FGV – 2018 – SEFIN-RO) Analise o trecho de pseudocódigo a seguir.

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

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

A)
```
2  20
5  20
8  20
```

B)
```
2  20
```

C)
```
2
5
8
```

D)
```
2
5
8
```

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

Comentários:

Vamos seguir o algoritmo.

Define o valor de a = 2

Depois, define o valor de b = a * 10 = 2*10 = 20.

Então, faz o teste do while.

Teste: a < 10 and b > 14. 2 < 10 and 20 > 14. Verdadeiro. Entra no while.

Depois, vem um if: if a <> b. Realiza o teste: a <> b. 2 <> 20.

Fora dos ifs, mas ainda dentro do loop, executa a = a + 3 = 5 + 3 = 8.

Fim do loop. Valores: a = 8, b = 20. Volta pro teste do while.

Teste: a < 10 and b > 14. 8 < 10 and 20 > 14. Verdadeiro. Entra no while.

Depois, vem um if: if a <> b. Realiza o teste: a <> b. 8 <> 20.

Verdadeiro. Entra no if, e cai em um outro if: if a > 5. Realiza o teste: a > 5. 8 > 5.

Verdadeiro. Entra no "então", e imprime "8 20"

Fora dos ifs, mas ainda dentro do loop, executa a = a + 3 = 8 + 3 = 11.

Fim do loop. Valores: a = 11, b = 20. Volta pro teste do while.

Falso. Sai do while.

Note que, em todo o processo, somente houve uma impressão: 8 20.

Gabarito: Letra C.

4. (FGV – 2018 – Prefeitura de Niterói – RJ) Sabendo-se que a função retorna o número de elementos de um array e que L assume o tipo de um array de inteiros, indexados a partir de zero, analise o pseudocódigo a seguir.

```
L := {10,2,40,53,28,12};
trocou := True;
while trocou {
    trocou := False;
    for k := 0 to len(L) - 2 {
        if L[k] > L[k+1] {
            L[k] = L[k+1];
            L[k+1] = L[k];
            trocou = True;
        }
    }
}
print L
```

Esse algoritmo deveria ordenar os elementos do array em ordem crescente, mas há problemas no código que produzem resultados errôneos. Assinale a opção que indica o que é de fato printado ao final da execução do código mostrado.

a) {10,2,40,53,28,12}
b) {2,10,12,28,40,53}
c) {53,40,28,12,10,2}
d) {2,2,12,12,12,12}
e) {2,10,10,10,10,12}

Comentários:

Primeiro, algumas considerações.

- As posições no array são contadas a partir do 0 neste algoritmo. Ou seja, a primeira posição é o 0, a segunda é o 1, e assim por diante.
- Era para o algoritmo ser o Bubble Sort, um famoso algoritmo de ordenação. O erro do algoritmo é não usar uma variável de troca na hora de inverter os valores.
  L[k] = L[k+1];
  L[k+1] = L[k];

  Deveria ser:

  aux = L[k];
  L[k] = L[k+1];
  L[k+1] = aux;

  Como resultado, ele acaba pegando o valor da próxima posição e colocando na atual; mas o valor da próxima posição se mantém intacto.

Dito isso, vamos analisar o algoritmo passo-a-passo.

Inicializa o array L = 10,2,40,53,28,12.

Inicializa a variável trocou = true.

Então, entra no while while trocou.

Realiza o teste. trocou é true, então entra no while.

Iteração do while número 1

Define trocou = false. Agora, executa o for k := 0 to len(L) - 2. Vai rodar um loop armazenando na variável k a cada iteração o valor de 0 até 4.

Para k = 0:

Testa L[k] > L[k+1], ou seja, L[0] > L[1], ou seja, 10 > 2.

Verdadeiro. Entra no if.

L[k] = L[k+1], ou seja, L[0] = L[1], ou seja, L[0] = 2.

L[k+1] = L[k], ou seja, L[1] = L[0], ou seja, L[1] =

10. Define trocou = true.

Valor de L = 2,2,40,53,28,12.

Para k = 1:

Testa L[k] > L[k+1], ou seja, L[1] > L[2], ou seja, 2 > 40.

Falso. Não entra no loop.

Valor de L = 2,2,40,53,28,12.

Para k = 2:

Testa L[k] > L[k+1], ou seja, L[2] > L[3], ou seja, 40 >

53. Falso. Não entra no loop.

Valor de L = 2,2,40,53,28,12.

Para k = 3:

Testa L[k] > L[k+1], ou seja, L[3] > L[4], ou seja, 53 >

28. Verdadeiro. Entra no if.

L[k] = L[k+1], ou seja, L[3] = L[4], ou seja, L[3] = 28.

L[k+1] = L[k], ou seja, L[4] = L[3], ou seja, L[4] =

53. Define trocou = true.

Valor de L = 2,2,40,28,28,12.

Para k = 4:

Testa L[k] > L[k+1], ou seja, L[4] > L[5], ou seja, 28 >

12. Verdadeiro. Entra no if.

L[k] = L[k+1], ou seja, L[4] = L[5], ou seja, L[4] = 12.

L[k+1] = L[k], ou seja, L[5] = L[4], ou seja, L[5] =

Valor de L = 2,2,40,28,12,12.

Volta para o while.

Realiza o teste. trocou é true, então entra no while.

Iteração do while número 2

Define trocou = false. Agora, executa o for k := 0 to len(L) - 2. Vai rodar um loop armazenando na variável k a cada iteração o valor de 0 até 4.

Para k = 0:

Testa L[k] > L[k+1], ou seja, L[0] > L[1], ou seja, 2 >

2. Falso. Não entra no loop.

Valor de L = 2,2,40,28,12,12.

Para k = 1:

Testa L[k] > L[k+1], ou seja, L[1] > L[2], ou seja, 2 > 40.

Falso. Não entra no loop.

Valor de L = 2,2,40,28,12,12.

Para k = 2:

Testa L[k] > L[k+1], ou seja, L[2] > L[3], ou seja, 40 >

28. Verdadeiro. Entra no if.

L[k] = L[k+1], ou seja, L[2] = L[3], ou seja, L[2] = 28.

L[k+1] = L[k], ou seja, L[3] = L[2], ou seja, L[3] =

40. Define trocou = true.

Valor de L = 2,2,28,28,12,12.

Para k = 3:

Testa L[k] > L[k+1], ou seja, L[3] > L[4], ou seja, 28 >

12. Verdadeiro. Entra no if.

L[k] = L[k+1], ou seja, L[3] = L[4], ou seja, L[3] = 12.

L[k+1] = L[k], ou seja, L[4] = L[3], ou seja, L[4] =

28. Define trocou = true.

Valor de L = 2,2,28,12,12,12.

Para k = 4:

Testa L[k] > L[k+1], ou seja, L[4] > L[5], ou seja, 12 >

12. Falso. Não entra no loop.

Valor de L = 2,2,28,12,12,12.

Volta para o while.

Realiza o teste. trocou é true, então entra no while.

Iteração do while número 3

Define trocou = false. Agora, executa o for k := 0 to len(L) - 2. Vai rodar um loop armazenando na variável k a cada iteração o valor de 0 até 4.

Para k = 0:

Testa L[k] > L[k+1], ou seja, L[0] > L[1], ou seja, 2 >

2. Falso. Não entra no loop.

Valor de L = 2,2,28,12,12,12.

Para k = 1:

Testa L[k] > L[k+1], ou seja, L[1] > L[2], ou seja, 2 > 28.

Falso. Não entra no loop.

Valor de L = 2,2,28,12,12,12.

Para k = 2:

Testa L[k] > L[k+1], ou seja, L[2] > L[3], ou seja, 28 >

12. Verdadeiro. Entra no if.

L[k] = L[k+1], ou seja, L[2] = L[3], ou seja, L[2] = 12.

L[k+1] = L[k], ou seja, L[3] = L[2], ou seja, L[3] =

28. Define trocou = true.

Valor de L = 2,2,12,12,12,12.

Para k = 3:

Testa L[k] > L[k+1], ou seja, L[3] > L[4], ou seja, 12 >

12. Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Para k = 4:

Testa L[k] > L[k+1], ou seja, L[4] > L[5], ou seja, 12 >

12. Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Volta para o while.

Realiza o teste. trocou é true, então entra no while.

Iteração do while número 4

Define trocou = false. Agora, executa o for k := 0 to len(L) - 2. Vai rodar um loop armazenando na variável k a cada iteração o valor de 0 até 4.

Para k = 0:

Testa L[k] > L[k+1], ou seja, L[0] > L[1], ou seja, 2 >

2. Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Para k = 1:

Testa L[k] > L[k+1], ou seja, L[1] > L[2], ou seja, 2 > 12.

Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Para k = 2:

Testa L[k] > L[k+1], ou seja, L[2] > L[3], ou seja, 12 >

12. Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Para k = 3:

Testa L[k] > L[k+1], ou seja, L[3] > L[4], ou seja, 12 >

12. Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Para k = 4:

Testa L[k] > L[k+1], ou seja, L[4] > L[5], ou seja, 12 >

12. Falso. Não entra no loop.

Valor de L = 2,2,12,12,12,12.

Note que, nesta iteração, ele não entrou no if nenhuma das vezes. Logo, não definiu trocou = true.

Então, ele volta pro while com a variável trocou = false, e então sai do while.

Valor final de L = 2,2,12,12,12,12.

Gabarito: Letra D.

5. (FGV – 2016 – SEE-PE) Analise o trecho de pseudocódigo exibido a seguir.

```
x = 1;
while x < 10
{
    y = x+2;
    while y < 10
    {
        print x, y;
        y = y + 1;
    };
    x = x+1;
};
```

De acordo com o pseudocódigo acima, assinale a opção que indica o número de vezes que o comando print é executado.

a) 28
b) 56
c) 60
d) 84
e) 100

Comentários:

- O algoritmo itera x e y sempre do valor inicial até 9.

- Primeiro ele itera incrementando o valor de x; e dentro da iteração do x, ele itera y, incrementando o valor de y, e printando os valores de x e y.
    - Dessa forma, a 1ª iteração de x irá de 1 até 9.
        - Como y = x + 2, então a 1ª iteração de y será de 3 até 9.
        - Portanto, ele entrará no print 7 vezes.
    - Na 2ª iteração do x, irá de 2 até 9.
        - Como y = x + 2, então a 2ª iteração de y será de 4 até 9.
        - Portanto, ele entrará no print 6 vezes.
    - Na 3ª iteração do x, irá de 3 até 9.
        - Como y = x + 2, então a 3ª iteração de y será de 5 até 9.
        - Portanto, ele entrará no print 5 vezes.
    - Na 4ª iteração do x, irá de 4 até 9.
        - Como y = x + 2, então a 4ª iteração de y será de 6 até 9.
        - Portanto, ele entrará no print 4 vezes.
    - Na 5ª iteração do x, irá de 5 até 9.
        - Como y = x + 2, então a 5ª iteração de y será de 7 até 9.
        - Portanto, ele entrará no print 3 vezes.
    - Na 6ª iteração do x, irá de 6 até 9.
        - Como y = x + 2, então a 6ª iteração de y será de 8 até 9.
        - Portanto, ele entrará no print 2 vezes.
    - Na 7ª iteração do x, irá de 7 até 9.
        - Como y = x + 2, então a 7ª iteração de y será de 9 até 9. Ou seja, apenas 9 mesmo.
        - Portanto, ele entrará no print 1 vez.
    - Na 8ª iteração do x, irá de 8 até 9.
        - Como y = x + 2, então y inicial seria 10. Não entra no while. Não printa nada.
    - Na 9ª iteração do x, irá de 9 até 9.
        - Como y = x + 2, então y inicial seria 11. Não entra no while. Não printa nada.

Os prints seriam os seguintes:

1 3

1 4

1 5

1 6

1 7

1 8

1 9

2 4

2 6

2 7

2 8

2 9

3 5

3 6

3 7

3 8

3 9

4 6

4 7

4 8

4 9

5 7

5 8

5 9

6 8

6 9

7 9

Ou seja, 28 vezes.

Gabarito: Letra A.

6. (FGV – 2016 – Prefeitura de Paulínia – SP) Analise o pseudocódigo de uma função recursiva exibido a seguir.

```
function f(x as integer) as integer:
    if x = 1 then
        return 1
    else
        return x + f(x-1)
```

Assinale a opção que indica o valor retornado para f (9).

a) 1
b) 29
c) 36
d) 45
e) 55

Comentários:

Trata-se de uma questão de recursividade. Assim, temos que ir calculando passo-a-passo o fluxo do programa, e depois retornar somando os resultados das funções recursivas.

Vamos, então:

Entrou na função com f(9).

Testa o if x = 1.

Como x é 9, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 9 + f(8)

Entrou na função com f(8).

Testa o if x = 1.

Como x é 8, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 8 + f(7)

Entrou na função com f(7).

Testa o if x = 1.

Como x é 7, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 7 + f(6)

Entrou na função com f(6).

Testa o if x = 1.

Como x é 6, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 6 + f(5)

Entrou na função com f(5).

Testa o if x = 1.

Como x é 5, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 5 + f(4)

Entrou na função com f(4).

Testa o if x = 1.

Como x é 4, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 4 + f(3)

Entrou na função com f(3).

Testa o if x = 1.

Como x é 3, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 3 + f(2)

Testa o if x = 1.

Como x é 2, entra no "senão".

Vai retornar recursivamente return x + f(x - 1), ou seja, return 2 + f(1)

Entrou na função com f(1).

Testa o if x = 1.

Como x é 1, entra no "então". f(1) = 1. Agora, vai voltar em cada uma das recursões.

Volta da recursão em que houve return 2 + f(1). Assim, retorna 2 + 1 = 3.

Volta da recursão em que houve return 3 + f(2). Assim, retorna 3 + 3 = 6.

Volta da recursão em que houve return 4 + f(3). Assim, retorna 4 + 6 = 10.

Volta da recursão em que houve return 5 + f(4). Assim, retorna 5 + 10 = 15.

Volta da recursão em que houve return 6 + f(5). Assim, retorna 6 + 15 = 21.

Volta da recursão em que houve return 7 + f(6). Assim, retorna 7 + 21 = 28.

Volta da recursão em que houve return 8 + f(7). Assim, retorna 8 + 28 = 36.

Volta da recursão em que houve return 9 + f(8). Assim, retorna 9 + 36 = 45.

Gabarito: Letra D.

7. (FGV – 2016 – SEE-PE) Analise a função ff definida pelo trecho de pseudocódigo exibido a seguir.

```
function ff (N as integer) as integer
{
    if N = 5 then
    {   return N;
    }
    else
    {   return N + ff(N-1);
    };
}
```

Assinale a opção que indica o valor correto da expressão ff(12).

a) 12
b) 17
c) 68
d) 72
e) 78

Comentários:

Trata-se de uma questão de recursividade. Assim, temos que ir calculando passo-a-passo o fluxo do programa, e depois retornar somando os resultados das funções recursivas.

Entrou na função com ff(12).

Testa o if N = 5.

Como N é 12, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 12 + ff(11)

Entrou na função com ff(11).

Testa o if N = 5.

Como N é 11, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 11 + ff(10)

Testa o if N = 5.

Como N é 10, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 10 + ff(9)

Entrou na função com ff(9).

Testa o if N = 5.

Como N é 9, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 9 + ff(8)

Entrou na função com ff(8).

Testa o if N = 5.

Como N é 8, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 8 + ff(7)

Entrou na função com ff(7).

Testa o if N = 5.

Como N é 7, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 7 + ff(6)

Entrou na função com ff(6).

Testa o if N = 5.

Como N é 6, entra no "senão".

Vai retornar recursivamente return N + ff(N - 1), ou seja, return 6 + ff(5)

Testa o if N = 5.

Como N é 5, finalmente entra no "então". ff(N) = N, ou seja, ff(5) = 5. Agora, vai voltar em cada uma das recursões.

Volta da recursão em que houve return 6 + ff(5). Assim, retorna 6 + 5 = 11. Volta da

recursão em que houve return 7 + ff(6). Assim, retorna 7 + 11 = 18. Volta da

recursão em que houve return 8 + ff(7). Assim, retorna 8 + 18 = 26. Volta da

recursão em que houve return 9 + ff(8). Assim, retorna 9 + 26 = 35. Volta da

recursão em que houve return 10 + ff(9). Assim, retorna 10 + 35 = 45. Volta da

recursão em que houve return 11 + ff(10). Assim, retorna 11 + 45 = 56. Volta da

recursão em que houve return 12 + ff(11). Assim, retorna 12 + 56 = 68. Portanto,

ff(12) = 68.

Gabarito: Letra C.

8. (FGV – 2017 – SEPOG-RO) Considere o algoritmo em pseudocódigo descrito a seguir.

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

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

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

Comentários:

Vamos analisar este algoritmo.

Define v = 1.

Entra no "para i=0 até n". Ele vai executar o trecho abaixo 9 vezes.

Define j = 1

Testa o enquanto j<n, ou seja, 1 < 9

Entra no loop, pois 1 < 9

Define j = 2*j = 2*1 = 2

Toda o "para k = 0 até j". Ou seja, como j = 2, executou f 3x. Total: 3

Entra no loop, pois 2 < 9

Define j = 2*j = 2*2 = 4

Toda o "para k = 0 até j". Ou seja, como j = 4, executou f 5x. Total: 8

Entra no loop, pois 4 < 9

Define j = 2*j = 2*4 = 8

Toda o "para k = 0 até j". Ou seja, como j = 8, executou f 9x. Total: 17

Agora, ele volta para o "para i=0 até n". Ele vai executar esse mesmo trecho 9 vezes, já que i vai de 0 a 8. Como, em cada vez, ele executa f 17 vezes, então 9*17 = 153.

Portanto, vai executar 153 vezes.

Gabarito: Letra B.

9. (FGV – 2016 – SEE-PE) Analise o trecho de pseudocódigo a seguir.

```
function f(a as integer, b as integer, c as
integer) as integer
{
    while b <= c
    {
        a = b+c;
        y = b+1;
    };
    return a+b+c;
};
x=10;
y=2;
z=3;
print f(x, y, z), x, y, z;
```

Em algumas linguagens de programação é possível fazer a passagem de parâmetros como *byref* ou *byvalue*, e assim podemos supor que a função f acima poderia ser reescrita especificando, para cada parâmetro, uma das duas formas citadas.

Supondo-se que o primeiro e o segundo parâmetro da função tenham sido passados como *byref* e o terceiro, como *byvalue*, os quatro valores exibidos pelo comando print seriam, respectivamente,

a) 13, 6, 2 e 3.
b) 13, 6, 4 e 3.
c) 10, 10, 2 e 3.
d) 13, 0, 0 e 3.
e) 10, 8, 5 e 2.

Comentários:

Vamos analisar este algoritmo.

Ele entra na funcão f passando parâmetros f(x, y, z) = f(10, 2, 3), sendo os dois primeiros por referência (*byref*), e o último por valor (*byvalue*).

Quando você passa um valor por referência, ele é atualizado na variável original. Quando você passa por valor, a variável original não é alterada.

Também consideremos que a função utiliza o valor de y, o valor global. Como y é passado para b por referência, as duas variáveis podem ser tratadas como a mesma coisa.

Dito isso, vamos seguir o algoritmo a partir da chamada da função:

Testar o while "while b <= c"

b <= c" é "2 <= 3. Verdadeiro, então entra no loop. a

= b + c = 2 + 3 = 5

y = b + 1 = 2 + 1 = 3

b <= c" é "3 <= 3. Verdadeiro, então entra no loop. a

= b + c = 3 + 3 = 6

y = b + 1 = 3 + 1 = 4

Volta pro while. "b <= c" é "4 <= 3". Falso, então sai do loop.

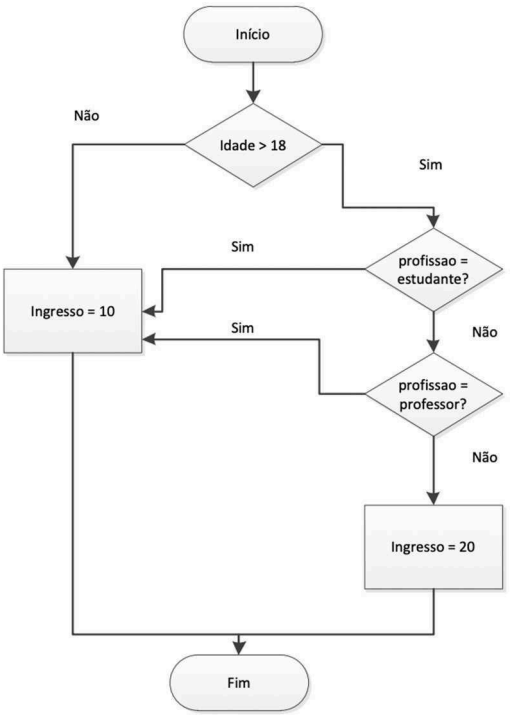
Valores finais: a = 6, b = 4 e c = 3. Como a e b foram passados por referência, eles voltam para os valores de x e y passados na função. Como c foi passado por valor, ele não volta para z. Mas tanto faz, já que o valor final de c foi 3, e o valor de y era 3 também.

O retorno de f(x, y, z) = f(6, 4, 3) = 6 + 4 + 3 = 13.

Vai imprimir: 13 6 4 3

Gabarito: Letra B.

10. (FGV – 2014 – SUSAM) A figura a seguir apresenta um fluxograma de uma função que fornece ao usuário o valor do ingresso que deverá ser cobrado para a entrada no cinema



Os parâmetros de entrada da função são sua IDADE e PROFISSÃO. A conversão deste fluxograma em pseudolinguagem de programação é:

(A)
```
SE (IDADE < 18) OU (PROFISSÃO = ESTUDANTE) OU
(PROFISSÃO = PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
FIM SE
```

(B)
```
SE (IDADE < 18) E (PROFISSÃO = ESTUDANTE) E
(PROFISSÃO = PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
FIM SE
```

(C)
```
SE (IDADE < 18) E [(PROFISSÃO = ESTUDANTE) OU
(PROFISSÃO = PROFESSOR)] ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
FIM SE
```

(D)
```
SE [(IDADE < 18) OU (PROFISSÃO = ESTUDANTE)] E
(PROFISSÃO = PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
FIM SE
```

(E)
```
SE (IDADE < 18) OU (PROFISSÃO <> ESTUDANTE) OU
(PROFISSÃO <> PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
FIM SE
```

Comentários:

Vamos analisar o algoritmo.

No início, ele verifica se a idade é maior do que 18. Se for menor de 18, ele já manda ingresso a 10 reais.

Se for sim, ele verifica se a profissão é estudante. Se for sim, ele também já manda ingresso a 10 reais. Mas, se for não, ele verifica outra possibilidade, que é a profissão ser professor. Neste caso, se for professor, ele manda ingresso a 10 reais. Do contrário, o ingresso é a 20 reais.

Note que, se ele for menor de 18, já tem ingresso de 10 reais.

Agora, se não for, mas for estudante, ele tem ingresso a 10 reais.

E, se não for estudante, ele tem ingresso a 10 reais também.

Ou seja, a condição para ter ingresso a 10 reais é uma dessas 3 possibilidades! Neste caso, idade < 18 OU profissao = estudante OU profissao = professor.

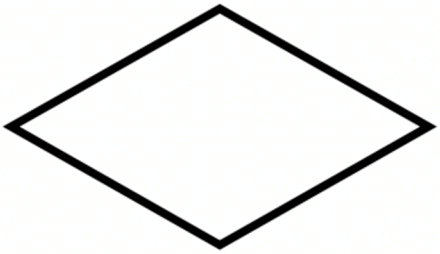
Neste caso, a correta é letra A, pois faz justamente esse condicional para definir ingresso a 10 reais.

Gabarito: Letra A.

11. (FGV – 2013 – TCE-BA) No mapeamento de um processo, o fluxograma tem papel relevante, pois ele descreve e mapeia as etapas do processo de forma lógica e planejada. Com o auxílio do fluxograma, pode-se eliminar erros, evitar duplicidades, eliminar tarefas desnecessárias, otimizar o ciclo etc. Em um fluxograma, o símbolo que indica uma decisão é

a) 

b) 

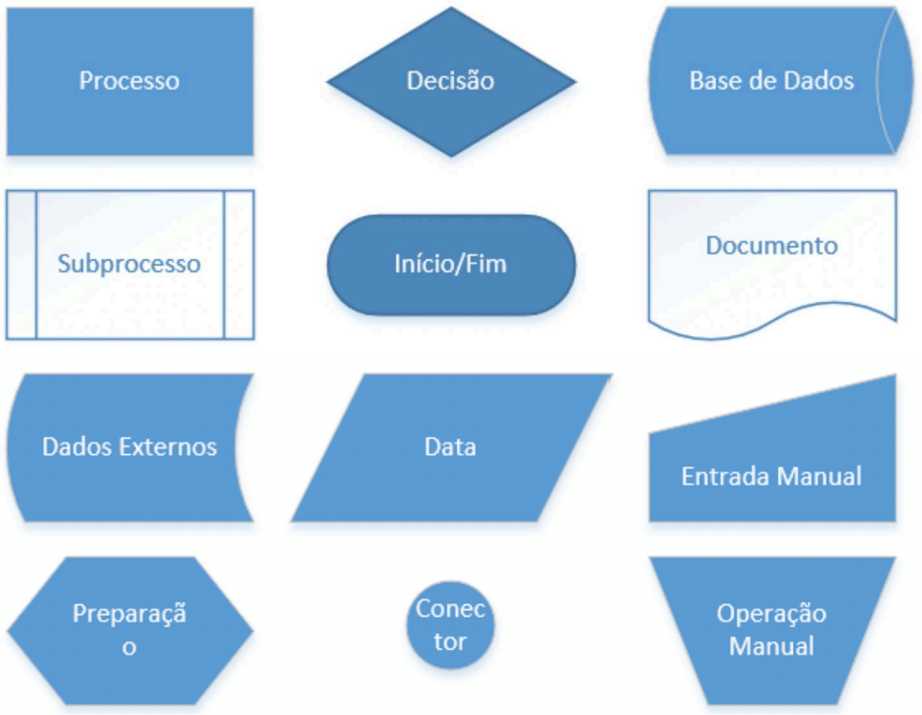
c) 

d) 

e)

Comentários:

Os símbolos de fluxograma são os seguintes:



Portanto, o de decisão é o C.

Gabarito: Letra C.

QUESTÕES COMENTADAS – MULTIBANCAS

1. (Quadrix – 2022 – CRC-PR) A respeito dos diagramas de casos de uso, dos diagramas de classe, da análise essencial e da lógica de programação, julgue o item a seguir

Na lógica de programação, um algoritmo é conceituado como uma sequência estruturada e organizada de passos que tem por objetivo atingir um objetivo, seja ele definido ou indefinido.

Comentários:

De fato, na lógica de programação, um algoritmo é uma sequência estruturada e organizada de passos para atingir um objetivo. Mas esse objetivo é sempre definido, e nunca indefinido. Gabarito: Errado.

2. (SELECON – 2022 – Prefeitura de Pontes e Lacerda) O algoritmo abaixo utiliza os conceitos de passagem de parâmetros, sendo de SD para X por referência; de NR para Y e de VL para W por valor.

```
algoritmo "PREF_PL"
var
   SD : caracter
   NR : inteiro
   VL : logico
procedimento PROC_PL(var X:caracter;Y:inteiro;W:logico)
inicio
      X <- "BURITI"
      Y <- 2021
      W <- VERDADEIRO
fimprocedimento
inicio
      SD <- "GUAPORÉ"
      NR <- 2023
      VL <- FALSO
      PROC_PL(SD,NR,VL)
      se VL = VERDADEIRO entao
         NR <- 2022
      fimse
      escreval("SD = ",SD:10," NR = ",NR:4," VL = ",VL)
fimalgoritmo
```

Nessas condições, ao final da execução, os valores para SD, NR e VL serão, respectivamente:

a) GUAPORÉ, 2023 e VERDADEIRO

b) BURITI, 2022 e VERDADEIRO
c) GUAPORÉ, 2022 e FALSO
d) BURITI, 2023 e FALSO

Comentários:

Se a passagem é por referência, significa que, ao fim da execução do procedimento, o valor da variável sendo passada no procedimento vai mudar, caso seja manipulada. Se for por valor, o valor da variável não muda.

Dito isso, vamos analisar o algoritmo:

As variáveis SD, NR e VL são declaradas como variáveis globais.

Depois, o algoritmo inicializa as variáveis SD = "GUAPORÉ", NR = 2023 e VL = FALSO.

Então, é chamado o PROC_PL passando as variáveis:

- Para o parâmetro X, é atribuída a variável SD, por referência; ou seja, se o valor de X mudar, o valor de SD muda também.
- Para o parâmetro Y, é atribuída a variável NR, por valor; ou seja, se o valor de Y mudar, o de NR não irá mudar.
- Para o parâmetro W, é atribuída a variável VL, pro valor; ou seja, se o valor de W mudar, o de VL não mudará.

O procedimento PROC_PL altera as variáveis X, Y e W. Mas a única que importa é a variável X, a que é atribuído o valor "BURITI". Significa que a variável SD também será "BURITI".

Depois, se o valor de VL for verdadeiro, atribui a NR o valor 2022. Mas VL foi passada por valor, e mantém o seu valor inicial, que é FALSO. Não entra no if.

Por fim, escreve os valores na tela. SD = "BURITI", NR = 2023. VL = FALSO. Gabarito: Letra D.

3. (SELECON – 2022 – Prefeitura de Pontes e Lacerda-MT) No que se refere à lógica de programação, observe o pseudocódigo abaixo, referente a um algoritmo que contém uma função recursiva.

```
algoritmo "PL_2022"
var
    CT, RP : inteiro
funcao FREC(AUX:inteiro):inteiro
inicio
   se  (AUX = 0) OU (AUX = 1)  entao
       retorne AUX
   senao
       retorne FREC(AUX-2) + FREC(AUX-1)
   fimse
fimfuncao
inicio
para CT de 0 ate 5 faca
     RP <- FREC(CT)
     escreva(RP:3)
fimpara
fimalgoritmo
```

Após a execução, a sequência de números de saída é:

a) 0 1 1 2 3
b) 1 1 2 3 5
c) 0 1 1 2 3 5
d) 1 1 2 3 5 8

Comentários:

Nessa questão, se a gente for seguir o algoritmo inteiro para resolvê-la, a gente vai perder muito tempo e ficar confuso. É um dos casos em que é necessário identificar por eliminação, até pra resolver a questão mais rápido.

Observe que o comando escreva(RP:3) se encontra dentro de um loop que para...até, que roda de 0 a 5, ou seja, 6 vezes. Significa que serão impressos 6 números em tela.

Assim, a gente já elimina as alternativas a e b.

Agora, vamos ao algoritmo:

- Declara as variáveis CP e RP como inteiros, e globais.
- Entra no "para CT de 0 até 5 faça".
- Para CT = 0:
    - Executa RP <- FREC(CT) = FREC(0)
    - Entra na função FREC(0).
        - Testa (AUX = 0) OU (AUX = 1), ou seja, (0 = 0) OU 0 = 1), ou seja, true OU false.
        - Verdadeiro, entra no "então" e retorna o valor de AUX = 0. O valor de AUX é retornado ao RP.

- Escreve o valor de RP, que é 0.

Pronto. Agora já sabemos que a resposta da questão é a letra C, por eliminação, pois é a única que começa com 0, e que imprime 6 valores. Gabarito: Letra D.

4. (UFV – 2022 – UFV-MG) Considere o algoritmo em pseudocódigo a seguir

```
← é o comando de atribuição
% é o resto de divisão inteira

início
        inteiro x
        inteiro y
        inteiro z
        x ← 100
        z ← 3
        x ← x + 40
        y ← 5 ^ z * 4
        enquanto (z <= 7) faça
                x ← x – 15
                se x % 2 = 0 então
                        z ← z + 1
                se não
                        z ← z + 2
                fim se
                y ← y + z - x
        fim enquanto
        escreva x, y, z
fim
```

Assinale a alternativa que apresenta CORRETAMENTE os valores impressos pelo algoritmo:

a) 95, 276, 8.
b) 95, 189, 8.
c) 80, 189, 7.
d) 110, 276, 6.

Comentários:

Vamos analisar o código linha a linha.

As variáveis x, y e z são declaradas como inteiro.

Atribui x = 100.

Atribui z = 3.

x = x + 40 = 100 + 40 = 140

y = 5^{z}*4 = 5^{3}*4 = 5*5*5*4 = 125*4 = 500

Entra no "enquanto (z <= 7) faça".

Como z <= 7, ou seja, 3 <= 7 é verdadeiro, então segue no "enquanto".

x = x - 15 = 140 - 15 = 125

Chega no "if x % 2 = 0 então".

x % 2 = 125 % 2 = 1 = 0, é falso, entra no "senão".

z = z + 2 = 3 + 2 = 5.

y = y + z - x = 500 + 5 - 125 = 380

Valores parciais: x = 125 y = 380 z = 5. Volta para o "enquanto".

Como z <= 7, ou seja, 5 <= 7 é verdadeiro, então segue no "enquanto".

x = x - 15 = 125 - 15 = 110

Chega no "if x % 2 = 0 então".

x % 2 = 110 % 2 = 0 == 0, é verdadeiro, entra no "então".

z = z + 1 = 5 + 1 = 6.

y = y + z - x = 380 + 6 - 110 = 276

Valores parciais: x = 110 y = 276 z = 6. Volta para o "enquanto".

Como z <= 7, ou seja, 6 <= 7 é verdadeiro, então segue no "enquanto".

x = x - 15 = 110 - 15 = 95

Chega no "if x % 2 = 0 então".

x % 2 = 95 % 2 = 1 = 0, é falso, entra no "senão".

z = z + 2 = 6 + 2 = 8.

y = y + z - x = 276 + 8 - 95 = 189

Valores parciais: x = 95 y = 189 z = 8. Volta para o "enquanto".

Como z <= 7, ou seja, 8 <= 7 é falso, então sai do "enquanto".

Portanto, ele escreve: 95 189 8. **Gabarito: Letra B.**

5. (UFV – 2022 – UFV-MG) Considere o algoritmo em pseudocódigo a seguir

```
← é o comando de atribuição
o índice da primeira posição do vetor é 1

início
        inteiro vetor[ ] = {91, 47, 61, 87, 29}
        inteiro x
        inteiro i
        i ← 1
        enquanto (i < 5) faça
            se (vetor[i] > vetor[i+1]) então
                    x ← vetor[i]
                    vetor[i] ← vetor[i +1]
                    vetor[i+1] ← x
             fim se
            i ← i + 1
        fim enquanto
        i ← 1
        enquanto (i < 6) faça
                escreva vetor[i]
               i ← i + 1
        fim enquanto
fim
```

Assinale a alternativa que apresenta CORRETAMENTE a sequência de valores impressos pelo algoritmo:

a) 91, 47, 61, 87, 29.
b) 91, 61, 47, 87, 29.
c) 51, 47, 87, 29, 91.
d) 47, 61, 87, 29, 91.

Comentários:

Novamente, temos um código que você precisa saber identificar de cara, pois aparece muito. É o Bubble Sort, cuja função é ordenar os valores de um vetor.

Só que tem um detalhe. Nessa versão do Bubble Sort, há um problema. Ele itera o valor de i de 1 a 4, e não de 1 a 5. Dessa forma, ele não vai executar o Bubble Sort por completo, faltando uma vez.

Vamos seguir o algoritmo passo a passo.

Inicializa o vetor = [91, 47, 61, 87, 29].

Chega no “enquanto (i < 5) faça”. Como i < 5, ou seja, 1 < 5, entra no "enquanto".

Chega no "se (vetor[i] > vetor[i + 1]) então".

Como vetor[1] > vetor[2], ou seja, 91 > 47 é verdadeiro, entra no "então".

Então troca os valores:

x = vetor[1] = 91

vetor[1] = vetor[2] = 47

vetor[2] = x = 91

i = i + 1 = 1 + 1 = 2

Valores parciais do vetor: [47,91,61,87,29]. Volta pro "enquanto".

Como i < 5, ou seja, 2 < 5, entra no "enquanto".

Chega no "se (vetor[i] > vetor[i + 1]) então".

Como vetor[2] > vetor[3], ou seja, 91 > 61 é verdadeiro, entra no "então".

Então troca os valores:

x = vetor[2] = 91

vetor[2] = vetor[3] = 61

vetor[3] = x = 91

i = i + 1 = 2 + 1 = 3

Valores parciais do vetor: [47,61,91,87,29]. Volta pro "enquanto".

Como i < 5, ou seja, 3 < 5, entra no "enquanto".

Chega no "se (vetor[i] > vetor[i + 1]) então".

Como vetor[3] > vetor[4], ou seja, 91 > 87 é verdadeiro, entra no "então".

Então troca os valores:

x = vetor[3] = 91

vetor[3] = vetor[4] = 87

vetor[4] = x = 91

i = i + 1 = 3 + 1 = 4

Valores parciais do vetor: [47,61,87,91,29]. Volta pro "enquanto".

Como i < 5, ou seja, 4 < 5, entra no "enquanto".

Chega no "se (vetor[i] > vetor[i + 1]) então".

Como vetor[4] > vetor[5], ou seja, 91 > 29 é verdadeiro, entra no "então".

Então troca os valores:

x = vetor[4] = 91

vetor[4] = vetor[5] = 29

vetor[5] = x = 91

i = i + 1 = 4 + 1 = 5

Valores parciais do vetor: [47,61,87,29,91]. Volta pro "enquanto".

Como i < 5, ou seja, 5 < 5, é falso, então. sai "enquanto".

E imprime o valor do vetor inteiro, já que o loop itera de 1 até 5:

47 61 87 29 91

Gabarito: Letra D.

6. (UFSC – 2022 – UFSC) Considere o pseudocódigo do método de ordenação Insertion Sort, o qual ordena em ordem crescente os números naturais armazenados em um vetor (array) v de tamanho t indexado a partir de zero (ou seja, índices do vetor variam de 0 a t-1).

Assinale a alternativa que completa corretamente o espaço pontilhado entre chaves do pseudocódigo abaixo.

```
função Ordena(v, t)
{
   i ← 1
   enquanto i < t faça
   {
      j ← i
      enquanto j > 0 e v[j-1] > v[j] faça
      {
         ..........................
      }
      i ← i + 1
   }
}
```

a) x ← v[j]
   v[j] ← v[j - 1]
   v[j – 1] ← x
   j ← j + 1
b) x ← v[j]
   v[j] ← v[j - 1]

v[j – 1] ← x
j ← j - 1

c) x ← v[j]
v[j] ← v[j + 1]
v[j + 1] ← x
j ← j - 1

d) x ← v[j]
v[j] ← v[j + 1]
v[j + 1] ← x
j ← j + 1

e) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x
j ← j – 2

Comentários:

Novamente, é importantíssimo ter em mente os algoritmos de ordenação, pois caem muito!

Neste caso, o Insertion Sort, eu vou ter que considerar o seguinte:

- O algoritmo compara v[j-1] com v[1].
    - Isso significa que ele deverá trocar os valores de v[j] e de v[j – 1], e não de v[j] e v[j+1], como ocorre com o Bubble Sort.
- A variável j deve ser decrementada, e não incrementada, já que o segundo enquanto testa j > 0.
- Deve ser usada uma variável auxiliar, que no caso, aqui, será x. Ela é usada para trocar os valores de v[j] e v[j -1].

Vamos item a item:

a) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x
j ← j + 1

Não funciona, pois a variável j está sendo incrementada.

b) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x
j ← j - 1

Este é o correto, com todas as características que mencionei anteriormente.

c) x ← v[j]
v[j] ← v[j + 1]
v[j + 1] ← x
j ← j – 1

Estão sendo trocados v[j] com v[j+1]. Errado!

d) x ← v[j]
v[j] ← v[j + 1]
v[j + 1] ← x
j ← j + 1

Também não funciona, pois a variável j está sendo incrementada.

e) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x
j ← j – 2

Também não funciona, pois a variável j está decrementada com – 2. Gabarito: Letra B.

7. (IDECAN – 2021 – PEFOCE) Na construção de algoritmos e programas de computador, sendo x e y duas condições de teste, os operadores lógicos AND e OR são bastante utilizados nas estruturas de controle dos tipos seleção e repetição e correspondem às tabelasverdade mostradas, respectivamente, em

A)

x \ y	falso	verdadeiro
falso	V	V
verdadeiro	V	F

e

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	F

B)

x \ y	falso	verdadeiro
falso	F	F
verdadeiro	F	V

e

x \ y	falso	verdadeiro
falso	F	V
verdadeiro	V	V

C)

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	V

e

x \ y	falso	verdadeiro
falso	F	F
verdadeiro	F	V

D)

x \ y	falso	verdadeiro
falso	F	V
verdadeiro	V	V

e

x \ y	falso	verdadeiro
falso	V	V
verdadeiro	V	F

E)

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	F

e

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	V

Comentários:

Na tabela-verdade, o AND é verdadeiro apenas se as duas preposições forem verdadeiras; enquanto que o OR é verdadeiro se pelo menos uma das duas preposições forem verdadeiras.

Os diagramas que mostram isso estão na alternativa B. Gabarito: Letra B.

8. (CESPE/CEBRASPE – 2021 – SEED-PR) Assinale a opção que apresenta o resultado do algoritmo apresentado.

```
programa {
    funcao inicio() {
        inteiro vetor[] = { 81, 37, 51, 77, 19 }
        inteiro naosei
        logico achou = verdadeiro
        enquanto (achou)
        {
            achou=falso
            para (inteiro i = 0; i <4; i++)
            {
                se (vetor[i] > vetor[i+1])
                {
                    naosei = vetor[i]
                    vetor[i] = vetor[i+1]
                    vetor[i+1] = naosei
                    achou = verdadeiro
                }
            }
        }
        para (inteiro i = 0; i < 5; i++)
        {
            escreva (vetor[i]+ "\n")
        }
    }
}
```

a) 81
37
51
77
19

b) 81
51
37
17
19

c) 19
37
51
77
81

d) 51
81
37
77
19

e) 19
77
37
81
51

Comentários:

O que este algoritmo faz é ordenar os números do vetor, o *Bubble Sort*. É um algoritmo clássico e a melhor forma de resolver essa questão é conhecendo-o previamente, para já responder sem precisar nem passar passo-a-passo no algoritmo. Basta informar a opção em que o vetor está ordenado, que é a letra C.

De toda forma, vamos mostrar o passo-a-passo deste algoritmo:

Inicializa o vetor = [81,37,51,77,19]

Declara a variável auxiliar naosei.

Inicializa a variável achou = true

Testa o "enquanto (achou)".

Como achou = true, então entra no "enquanto".

Define achou = false.

Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.

Para i = 0:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 81 > 37

naosei = vetor[i] = vetor[0] = 81

Agora, troca os valores de vetor[0] e vetor[1].

vetor[0] = vetor[i + 1] = vetor[1] = 37

vetor[1] = naosei = 81

Define achou = true.

Valores do vetor: [37,81,51,77,19].

Para i = 1:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 81 > 51

naosei = vetor[i] = vetor[1] = 81

Agora, troca os valores de vetor[1] e vetor[2].

vetor[1] = vetor[i + 1] = vetor[2] = 51

vetor[2] = naosei = 81

Define achou = true.

Valores do vetor: [37,51,81,77,19].

Para i = 2:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 81 > 77

naosei = vetor[i] = vetor[2] = 81

Agora, troca os valores de vetor[2] e vetor[3].

vetor[2] = vetor[i + 1] = vetor[3] = 77

vetor[3] = naosei = 81

Define achou = true.

Valores do vetor: [37,51,77,81,19].

Para i = 3:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 81 > 19

naosei = vetor[i] = vetor[3] = 81

Agora, troca os valores de vetor[3] e vetor[4].

vetor[3] = vetor[i + 1] = vetor[4] = 19

vetor[4] = naosei = 81

Define achou = true.

Valores do vetor: [37,51,77,19,81].

Volta para o enquanto.

Como achou = true, então entra no "enquanto".

Define achou = false.

Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.

Para i = 0:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 37 > 51

Falso. Não entra no if.

Valores do vetor: [37,51,77,19,81].

Para i = 1:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 51 > 77

Falso. Não entra no if.

Valores do vetor: [37,51,77,19,81].

Para i = 2:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 77 > 19

naosei = vetor[i] = vetor[2] = 77

Agora, troca os valores de vetor[2] e vetor[3].

vetor[2] = vetor[i + 1] = vetor[3] = 19

vetor[3] = naosei = 77

Define achou = true.

Valores do vetor: [37,51,19,77,81].

Para i = 3:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81

Falso. Não entra no if.

Valores do vetor: [37,51,19,77,81].

Volta para o enquanto.

Como achou = true, então entra no "enquanto".

Define achou = false.

Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.

Para i = 0:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 37 > 51

Falso. Não entra no if.

Valores do vetor: [37,51,19,77,81].

Para i = 1:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 51 > 19

naosei = vetor[i] = vetor[1] = 51

Agora, troca os valores de vetor[1] e vetor[2].

vetor[1] = vetor[i + 1] = vetor[2] = 19

vetor[2] = naosei = 51

Define achou = true.

Valores do vetor: [37,19,51,77,81].

Para i = 2:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 51 > 77

Falso. Não entra no if.

Valores do vetor: [37,19,51,77,81].

Para i = 3:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81

Falso. Não entra no if.

Valores do vetor: [37,19,51,77,81].

Volta para o enquanto.

Como achou = true, então entra no "enquanto".

Define achou = false.

Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.

Para i = 0:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 37 > 19

naosei = vetor[i] = vetor[0] = 37

Agora, troca os valores de vetor[0] e vetor[1].

vetor[0] = vetor[i + 1] = vetor[1] = 19

vetor[1] = naosei = 37

Define achou = true.

Valores do vetor: [19,37,51,77,81].

Para i = 1:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 37 > 51

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Para i = 2:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 51 > 77

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Para i = 3:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Volta para o enquanto.

Como achou = true, então entra no "enquanto".

Define achou = false.

Entra no "para", que irá rodar de 0 a 3, cada iteração com um valor para a variável i.

Para i = 0:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[0] > vetor[1], ou seja, se 19 > 37

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Para i = 1:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[1] > vetor[2], ou seja, se 37 > 51

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Para i = 2:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[2] > vetor[3], ou seja, se 51 > 77

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Para i = 3:

Testa o if: se vetor[i] > vetor [i+1], ou seja, se vetor[3] > vetor[4], ou seja, se 77 > 81

Falso. Não entra no if.

Valores do vetor: [19,37,51,77,81].

Volta para o enquanto.

Como achou = false, então sai do "enquanto".

Valor final do vetor = [19,37,51,77,81].

Gabarito: Letra C.

9. (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

10. (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

11. (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

12. (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

13. (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

14. (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**

15. **(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

16. (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

17. (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**

18. **(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

19. (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

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

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

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.

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

21. (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

22. (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

23. (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

24. (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

LISTA DE QUESTÕES – CESPE

1. (CESPE / CEBRASPE – 2022 – BANRISUL) Em um algoritmo, todo resultado de uma operação de entrada de dados é armazenado em uma posição na memória.

2. (CESPE / CEBRASPE – 2022 – BANRISUL) Os laços usados em estruturas de repetição e teste podem ser feitos por meio de comandos como enquanto e repita.

3. (CESPE / CEBRASPE – 2022 – BANRISUL) O fluxograma é uma das formas de se representarem as instruções de um programa, utilizando-se de alguns comandos genéricos ou primitivos para a interpretação do algoritmo.

4. (CESPE / CEBRASPE – 2022 – BANRISUL) As estruturas se e senão são estruturas de repetição utilizadas nas situações em que, caso determinada condição seja alcançada, um comando é realizado, caso contrário, outro comando é executado.

5. (CESPE/CEBRASPE – 2022 – POLITEC - RO) O autômato finito determinístico:
   a) corresponde à função de transição que recebe um estado ou um símbolo de entrada que sempre retorna um conjunto de estados como resultado.
   b) tem a capacidade de adivinhar algo sobre sua entrada ao testar valores.
   c) pode, para cada entrada, transitar a partir do seu estado atual em um e somente um estado.
   d) permite zero, uma ou n transições para os estados de entrada.
   e) consegue estar em vários estados ao mesmo tempo.

6. (CESPE/CEBRASPE – 2022 – SEED-PR)

```
programa
{
    funcao inicio()
    {
        inteiro numero, atual = 1, valor = 1
        numero = 6
        enquanto (atual <= numero)
        {
            valor = valor * atual
            atual = atual + 1
        }
        escreva(valor)
    }
}
```

O resultado do algoritmo precedente é:

a) 2
b) 6
c) 24
d) 120
e) 720

7. (CESPE / CEBRASPE – 2021 – SEED-PR) Considerando a, b, c e d como variáveis com valores iniciais iguais a 5, 7, 3 e 9, respectivamente, assinale a opção correta.

a) O resultado da expressão (a != 3 || b < 10 || c == 5) é falso.
b) O resultado da expressão (d > 8 && c == 3 || a >= 10) é verdadeiro.
c) O resultado da expressão !(d == 12 && a != 10) é falso.
d) O resultado da expressão (c == 4 || d <=6) && (a >= 5 && b !=9) || (! (a < 5)) é falso.
e) O resultado da expressão (a == 3 || b > 10 || d == 8) é verdadeiro.

8. (CESPE / CEBRASPE – 2021 – SEED-PR)

```
1. var
2. valores: vetor [1..5] de inteiro
3. resultado, x: real
4.
5. inicio
6. para i de 1 ate 5 faca
7.    leia(valores)
8.    x<- x + valores
9. fimpara
10. resultado <- x / 5
11. escreva("Resultado: ", resultado)
12. fimalgoritmo
```

O resultado da lógica dos algoritmos precedente é a:

a) média dos valores da matriz vetor.
b) soma dos valores de 1 a 5, ou seja, 15.
c) média dos valores de 1 a 5, ou seja, 3.
d) soma dos valores da matriz vetor.
e) ordenação dos valores de 1 a 5.

9. (CESPE / CEBRASPE – 2021 – SEED-PR)

```
1. var
2. num: inteiro
3. inicio
4. leia(num)
5. se (num % 2) =/= 0 entao
6. escreva("X")
7. senao
8. escreva("Y")
9. fimse
10. fimalgoritmo
```

Considerando-se a lógica do algoritmo anterior, é correto afirmar que, para todo valor de num (linha 4):

a) maior que 2, escreve Y.
b) menor que 2, escreve X.
c) igual a 2, escreve Y.
d) igual a 2, escreve X.
e) diferente de 2, escreve X.

10.(CESPE / CEBRASPE – 2021 – SEED-PR)

```
var
valor: real
resultado: real
inicio
soma <- 0
escreva ("digite um valor")
leia (valor)
enquanto valor <> 0 faca
      resultado <- resultado + valor
   escreva ("digite um valor")
   leia (valor)
fimenquanto
escreva ("resultado: ", resultado)
fimalgoritmo
```

Considere que, na lógica do algoritmo apresentado, sejam inseridos sequencialmente os valores a seguir: 3 7 5 0 1 2

Nessa situação, o resultado da execução será

a) 5.
b) 18.
c) 375012
d) 15.
e) 16.

11.(CESPE / CEBRASPE – 2021 – SEED-PR)

```
1. var
2. cont,n,resultado:inteiro
3. inicio
4. resultado <-1
5. leia(n)
6. para cont de 1 ate n passo 1 faca
7.    resultado <- resultado *cont
8. fimpara
9. escreva(resultado)
10. fimalgoritmo
```

.

Em relação à lógica do algoritmo precedente, caso o valor de n (linha 5) seja igual a:

a) 4, o resultado será 4.
b) 3, o resultado será 5
c) 2, o resultado será 12.
d) 6, o resultado será 30.
e) 5, o resultado será 120.

12. (CESPE / CEBRASPE – 2021 – SEED-PR) Assinale a opção que apresenta a representação booleana equivalente mais simplificada para a função F(X, Y, Z) = X·Z + Z·(X' + XY).

a) Z + Y·Z
b) Z + X·Y
c) X·Z
d) X + Y·Z
e) Z

13.(CESPE / CEBRASPE – 2021 – SEED-PR) (Adaptada) O fato de o complemento do produto ser igual à soma dos complementos, ou seja, (A·B) ' = A' + B', é justificado:

a) pela lei comutativa.
b) pela lei associativa.
c) pela propriedade distributiva.
d) pelo teorema de Morgan.
e) pelo teorema da identidade.

14.(CESPE / CEBRASPE – 2021 – PG-DF) O resultado do pseudocódigo precedente será 120.

```
função avaliar( a, b )
início
       ma <-a;
       se (ma < b) então ma <- b;
       me <-a;
       se (me > b) então me <- b;
    resultado <- ( ma % me );
    se (resultado = 0)
       então retorne me
       senão avaliar(me, ma)
fim

escreva avaliar (120,30);
```

15.(CESPE / CEBRASPE – 2021 – PG-DF) O resultado do pseudocódigo apresentado será 6.

```
início
   v <- vetor (2, 4, 6, 8 , 10 ,12 );
    escreva ( v[0] + " " +v[1] );
fim;
```

16. (CESPE / CEBRASPE – 2021 – SEED-PR) (4 > 2) xor (5 = 3) and (4 > 2) or (5 = 5)
Assinale a opção que apresenta o resultado da expressão anterior.

a) 2
b) 3
c) 4
d) Falso
e) Verdadeiro

17.(CESPE/CEBRASPE – 2021 – SEED-PR) Assinale a opção que apresenta o resultado do algoritmo apresentado.

```
programa {
    funcao inicio() {
        inteiro vetor[] = { 81, 37, 51, 77, 19 }
        inteiro naosei
        logico achou = verdadeiro
        enquanto (achou)
        {
            achou=falso
            para (inteiro i = 0; i <4; i++)
            {
                se (vetor[i] > vetor[i+1])
                {
                    naosei = vetor[i]
                    vetor[i] = vetor[i+1]
                    vetor[i+1] = naosei
                    achou = verdadeiro
                }
            }
        }
        para (inteiro i = 0; i < 5; i++)
        {
            escreva (vetor[i]+ "\n")
        }
    }
}
```

a) 81
37
51
77
19

b) 81
51
37
17
19

c) 19
37
51
77
81

d) 51
81
37
77
19

e) 19
77
37
81
51

18. (CESPE/CEBRASPE – 2020 – TJPA] Assinale a opção que apresenta o comando que tem a função de implementar desvios incondicionais no programa, mas que é de uso proibido na programação estruturada.

a) IF-THEN-ELSE
b) CASE
c) GOTO
d) WHILE
e) REPEAT

19.(CESPE/CEBRASPE – 2019 – TJ-AM] Os operadores lógicos 'e' e 'ou' possuem, respectivamente, as funções de conjunção e disjunção.

20.(CESPE / CEBRASPE – 2019 – TJ-AM]

```
programa TROCA_VALORES
var
  A, B : inteiro
procedimento TROCA
var
     Y : inteiro
inicio
     Y <- A
     A <- B
     B <- Y
fim
inicio
  leia A, B
  TROCA
  escreva A, B
fim
```

As variáveis A e B estão definidas no programa TROCA_VALORES com escopo global, e a variável Y está definida com escopo local na área de dados da memória; dessa forma, as variáveis A e B somente são visíveis quando a sub-rotina TROCA é executada.

21. (CESPE / CEBRASPE – 2019 – TJ-AM]

```
programa TROCA_VALORES
var
  A, B : inteiro
procedimento TROCA
var
     Y : inteiro
inicio
     Y <- A
     A <- B
     B <- Y
fim
inicio
  leia A, B
  TROCA
  escreva A, B
fim
```

Se as variáveis A e B tivessem sido definidas novamente dentro da sub-rotina TROCA, elas seriam novas variáveis e teriam escopo global para a sub-rotina TROCA

22.(CESPE/CEBRASPE – 2018 – ABIN) Julgue o item subsequente, relativo à lógica de programação.

A expressão a seguir especifica que: 1 será adicionado a x, se x for maior que 0; 1 será subtraído de x, se x for menor que 0; o valor de x será mantido, se x for igual a zero.
Se (x > 0) então x++; senão if (x < 0) x-- ;

23. (CESPE/CEBRASPE – 2018 – ABIN) Na lógica de programação, um bloco de comando é definido como um conjunto de ações para determinada função e tem como delimitadores as palavras reservadas INPUT e OUTPUT.

24.(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

25.(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

26.(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.

27. (CESPE/CEBRASPE – 2016 – TCE/PA) A passagem de parâmetro em uma rotina pode ocorrer de duas maneiras: por valor ou por referência. Em se tratando da passagem por valor, alteram-se os valores dos parâmetros que foram passados para a função.

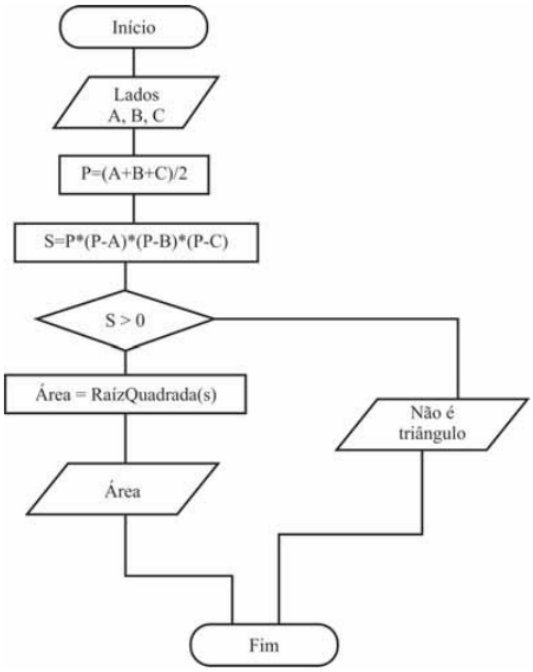
28. (CESPE/CEBRASPE – 2016 – TCE/PA) Em se tratando de linguagens procedimentais, os dados são globais e, portanto, acessíveis a todos os procedimentos.

29. (CESPE/CEBRASPE – 2015 – TER-GO) Comumente usados em fluxogramas representativos de sistemas, os símbolos abaixo correspondem, respectivamente, a dados armazenados, processo, documento e entrada manual.

30.(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.

31.(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.

32.(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.



33. (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.

34. (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.

35. (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.

36. (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.

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

GABARITO – CESPE

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

LISTA DE QUESTÕES – FCC

1. (FCC – 2022 – TST) Considere o trecho de um algoritmo em pseudocódigo que mostra comandos condicionais (se) aninhados com início e fim delimitados por { }:

```
se  (B1)
então     {     Comando1
                Comando2
          }
senão     {  se (B2)
             então    {   Comando3
                          }
             senão
                      {   Comando4
                          }
          }
Comando5;
```

Analisando este trecho, é correto afirmar que

a) se B1 for falso, o Comando3 e o Comando4 serão executados.
b) se B2 for verdadeiro, somente o Comando3 será executado.
c) o Comando5 poderá ser o único comando a ser executado.
d) o Comando4 sempre será executado, uma vez que o comando B2 é sempre falso.
e) o Comando5 sempre será executado.

2. (FCC – 2022 – DPE-RS) Considere o seguinte algoritmo em pseudocódigo:

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

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

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

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

a) I deve ser preenchida com numero>=1900 e numero<=2017
b) II deve ser preenchida com leia(vet[indice])
c) I deve ser preenchida com numero2017
d) II deve ser preenchida com imprima ("Valor valido = ", vetor[indice])
e) I deve ser preenchida com numero>=1900 ou numero<=2017

3. (FCC – 2019 - TRF-4) Considere o programa em pseudocódigo abaixo, que não apresenta erros.

```
        var  var1=1, var2=2: inteiro

        funcao1()
        inicio
            var var1=100, var2=100: inteiro
            imprima("Variaveis dentro da funcao1(): var1= ", var1, " var2=", var2)
        fim

        funcao2()
        inicio
           var1 = var1 +1;
           var2 = var2 +2;
           imprima("Variaveis dentro da funcao2(): var1= ", var1, " var2=", var2)
        fim

inicio
   imprima("Variaveis antes de chamar a funcao1(): var1= ", var1, " var2=", var2)
   funcao1()
   imprima("Variaveis depois de chamar a funcao1():var1= ", var1, " var2=", var2)
   funcao2()
   imprima("Variaveis depois de chamar a funcao2():var1= ", var1, " var2=", var2)
fim
```

O pseudocódigo, ao ser executado, imprimirá

a) Variaveis antes de chamar a funcao1(): var1=0 var2=0
b) Variaveis dentro da funcao1(): var1=1 var2=2
c) Variaveis dentro da funcao2(): var1=101 var2=102
d) Variaveis depois de chamar a funcao1(): var1=1 var2=2
e) Variaveis depois de chamar a funcao2(): var1=101 var2=102

4. (FCC – 2019 – AFAP) No âmbito dos sistemas de numeração computacionais, o número decimal 132 tem sua respectiva correspondência aos seguintes em binário e hexadecimal:

a) 1110 0111 e 84
b) 0100 0010 e C3.
c) 1000 0100 e 84.
d) 1110 0100 e E4.
e) 1000 0100 e 78

5. (FCC – 2019 – AFAP) A soma do hexadecimal 1C5 com o binário de mais baixa ordem 1101, terá como resultado o decimal

a) 434.
b) 466.
c) 737.
d) 479.
e) 482

6. (FCC – 2018 – SABESP) Antes de se escrever um programa em uma linguagem de programação, uma prática recomendada é apresentar a lógica de programação usando uma pseudolinguagem. Considere o algoritmo em pseudolinguagem apresentado abaixo, em que o operador mod retorna o resto da divisão inteira

```
Programa Sabesp
Var ano, n: inteiro
Início
        imprima("Digite o ano com 4 digitos: ")
        leia (ano)

        se (ano > 1949 e ano<=1999)
            então
                 n ← ano mod 100

                 I
                .....

            senão
                imprima("Ano inválido")
        fim se
Fim
```

Um Estagiário, ao analisar o algoritmo acima, conclui corretamente que

a) o operador lógico e está errado e deve ser substituído pelo operador lógico ou na instrução se.
b) a lacuna I deve ser preenchida com a instrução imprima ("O ano é do século XX pois inicia-se com ", n).
c) a lacuna I deve ser preenchida com a instrução imprima ("Os dois últimos dígitos do ano = ", n).
d) se for fornecido 1950 para ano será impresso Ano inválido. (
e) o usuário pode digitar apenas valores de ano com 4 dígitos, positivos e menores ou iguais ao ano atual (2018).

7. (FCC – 2018 – SABESP) Considere, por hipótese, que a SABESP utiliza diferentes preços de tarifas para os serviços de abastecimento de água e/ou coleta de esgoto para o município de São Paulo. Para a categoria Residencial/Favela as tarifas são:

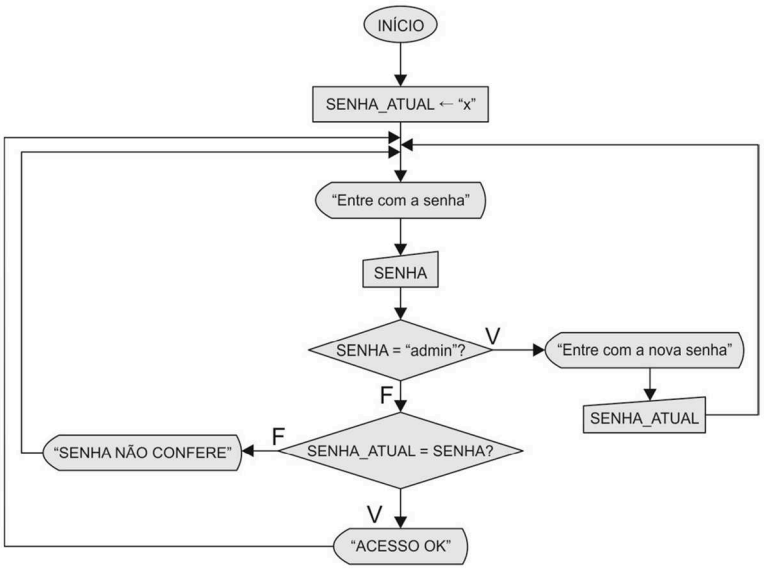
Consumo	Valor da Tarifa
0 a 10	6,25/mês
11 a 20	0,71/m3
21 a 30	2,36/m3
31 a 50	7,14/m3
acima de 50	7,89/m3

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

a. o tipo básico inteiro para ambas as variáveis.
b. o tipo básico caracter para consumo e o tipo básico inteiro para valor.
c. a instrução leia para ler o valor que o usuário deverá pagar.
d. as instruções se então senão de forma aninhada para avaliar as diferentes faixas de consumo.
e. a instrução escolha caso para avaliar os diferentes tipos de valor que poderão ser pagos pelo usuário.

8. (FCC – 2017 – ARTESP) Considere o fluxograma abaixo.

De acordo com a lógica expressa pelo fluxograma, conclui-se que:



a) a solicitação da senha é encerrada quando o usuário fornece a senha admin.
b) somente o usuário com a senha admin consegue alterar a variável SENHA.
c) o usuário com a senha admin avaliada como verdadeira nunca chega ao comando que exibe ACESSO OK.
d) quando a SENHA_ATUAL não confere, esta é inicializada com "x".
e) após acessar o comando que exibe ACESSO OK a estrutura de repetição finaliza.

10.(FCC – 2017 – ARTESP) Considere o algoritmo em pseudocódigo abaixo.

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

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

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

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

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

Este algoritmo

a) não poderia usar a categoria 0 no comando escolha, nem atribuir zero ao valor do pedágio.
b) apresenta erro de lógica na condição do comando condicional se.
c) teria que usar uma condição no comando enquanto (verdadeiro) faça, pois este não pode avaliar apenas o valor lógico verdadeiro.
d) tem erro de sintaxe, pois o comando escolha deveria estar dentro da cláusula senão do comando condicional se.
e) tem erro de sintaxe, pois o comando escolha deveria ter a cláusula senão, que é obrigatória.

11. (FCC – 2017 – TRE-SP) Considere as duas funções, abaixo, escritas em pseudocódigo, que implementam uma lógica.

```
função f1 (N: inteiro): real
Início
        se (N<=1)
              então    retorna  1
              senão    retorna (N  * f1 (N - 1))
        fim se
Fim

função f2 (N: inteiro): real
Var     i: inteiro
        result: real
Início
     result ← 1
     para (i←2 até N passo 1) faça
          result  ← result * i
     fim para
     retorna result
Fim
```

A função

a) f1 e a função f2 recebem uma variável real e retornam um valor inteiro.
b) f1 é executada apenas uma vez, já que em seu corpo existe apenas um comando condicional.
c) f2 é executada N-2 vezes.
d) recursiva faz cálculos e apresenta resultados totalmente diferentes da função iterativa.
e) iterativa e a função recursiva retornam 1 para valores de N=0 e N=1.

12. (FCC – 2017 – TRE-SP) Considere a lógica do algoritmo, abaixo, expressa em pseudocódigo.

```
Var
   tipo V= vetor [0..4] inteiro
   var  j, voto: inteiro
   votos: V

Início
   para (j ← 0 até 4 passo 1) faça
             votos[j] ← 0
   fim para

   enquanto  (verdadeiro)
       imprima ("Digite o voto (1,2,3 ou 0 (branco) -1 finaliza): ")
       leia (voto)
       se (voto = -1)
          então   vá para  RESULT
       fim se
       se (voto < 0 OU voto > 3)
           então   votos[4] ← votos[4] + 1
           senão   votos[voto] ← votos[voto] + 1
       fim se
   fim enquanto
   RESULT:
    para (j ← 1 até 3 passo 1) faça
              imprima ("O candidato ", j," obteve ", votos[j], " votos")
    fim para

    imprima ("Número de votos em branco= ", ..I..)
    imprima ("..II..", votos[4])
Fim
```

De acordo com a lógica apresentada,

a) a instrução se (voto < 0 OU voto > 3) deveria utilizar o operador lógico E ao invés do OU.
b) a lacuna I deve ser preenchida com votos[1]
c) a lacuna II deve ser preenchida com Número de votos nulos =
d) para saber o número total de eleitores basta percorrer o vetor e somar todas as posições de 1 a 3.
e) logo após RESULT:, a instrução para deveria se iniciar em 0 e ir até 4.

13.(FCC – 2017 – TRT-24) 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

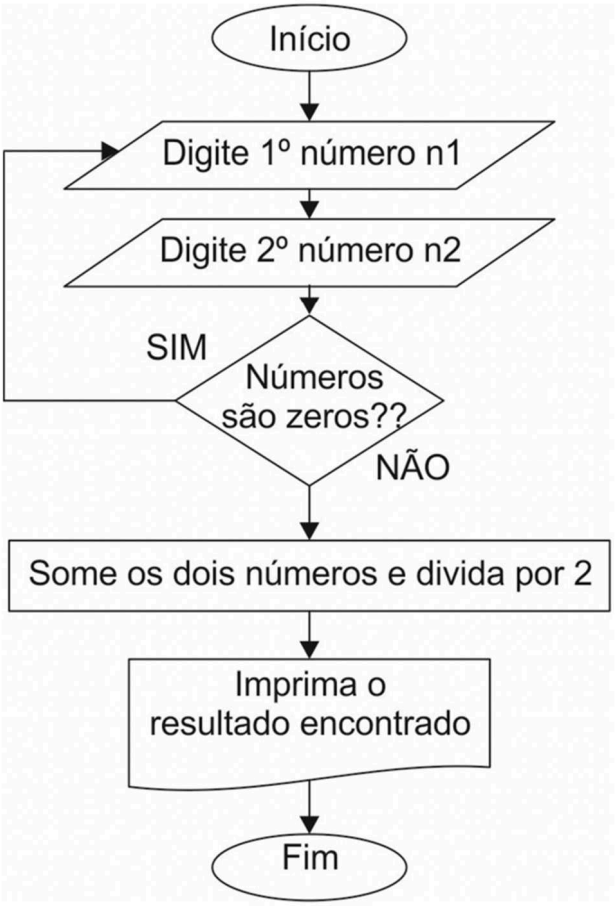
14. (FCC – 2017 – ARTESP) Considere o algoritmo em pseudocódigo abaixo:

```
função  digitos (n: inteiro): inteiro
início
      se (n < 10)
         então          retorna 1
         senão          retorna (1+ digitos (n/10))
      fim se
fim

início

   imprima(" Para 654321 o resultado da função digitos = ", digitos (654321))

fim.
```

Considerando que o operador / realiza a divisão inteira, ao executar o algoritmo acima será impresso: Para 654321 o resultado da função digitos =

a) 21
b) 123456
c) 654321
d) 100000
e) 6

15.(FCC – 2016 – CREMESP) Considere o diagrama abaixo:



Analisando o raciocínio lógico e as estruturas lógicas utilizadas no diagrama, é correto afirmar que:

a) o losango com a inscrição "Números são zeros??" indica que há uma estrutura condicional do tipo escolha-caso.
b) há um comando de repetição do tipo enquanto (condição) faça sendo "Números são zeros??" a condição.
c) a lógica implementa a solução de cálculo da média de 2 números diferentes de zero.
d) se um dos números digitados for zero, o comando de repetição para e nada é impresso.
e) se os dois números digitados na primeira vez forem zero, os dois serão somados e divididos por 2.

16.(FCC – 2015 – DPE-SP) Considere o algoritmo em pseudocódigo no qual DIV calcula o quociente da divisão inteira e MOD o resto da divisão inteira:

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

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

Fim.
```

O algoritmo em pseudocódigo acima

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

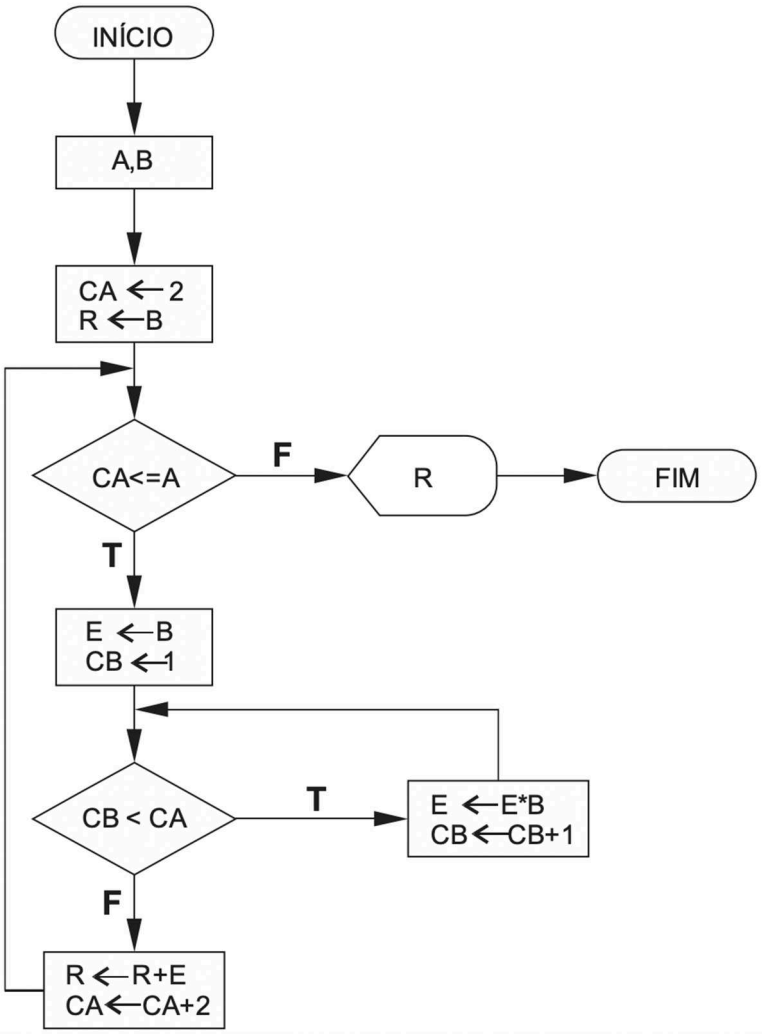
17. (FCC – 2015 – DPE-SP) Considere o algoritmo a seguir, na forma de pseudocódigo:

```
Var n, i, j, k, x: inteiro
Var v: vetor[0..7] inteiro
Início
  v[0] ← 12
  v[1] ← 145
  v[2] ← 1
  v[3] ← 3
  v[4] ← 67
  v[5] ← 9
  v[6] ← 45
  n ← 8
  k ← 3
  x ← 0
  Para j ← n-1 até k passo -1 faça
    v[j] ← v[j - 1];
  Fim_para
  v[k] ← x;
Fim
```

Este pseudocódigo

a) exclui o valor contido na posição x do vetor v.
b) insere o valor de x entre v[k-1]e v[k] no vetor v.
c) exclui o valor contido na posição k do vetor v.
d) tentará, em algum momento, acessar uma posição que não existe no vetor.
e) insere o valor de k entre v[x] e v[x+1] no vetor v.

18.(FCC – 2015 - MANAUSPREV) Considere o fluxograma abaixo:



Se forem lidos para as variáveis A e B, respectivamente, os valores 4 e 4 será exibido o valor

a) 47994.
b) 276.
c) 1338.
d) 4372.

19.(FCC – 2014 – TRT-16) Para responder às questões de números 43 e 44, considere o algoritmo em pseudo-código abaixo.

```
tipo V= vetor [1..10] inteiro
var   i, k: inteiro
      a: logico
      vet: V

inicio

      leia (k)
      i ← 1
      a ← falso
      enquanto (i<=10 e a=falso) faça
      inicio
            se (vet[i] = k)
              entao
                    imprima("Sucesso")
                    a ← verdadeiro

              senão
                    imprima("Insucesso")
            fim se
            i ← i+1

      fim enquanto

fim
```

Considere que o vetor vet possua os seguintes valores: 6, 5, 1, 9, 0, 1, 4, 2, 3 e 7. É correto afirmar:

a) O algoritmo não funciona quando há elementos repetidos no vetor, portanto, o vetor não pode ter elementos repetidos.
b) Se for fornecido 1 para k, será impresso "Insucesso" duas vezes.
c) Somente se o valor de k não for encontrado no vetor será impresso "Insucesso".
d) Faltou o comando a ← falso depois de imprima("Insucesso") no senão do comando se para esta frase ser impressa uma única vez na pesquisa sequencial malsucedida no vetor.
e) Se for fornecido 1 para k, será impresso "Sucesso" duas vezes.

20. (FCC – 2014 – TRT-16) No algoritmo há:

```
tipo V= vetor [1..10] inteiro
var   i, k: inteiro
      a: logico
      vet: V

inicio

      leia (k)
      i ← 1
      a ← falso
      enquanto (i<=10 e a=falso) faça
      inicio
            se (vet[i] = k)
              entao
                    imprima("Sucesso")
                    a ← verdadeiro

              senão
                    imprima("Insucesso")
            fim se
            i ← i+1

      fim enquanto

fim
```

a) diferentes estruturas de dados, de diferentes tipos básicos, que são encontrados em linguagens de programação procedurais.
b) tipos estruturados de dados que são indivisíveis, como inteiro e lógico.
c) um tipo abstrato de dados, que pode ser visto como um modelo matemático, acompanhado das operações definidas sobre o modelo.
d) um tipo estruturado homogêneo de dados, de um tipo básico que pode ser encontrado em diversas linguagens de programação.
e) a implementação de uma lógica particular a uma classe de linguagens de programação, o que limita sua implementação a linguagens orientadas a objetos.

21.(FCC – 2013 – TRT-9) Analise o algoritmo em pseudocódigo abaixo:

```
início
real: n1, n2;
     imprima("Digite a primeira nota: ");
     leia(n1);
     imprima("Digite a segunda nota: ");
     leia(n2);
     se [                                        ]
     então
         media ←(n1+n2)/2;
         imprima ("A media das notas é ", media);
     senão
        imprima("Alguma nota fornecida é inválida.");
     fim se;
fim.
```

Considerando que uma nota válida deve possuir valores entre 0 e 10 (inclusive), a lacuna que corresponde à condição do comando SE é corretamente preenchida por

a) n1 >= 0 OU n1 <=10 OU n2 >= 0 OU n2 <=10
b) (n1 >= 0 E n1 <=10) OU (n2 >= 0 E n2 <=10)
c) (n1 >= 0 OU n1 <=10) E (n2 >= 0 OU n2 <=10)
d) n1 >= 0 E n1 <=10 E n2 >= 0 E n2 <=10
e) n1 > 0 E n1 < 0 E n2 < 10

21.(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.

22. (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.

23. (FCC - 2012 – BANESE) Dadas as variáveis reais (K e M), inteiras (X e Y) e lógicas (W e Z), produz um resultado correto o comando de atribuição:

a) X ← K = M

b) W ← X > Y
c) Y ← M
d) K + M ← 7.5
e) Z ← X + Y

24. (FCC – 2010 – TJ-PA) Considere a seguinte e somente a seguinte situação: Se um procedimento Px contiver uma referência a um outro procedimento Py que por sua vez contém uma referência direta ou indireta a Px, então

a) Px é subconjunto de Py.
b) Px é categorizado como indiretamente recursivo.
c) Py é subconjunto de Px.
d) Py é categorizado como diretamente recursivo.
e) Py é categorizado como indiretamente recursivo.

25. (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

26. (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.

27. (FCC – TRT – 2011) Em relação à programação de computadores, considere:

I. Métodos de passagem de parâmetros permitem que parâmetros sejam transmitidos entre o programa principal e os subprogramas, sendo que, na passagem de parâmetros por valor, o valor real é passado e uma variável local é criada para armazená-lo; nesse processo sempre será efetuada a cópia dessa variável.

II. Pilha é uma estrutura de dados com acesso restrito aos seus elementos ordenados pelo princípio FIFO; a pilha pode ser usada na avaliação de expressões numéricas, na recursividade e pelos compiladores, na passagem de parâmetros para as funções.
III. Prototipação é uma abordagem que envolve a produção de versões iniciais de um sistema futuro com a qual pode-se realizar verificações e experimentações para se avaliar algumas de suas qualidades antes que o sistema venha realmente a ser construído.
IV. Registro é uma estrutura básica que permite guardar coleções de dados de diferentes tipos, sendo normalmente utilizado quando um objeto tem diferentes atributos.

É correto o que consta APENAS em

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

28.(FCC – 2010 – TRE-AM) Formalização de algoritmo proposto em 1936, universalmente conhecido e aceito. Trata-se de um mecanismo simples, que formaliza a ideia de uma pessoa que realiza cálculos, denominado:

a) Recursividade de Bird
b) Máquina de Redução.
c) Máquina de Turing.
d) Sistema de Post.
e) Máquina com Pilhas.

29.(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.

GABARITO – FCC

1. LETRA E
2. LETRA E
3. LETRA D
4. LETRA C
5. LETRA B
6. LETRA C
7. LETRA D
8. LETRA C
9. LETRA B
10. LETRA E
11. LETRA C
12. LETRA D
13. LETRA E
14. LETRA C
15. LETRA B
16. LETRA B
17. LETRA B
18. LETRA B
19. LETRA D
20. LETRA C
21. LETRA C
22. LETRA B
23. LETRA B
24. LETRA B
25. LETRA A
26. LETRA D
27. LETRA C
28. LETRA E

LISTA DE QUESTÕES –

1. (FGV – 2018 – Câmara de Salvador-BA) Expressões lógicas são frequentemente utilizadas em linguagens de programação. Por exemplo, um comando if com a expressão

   if not (A and B)

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

   a) A or B
   b) not A or not B
   c) not A or B
   d) not (not A or not B)
   e) A and B

2. (FGV – 2018 – Câmara de Salvador-BA) Observe o trecho de pseudocódigo exibido a seguir.

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

Numa hipotética execução desse código, os valores exibidos seriam:

a) 2, 5, 7;
b) 6, 13, 15;
c) 6, 13, 19;
d) 7, 15, 21;
e) 7, 17, 23

3. (FGV – 2018 – SEFIN-RO) Analise o trecho de pseudocódigo a seguir.

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

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

A)
```
2  20
5  20
8  20
```

B)
```
2  20
```

C)
```
2
5
8
```

D)
```
2
5
8
```

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

4. (FGV – 2018 – Prefeitura de Niterói – RJ) Sabendo-se que a função retorna o número de elementos de um array e que L assume o tipo de um array de inteiros, indexados a partir de

```
L := {10,2,40,53,28,12};
trocou := True;
while trocou {
    trocou := False;
    for k := 0 to len(L) - 2 {
        if L[k] > L[k+1] {
            L[k] = L[k+1];
            L[k+1] = L[k];
            trocou = True;
        }
    }
}
print L
```

Esse algoritmo deveria ordenar os elementos do array em ordem crescente, mas há problemas no código que produzem resultados errôneos. Assinale a opção que indica o que é de fato printado ao final da execução do código mostrado.

a) {10,2,40,53,28,12}
b) {2,10,12,28,40,53}
c) {53,40,28,12,10,2}
d) {2,2,12,12,12,12}
e) {2,10,10,10,10,12}

5. (FGV – 2016 – SEE-PE) Analise o trecho de pseudocódigo exibido a seguir.

```
x = 1;
while x < 10
{
    y = x+2;
    while y < 10
    {
        print x, y;
        y = y + 1;
    };
    x = x+1;
};
```

De acordo com o pseudocódigo acima, assinale a opção que indica o número de vezes que o comando print é executado.

a) 28
b) 56
c) 60
d) 84
e) 100

6. (FGV – 2016 – Prefeitura de Paulínia – SP) Analise o pseudocódigo de uma função recursiva exibido a seguir.

```
function f(x as integer) as integer:
    if x = 1 then
        return 1
    else
        return x + f(x-1)
```

Assinale a opção que indica o valor retornado para f (9).

a) 1
b) 29
c) 36
d) 45
e) 55

7. (FGV – 2016 – SEE-PE) Analise a função ff definida pelo trecho de pseudocódigo exibido a seguir.

```
function ff (N as integer) as integer
{
    if N = 5 then
    {   return N;
    }
    else
    {   return N + ff(N-1);
    };
}
```

Assinale a opção que indica o valor correto da expressão ff(12).

a) 12
b) 17
c) 68
d) 72
e) 78

8. (FGV – 2017 – SEPOG-RO) Considere o algoritmo em pseudocódigo descrito a seguir.

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

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

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

9. (FGV – 2016 – SEE-PE) Analise o trecho de pseudocódigo a seguir.

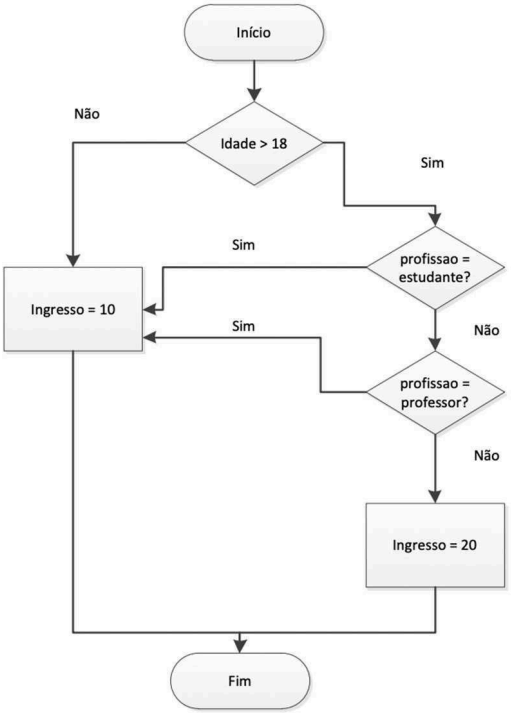
```
function f(a as integer, b as integer, c as
integer) as integer
{
    while b <= c
    {
        a = b+c;
        y = b+1;
    };
    return a+b+c;
};
x=10;
y=2;
z=3;
print f(x, y, z), x, y, z;
```

Em algumas linguagens de programação é possível fazer a passagem de parâmetros como *byref* ou *byvalue*, e assim podemos supor que a função f acima poderia ser reescrita especificando, para cada parâmetro, uma das duas formas citadas.

Supondo-se que o primeiro e o segundo parâmetro da função tenham sido passados como *byref* e o terceiro, como *byvalue*, os quatro valores exibidos pelo comando print seriam, respectivamente,

a) 13, 6, 2 e 3.
b) 13, 6, 4 e 3.
c) 10, 10, 2 e 3.
d) 13, 0, 0 e 3.
e) 10, 8, 5 e 2.

10.(FGV – 2014 – SUSAM) A figura a seguir apresenta um fluxograma de uma função que fornece ao usuário o valor do ingresso que deverá ser cobrado para a entrada no cinema



Os parâmetros de entrada da função são sua IDADE e PROFISSÃO. A conversão deste fluxograma em pseudolinguagem de programação é:

(A)
```
SE (IDADE < 18) OU (PROFISSÃO = ESTUDANTE) OU
(PROFISSÃO = PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
FIM SE
```

(B)
```
SE (IDADE < 18) E (PROFISSÃO = ESTUDANTE) E
(PROFISSÃO = PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
FIM SE
```

(C)
```
SE (IDADE < 18) E [(PROFISSÃO = ESTUDANTE) OU
(PROFISSÃO = PROFESSOR)] ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
FIM SE
```

(D)
```
SE [(IDADE < 18) OU (PROFISSÃO = ESTUDANTE)] E
(PROFISSÃO = PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
FIM SE
```

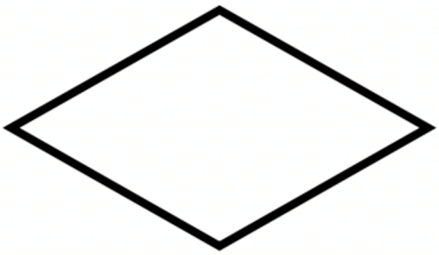
(E)
```
SE (IDADE < 18) OU (PROFISSÃO <> ESTUDANTE) OU
(PROFISSÃO <> PROFESSOR) ENTÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 10 REAIS')
SENÃO
     IMPRIMA ('VALOR DO INGRESSO IGUAL A 20 REAIS')
FIM SE
```

11.(FGV – 2013 – TCE-BA) No mapeamento de um processo, o fluxograma tem papel relevante, pois ele descreve e mapeia as etapas do processo de forma lógica e planejada. Com o auxílio do fluxograma, pode-se eliminar erros, evitar duplicidades, eliminar tarefas desnecessárias, otimizar o ciclo etc. Em um fluxograma, o símbolo que indica uma decisão é

a) 

b)

c)



d)

e)

GABARITO

1. LETRA B
2. LETRA D
3. LETRA C
4. LETRA D
5. LETRA A
6. LETRA D
7. LETRA C
8. LETRA B
9. LETRA B
10. LETRA A
11. LETRA C

LISTA DE QUESTÕES – MULTIBANCAS

1. **(Quadrix – 2022 – CRC-PR) A respeito dos diagramas de casos de uso, dos diagramas de classe, da análise essencial e da lógica de programação, julgue o item a seguir**

Na lógica de programação, um algoritmo é conceituado como uma sequência estruturada e organizada de passos que tem por objetivo atingir um objetivo, seja ele definido ou indefinido.

2. **(SELECON – 2022 – Prefeitura de Pontes e Lacerda) O algoritmo abaixo utiliza os conceitos de passagem de parâmetros, sendo de SD para X por referência; de NR para Y e de VL para W por valor.**

```
algoritmo "PREF_PL"
var
   SD : caracter
   NR : inteiro
   VL : logico
procedimento PROC_PL(var X:caracter;Y:inteiro;W:logico)
inicio
      X <- "BURITI"
      Y <- 2021
      W <- VERDADEIRO
fimprocedimento
inicio
      SD <- "GUAPORÉ"
      NR <- 2023
      VL <- FALSO
      PROC_PL(SD,NR,VL)
      se VL = VERDADEIRO entao
         NR <- 2022
      fimse
      escreval("SD = ",SD:10," NR = ",NR:4," VL = ",VL)
fimalgoritmo
```

Nessas condições, ao final da execução, os valores para SD, NR e VL serão, respectivamente:

a) GUAPORÉ, 2023 e VERDADEIRO
b) BURITI, 2022 e VERDADEIRO
c) GUAPORÉ, 2022 e FALSO
d) BURITI, 2023 e FALSO

3. **(SELECON – 2022 – Prefeitura de Pontes e Lacerda-MT) No que se refere à lógica de programação, observe o pseudocódigo abaixo, referente a um algoritmo que contém uma função recursiva.**

```
algoritmo "PL_2022"
var
    CT, RP : inteiro
funcao FREC(AUX:inteiro):inteiro
inicio
   se  (AUX = 0) OU (AUX = 1)  entao
       retorne AUX
   senao
       retorne FREC(AUX-2) + FREC(AUX-1)
   fimse
fimfuncao
inicio
para CT de 0 ate 5 faca
     RP <- FREC(CT)
     escreva(RP:3)
fimpara
fimalgoritmo
```

Após a execução, a sequência de números de saída é:

a) 0 1 1 2 3
b) 1 1 2 3 5
c) 0 1 1 2 3 5
d) 1 1 2 3 5 8

4. (UFV – 2022 – UFV-MG) Considere o algoritmo em pseudocódigo a seguir

```
← é o comando de atribuição
% é o resto de divisão inteira

início
        inteiro x
        inteiro y
        inteiro z
        x ← 100
        z ← 3
        x ← x + 40
        y ← 5 ^ z * 4
        enquanto (z <= 7) faça
                x ← x – 15
                se x % 2 = 0 então
                        z ← z + 1
                se não
                        z ← z + 2
                fim se
                y ← y + z - x
        fim enquanto
        escreva x, y, z
fim
```

Assinale a alternativa que apresenta CORRETAMENTE os valores impressos pelo algoritmo:

a) 95, 276, 8.
b) 95, 189, 8.
c) 80, 189, 7.
d) 110, 276, 6.

5. (UFV – 2022 – UFV-MG) Considere o algoritmo em pseudocódigo a seguir

```
← é o comando de atribuição
o índice da primeira posição do vetor é 1

início
        inteiro vetor[ ] = {91, 47, 61, 87, 29}
        inteiro x
        inteiro i
        i ← 1
        enquanto (i < 5) faça
            se (vetor[i] > vetor[i+1]) então
                    x ← vetor[i]
                    vetor[i] ← vetor[i +1]
                    vetor[i+1] ← x
             fim se
            i ← i + 1
        fim enquanto
        i ← 1
        enquanto (i < 6) faça
                escreva vetor[i]
              i ← i + 1
        fim enquanto
fim
```

Assinale a alternativa que apresenta CORRETAMENTE a sequência de valores impressos pelo algoritmo:

a) 91, 47, 61, 87, 29.
b) 91, 61, 47, 87, 29.
c) 51, 47, 87, 29, 91.
d) 47, 61, 87, 29, 91.

6. (UFSC – 2022 – UFSC) Considere o pseudocódigo do método de ordenação Insertion Sort, o qual ordena em ordem crescente os números naturais armazenados em um vetor (array) v de tamanho t indexado a partir de zero (ou seja, índices do vetor variam de 0 a t-1).

Assinale a alternativa que completa corretamente o espaço pontilhado entre chaves do pseudocódigo abaixo.

```
função Ordena(v, t)
{
  i ← 1
  enquanto i < t faça
  {
    j ← i
    enquanto j > 0 e v[j-1] > v[j] faça
    {
      ......................
    }
    i ← i + 1
  }
}
```

a) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x
j ← j + 1

b) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x
j ← j - 1

c) x ← v[j]
v[j] ← v[j + 1]
v[j + 1] ← x
j ← j - 1

d) x ← v[j]
v[j] ← v[j + 1]
v[j + 1] ← x
j ← j + 1

e) x ← v[j]
v[j] ← v[j - 1]
v[j – 1] ← x

j ← j – 2

7. (IDECAN – 2021 – PEFOCE) Na construção de algoritmos e programas de computador, sendo x e y duas condições de teste, os operadores lógicos AND e OR são bastante utilizados nas estruturas de controle dos tipos seleção e repetição e correspondem às tabelasverdade mostradas, respectivamente, em

A)

x \ y	falso	verdadeiro
falso	V	V
verdadeiro	V	F

e

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	F

B)

x \ y	falso	verdadeiro
falso	F	F
verdadeiro	F	V

e

x \ y	falso	verdadeiro
falso	F	V
verdadeiro	V	V

C)

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	V

e

x \ y	falso	verdadeiro
falso	F	F
verdadeiro	F	V

D)

x \ y	falso	verdadeiro
falso	F	V
verdadeiro	V	V

e

x \ y	falso	verdadeiro
falso	V	V
verdadeiro	V	F

E)

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	F

e

x \ y	falso	verdadeiro
falso	V	F
verdadeiro	F	V

8. (CESPE/CEBRASPE – 2021 – SEED-PR) Assinale a opção que apresenta o resultado do algoritmo apresentado.

```
programa {
    funcao inicio() {
        inteiro vetor[] = { 81, 37, 51, 77, 19 }
        inteiro naosei
        logico achou = verdadeiro
        enquanto (achou)
        {
            achou=falso
            para (inteiro i = 0; i <4; i++)
            {
                se (vetor[i] > vetor[i+1])
                {
                    naosei = vetor[i]
                    vetor[i] = vetor[i+1]
                    vetor[i+1] = naosei
                    achou = verdadeiro
                }
            }
        }
        para (inteiro i = 0; i < 5; i++)
        {
            escreva (vetor[i]+ "\n")
        }
    }
}
```

a) 81
37
51
77
19

b) 81
51
37
17
19

c) 19
37
51
77
81

d) 51
81
37
77
19

e) 19
77
37
81
51

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

10. **(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

11. **(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.

12. (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).

13. **(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.

14. **(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

15. **(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

16. **(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

17. **(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.

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

18. (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

19. (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);

20. (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.

21. **(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

...

22. (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.

23. **(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.

24. **(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.

GABARITO – MULTIBANCAS

1. ERRADO
2. LETRA D
3. LETRA D
4. LETRA B
5. LETRA D
6. LETRA B
7. LETRA B
8. LETRA C
9. LETRA C
10. LETRA B
11. LETRA C
12. LETRA C
13. LETRA D
14. LETRA B
15. LETRA E
16. LETRA D
17. LETRA C
18. LETRA B
19. LETRA A
20. LETRA C
21. LETRA E
22. LETRA D
23. LETRA E
24. LETRA D

COMPILADORES

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?

Código-Fonte (Source Code)

O código-fonte é um conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica. Atualmente, com a diversificação de linguagens, o código pode ser escrito de forma totalmente modular, podendo um mesmo conjunto de códigos ser compartilhado por diversos programas e, até mesmo, linguagens.

Código-Objeto (Object Code)

O código-objeto representa a versão compilada e montada de um arquivo de código-fonte. Ele não é diretamente um arquivo executável, na medida em que ele deve ser ligado com outros códigos-objeto para criar um executável. Trata-se de uma porção do código de máquina[1], mas que contém ainda tabelas de símbolos, constantes, strings e outras referências.

Pré-Processador (Preprocessor)

O pré-processador é um programa que recebe um texto e efetua conversões léxicas, tais como substituição de macros, inclusão condicional, inclusão de ficheiros e exclusão de comentários. É baseado em Diretivas de Compilação (Ex: #define, #include, etc), que são comandos que não são compilados, dirigidos ao pré-processador e executado pelo compilador antes do processo de compilação em si.

Normalmente ele é responsável por mudanças no código fonte destinadas de acordo com decisões tomadas em tempo de compilação. Por exemplo, um programa em C permite instruções condicionais para o pré-processador que podem incluir ou não parte do código caso uma assertiva lógica seja verdadeira ou falsa, ou simplesmente um termo esteja definido ou não.

O uso de pré-processadores tem vindo a ser cada vez menos comum à medida que as linguagens recentes fornecem características mais abstratas em vez de características orientadas lexicalmente. Há também linguagens recentes que tem pouca ou nenhuma funcionalidade, como por exemplo

[1] Alguns chamam o código-objeto de Código (Binário) de Máquina.

a linguagem Java, que não possui um pré-processador (mas quem programou C na faculdade usou muito!).

Compilador (Compiler)

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: **Análise ou Front-End** (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e **Síntese ou Back-end** (Otimização de Código Intermediário e Geração de Código Final).

Análise Léxica: nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados **tokens**, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

Em português, nós dividimos um texto em substantivos, verbos, adjetivos, etc; em linguagens de programação, nós dividimos um texto em palavras-reservadas, identificadores, operadores, pontuação, números, etc. A Análise Léxica é uma forma de verificar um **alfabeto**, ou seja, conseguimos verificar se existe ou não algum caractere que não faz parte do alfabeto.

Encontrar um erro léxico é um pouco difícil! Em geral, só haverá erro se o scanner achar um caractere que não pertence ao alfabeto da linguagem. Nesse caso, ou o projetista esqueceu de incluir o caractere no alfabeto ou realmente o usuário utilizou um caractere indevido. Se o programador escreve "fi" em vez de "if", o scanner não dará erro, visto que "fi" pode ser um identificador válido.

Ex: Eu vou passar nesse concurso! Os tokens obtidos foram: "Eu"; "vou"; "passar"; "nesse"; "concurso"; "!".

Análise Sintática: também conhecida como Análise Gramatical ou Parsing, analisa uma sequência de entrada para determinar sua estrutura gramatical, segundo uma determinada gramática formal. Ela pega os tokens resultantes do processo de análise léxica e joga em uma **estrutura hierárquica**, como uma **árvore.** Assim como no português, verifica-se a estrutura do texto[2].

Assim como na linguagem natural, nas linguagens de programação se espera que os símbolos sejam dispostos de uma forma lógica uns em relação aos outros, tal como as palavras se juntam para formar expressões, orações e frases – isso é sintaxe. Um erro sintático, portanto, é um caso em que as "frases" do programa estão mal formuladas, aquilo que comumente chamamos de **"erro gramatical".**

[2] Uma árvore de análise sintática de uma gramática de atributos é a árvore de análise sintática baseada em sua gramática BNF associada, com um conjunto possivelmente vazio de valores de atributos anexado a cada nó.

Erros sintáticos são bastante comuns, entre eles nós podemos mencionar: parênteses que abrem, mas não fecham; dois números um ao lado do outro sem nenhum operador entre eles; duas instruções sem um ponto-e-vírgula entre elas; uma palavra-chave sendo usada numa posição inesperada. Quem aí nunca esqueceu de fechar parênteses?

O analisador sintático consulta a tabela de símbolos para verificar a presença de variáveis definidas pelo programador. Se o analisador encontra uma variável para a qual não existe descrição na tabela de símbolos, ele emite uma mensagem de erro. O analisador sintático também detecta construções ilegais como A = B + C = D.

No entanto, o que o analisador sintático não faz é verificar se os operadores = ou + são válidos para as variáveis A, B, C e D – quem faz isso é o analisador semântico. Logo, nós podemos afirmar que o analisador sintático tem uma certa preocupação com a operação, mas não com os operandos. E mais: os erros de sintaxe são sempre detectados em tempo de compilação/parse.

Ex: string nome = "Diego'; ora, se eu declarar um valor com aspas-duplas, preciso fechá-lo com aspas duplas e, não, simples.

Análise Semântica: nesta fase, verificam-se erros semânticos, i.e., erros de sentido. Entre as principais atividades, estão a checagem de tipos, verificação de fluxos de controle e verificação de unicidade de declaração de variáveis. Essa fase também é encarregada de analisar a utilização dos identificadores e de ligar cada uma delas a sua declaração.

A semântica refere-se ao significado daquilo que se quer dizer. Da mesma forma que uma frase em linguagem natural pode estar gramaticalmente correta, mas não fazer o menor sentido, também as instruções dadas ao computador podem estar bem formatadas, mas não fazer aquilo que o programador quer - ou mesmo nada de útil ou ainda possível.

Erros semânticos também são bastante comuns, entre eles nós podemos mencionar: dividir um número por uma string; criar uma classe que herda de si mesma; usar o operador ^ achando que é de exponenciação, mas na verdade é um ou exclusivo; ou dividir zero por zero. Erros semânticos podem ser detectados tanto durante a compilação quanto durante a execução.

O Analisador Semântico usa a árvore de análise como entrada e verifica se os tipos de dados são apropriados usando informação da tabela de símbolos. O analisador semântico também faz as promoções adequadas de tipos de dados, tais como mudar um valor ou uma variável do tipo inteiro para ponto-flutuante, se tais promoções são suportadas pelas regras da linguagem.

Ex: int x = "Diego"; ora, eu não posso declarar uma variável do tipo inteiro e atribuir um valor textual.

Geração de Código Intermediário: nesta fase, ocorre a transformação da árvore sintática em uma representação intermediária do código fonte. Esta linguagem intermediária é mais próxima da

linguagem objeto do que o código fonte, mas ainda permite uma manipulação mais fácil do que se o código Assembly ou código de máquina fosse utilizado.

Otimização de Código Intermediário: nesta fase, examina-se o código intermediário produzido durante a fase anterior com objetivo de produzir, através de algumas técnicas, um código que execute com bastante eficiência o programa. Utilizam-se técnicas que detectam padrões dentro do código produzido e os **substitui** por códigos mais eficientes. Simples assim!

Geração de Código Final: chegamos à última fase do processo de compilação com a geração do código de montagem (ainda não é o código-objeto, apesar de muitos compiladores já realizarem a montagem). Um código-objeto não é imediatamente executável, visto que ainda há **código binário** a ser incluído no programa, tais como **bibliotecas**[3].

Montador (Assembler)

Existe uma maneira de efetuar tradução de forma mais **rápida e simples**, por meio de um **Montador**. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Cross-assembler: é executado em um computador com um processador diferente daquele para o qual se está gerando código.

Macro-assembler: dispõe de recursos de macro, efetuando a expansão do código cada vez que uma macro for encontrada.

Micro-assembler: permite a escrita de micro-instruções, definindo-se assim um conjunto de instruções de um processador microprogramável.

Meta-assembler: é um assembler que pode montar programas para vários processadores diferentes.

Assembler de um passo: varre o programa-fonte apenas uma vez, gerando o código (deve existir alguma forma de se revolver as referências adiante).

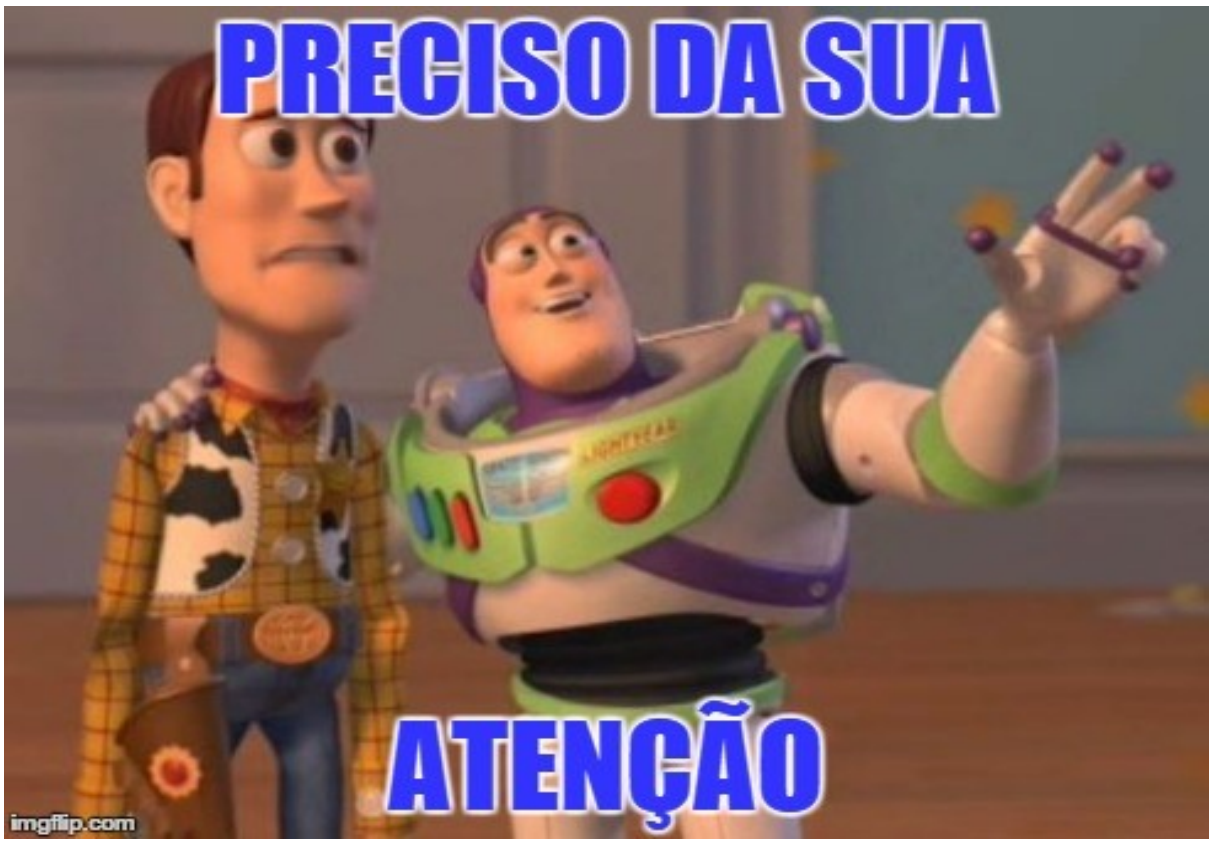
Assembler de dois passos: varre o programa-fonte duas vezes para gerar o código, podendo assim resolver automaticamente as referências adiante.

A Linguagem de Montagem é traduzida para uma Linguagem de Máquina (Binária). Ambas são dependentes do hardware! Como assim, professor? Assembly é uma linguagem de programação,

[3] Essa é a única fase da síntese que é obrigatória; as duas anteriores são opcionais.

todavia cada família de processadores possui sua própria linguagem de montagem particular (Ex: INTEL, x86, MIPS, ARM, SPARC). Por essa razão, não se trata de uma linguagem portável.

A Montagem é um tipo de tradução, assim como a compilação ou a interpretação. Logo, ela pode existir por si só sem que haja compilação alguma. Eu vos-pergunto: sendo rigoroso, um compilador traduz um código-fonte em um código-objeto? Na teoria, ele traduz um código-fonte em um código de montagem, que depois vai à um montador para se transformar em código-objeto.



Em tese, compiladores produzem um código de montagem, que é passado a um montador para processamento posterior. No entanto, na prática, a maioria dos compiladores realizam a tarefa do montador, produzindo um código de máquina relocável, que pode ser passado diretamente para um ligador e/ou carregador, i.e., já entregam um código-objeto!

Ligador (Linker ou Link-Editor)

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Carregador (Loader)

São programas que transferem o código de máquina de um módulo objeto para a memória e encaminham o início de sua execução. Eles recebem o módulo de carga como entrada, transferem seu código para a memória e realizam apenas os ajustes de realocação de acordo com o endereço base de memória. Um programa pode ser composto por partes independentemente carregadas e realocadas.

Existem dois tipos básicos de carregador: carregadores binários e realocáveis. Os carregadores binários (ou carregadores absolutos) são mais simples e apenas copiam o arquivo em formato binário para a memória, de tal forma que o arquivo executável é simplesmente uma imagem binária do programa em execução na memória.

Um programa que usa carregadores absolutos é associado com localizações específicas de memória, e por isso deve sempre ser carregado na mesma área de memória para serem executados corretamente (Ex: programas com extensão .COM). O carregador realocável pode ser colocado em qualquer local da memória para execução.

O programa executável realocável é semelhante ao programa executável absoluto, exceto que os endereços são todos relativos a zero (não são absolutos) e a informação de quais endereços relativos devem ser alterados quando o programa for colocado em execução estão junto com o arquivo executável (Ex: programas com extensão .EXE)[4].

Bibliotecas

Bibliotecas são uma coleção de funções e definições utilizadas no desenvolvimento de software para algum propósito específico. Elas provêm serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular. O Ligador é o responsável, em geral, por fazer referências entre os executáveis e as bibliotecas.

É muito comum, ao precisar resolver determinado problema, utilizar-se de uma biblioteca escrita por outro programador. Isso pode ser vantajoso, pois reduz a necessidade de criar códigos, mas também pode ser perigoso, visto que pode-se optar por uma biblioteca mal implementada (ou até mesmo maliciosa) e acabar tendo grandes dores de cabeça.

Existem basicamente dois tipos de bibliotecas, as bibliotecas estáticas e as bibliotecas dinâmicas (ou compartilhadas). Ao compilar um programa que chama uma biblioteca estática, todo o código da biblioteca é copiado e inserido dentro do binário final. O termo "estática" se deve ao fato da

[4] Atualmente também existem Carregadores que fazem a ligação de partes do programa em tempo de execução. Estes são chamados Carregadores-Ligadores.

linguagem se tornar uma parte estática do binário, não podendo ser compartilhada por outros programas.

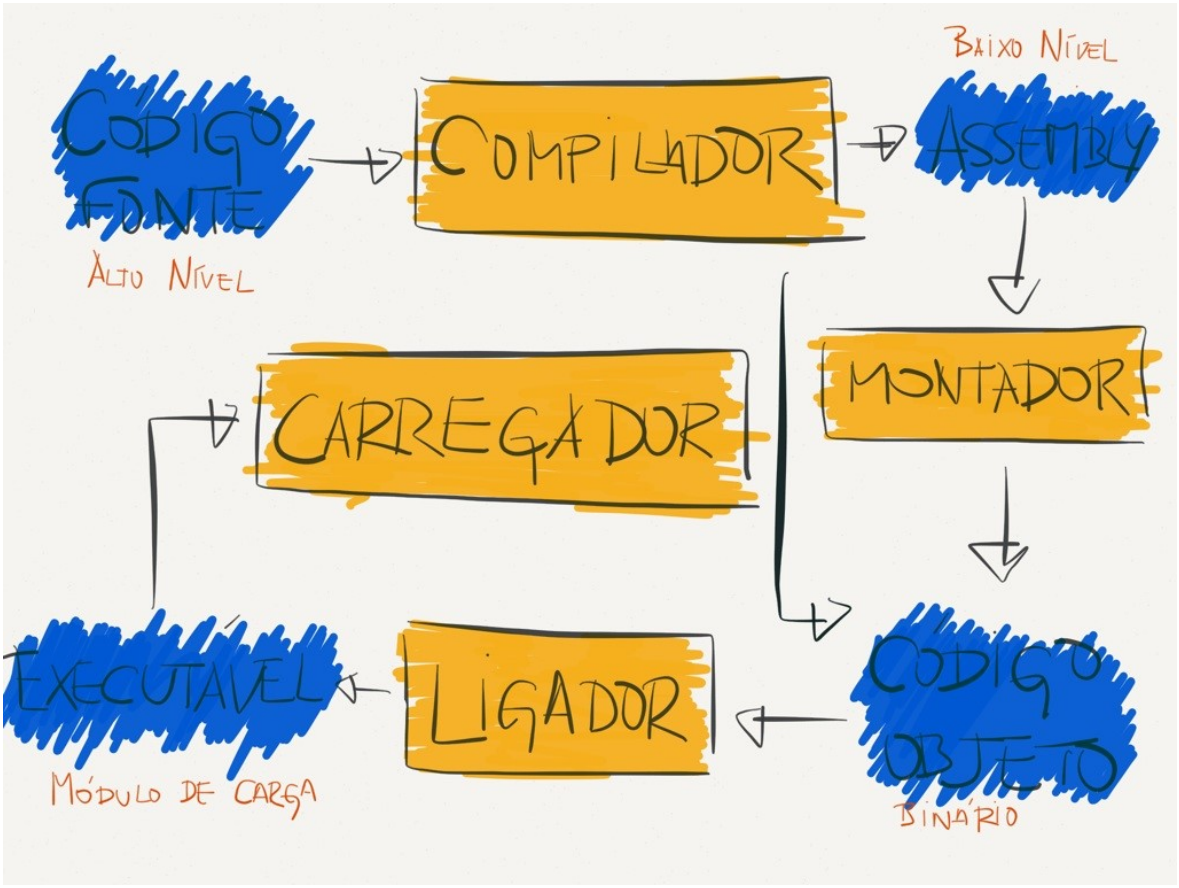
Dessa forma, mesmo quando a biblioteca presente no seu sistema **mude**, seja **removida** ou **corrompida**, o programa **continuará funcionando**. Porém além do seu binário final ocupar mais espaço em disco e memória (já que ele inclui a biblioteca estática), o seu programa não poderá usufruir de qualquer otimização que venha a ocorrer nesta biblioteca, a não ser que ele seja recompilado/reinstalado.

Ao contrário das bibliotecas estáticas, as bibliotecas dinâmicas ou compartilhadas não são inseridas em sua totalidade, elas são apenas **referenciadas no binário final**. O termo "compartilhada" expressa exatamente a característica de um ou mais programas poderem utilizar a mesma biblioteca, ocupando assim menos espaço em disco e na memória.

Além disso, a característica dinâmica da biblioteca compartilhada confere ao seu programa a possibilidade de usufruir de atualizações nas bibliotecas sem a necessidade de uma nova compilação/instalação. Entretanto pode acontecer de o programa não executar corretamente, seja por não encontrar a biblioteca ou (após uma atualização) pela biblioteca não ser mais compatível com seu programa.

O último ponto negativo da biblioteca compartilhada é que, devido a seu tempo de carregamento, é adicionada uma **pequena latência** nas chamadas a esta biblioteca, consequentemente seu programa será ligeiramente mais lento que o mesmo utilizando bibliotecas estáticas. No Windows, temos as .DLL (Dynamic-Link Libraries) e no Unix, temos as .LIB.

Processo Genérico de Compilação



Um programador escreve módulos de **código-fonte** em uma linguagem de programação de **alto nível (ex: C).** Esse código-fonte passa por um pré-processador que manipula diretivas, macros, comentários, etc. Começa, então, de fato o processo de compilação por meio de uma análise léxica (tokenização e tabela de símbolos), seguida de uma análise sintática (parsing e estrutura de árvore).

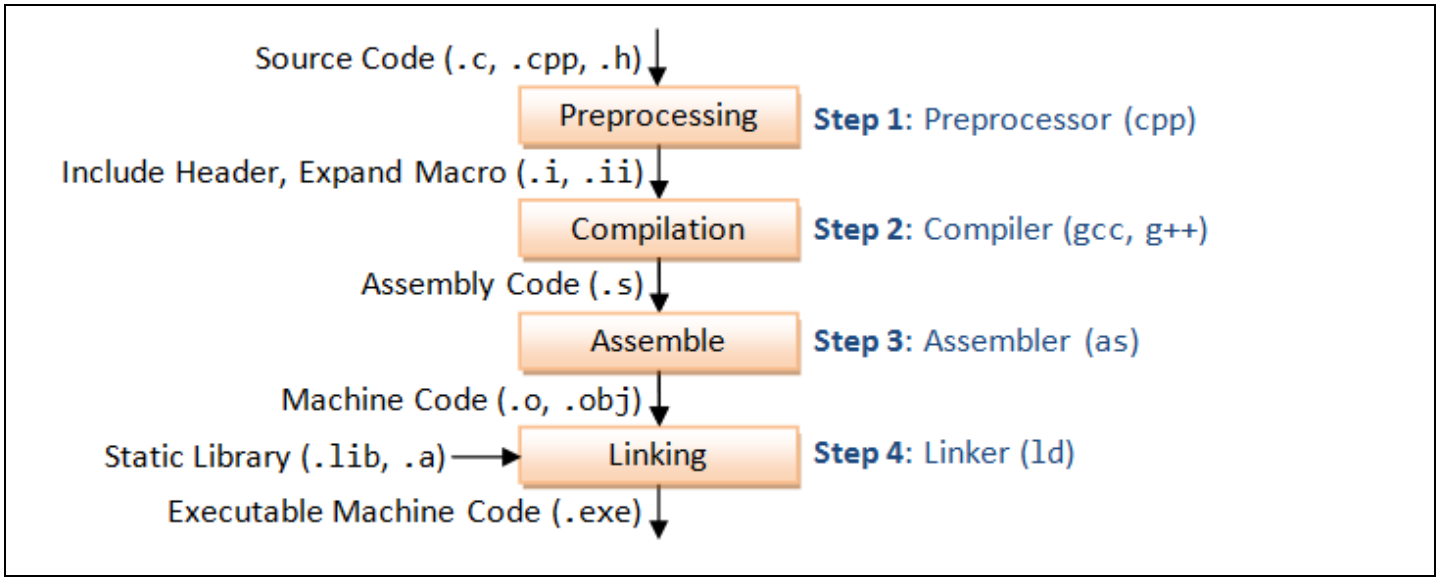
Avança-se à análise semântica (checagem de tipos e verificação de variáveis), depois à geração do código intermediário, sua otimização e, por fim, a geração do código final, que em geral é um programa em linguagem de montagem (Assembly). Porém, agora, nós temos diversos módulos em linguagem de programação de baixo nível, que os computadores ainda não conseguem entender.

Precisamos, então, do Montador[5]! Ele traduzirá cada modulo da linguagem de montagem para um código-objeto com algumas referências a resolver e que ainda não pode ser executado[6]. O

[5] Como já foi dito anteriormente, alguns compiladores fazem também o papel de montador e já entregam um código-objeto.
[6] O código-objeto é uma combinação de instruções de linguagem de máquina, dados e informações necessárias para colocar as instruções corretamente na memória.

Ligador combinará todos esses módulos-objeto em um módulo de carga, com diversas rotinas de bibliotecas para resolver as referências internas e externas.

Por fim, o carregador coloca o código de máquina nos locais apropriados da memória para ser executado pelo processador. Para agilizar o processo de tradução, algumas etapas são puladas ou combinadas. Alguns compiladores produzem módulos-objeto diretamente, e alguns sistemas utilizam Carregadores-Ligadores, que realizam as duas últimas etapas.



INTERPRETADORES

Como linguagens compiladas, linguagens interpretadas também têm um relacionamento um-para-muitos entre comandos do código fonte e instruções executáveis de máquina (i.e., nós escrevemos um comando no código-fonte, mas ele pode se transformar em diversas instruções de máquina). E qual seria a diferença principal entre os compiladores e os interpretadores?

Bem, diferente de compiladores, que leem todo o arquivo de código fonte antes de produzir uma sequência binária, interpretadores processam um comando de cada vez. Quem já trabalhou na prática sabe muito bem disso – a compilação ocorre toda de uma vez; na interpretação, é instrução por instrução (eu recomendo que vocês testem isso, mas só se estiverem com tempo sobrando).

Com tanto trabalho sendo feito "durante o voo", interpretadores geralmente são **muito mais lentos** do que compiladores. Pelo menos cinco dos seis passos requeridos por compiladores também devem ser feitos por interpretadores, e estes passos são realizados em "tempo real". Esta abordagem não dá oportunidade para otimização de código.

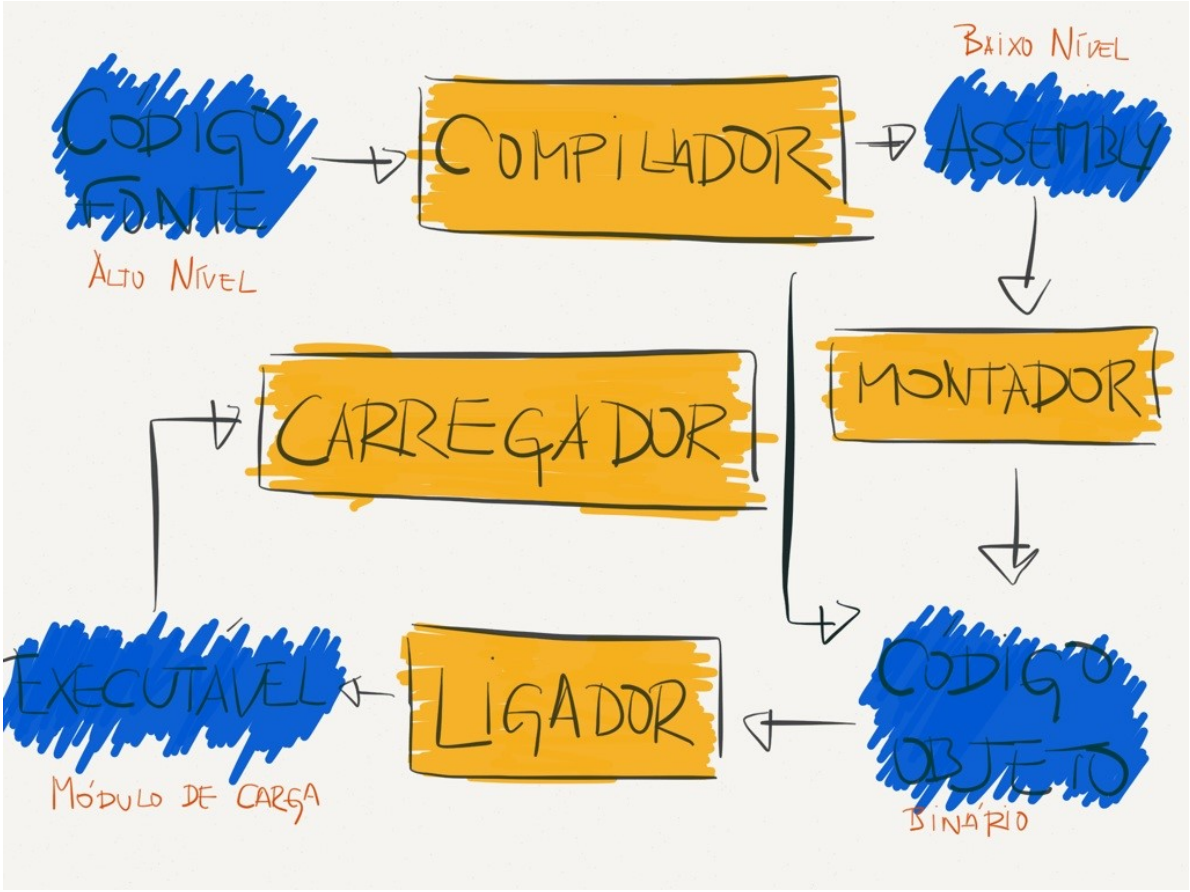
Além disso, a detecção de erros em interpretadores é geralmente limitada à verificação da sintaxe da linguagem e dos tipos de variáveis. Por exemplo, poucos interpretadores detectam possíveis operações aritméticas ilegais antes que ocorram ou avisam o programador antes de exceder os limites de um array. Os compiladores detectam erros bem mais variados.

Apesar da velocidade de execução vagarosa e da verificação tardia de erros, existem boas razões para usar uma **linguagem interpretada.** A mais importante delas é que linguagens interpretadas permitem depuração em nível de código-fonte, tornando-as ideais para programadores iniciantes e usuários finais – podemos dizer que a detecção de erros é mais fácil para quem está começando.

QUESTÕES COMENTADAS – COMPILADORES E INTERPRETADORES - CEBRASPE

1. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Uma diferença fundamental entre um compilador e um montador é que o compilador gera um arquivo executável a partir de um arquivo texto com o programa, enquanto o montador executa diretamente a descrição assembler, sem gerar arquivo na saída.

Comentários:



Conforme vimos em aula, bastava lembrar da imagem acima! Observem que há duas setinhas saindo do Compilador, i.e., ele pode gerar um código de montagem ou um código-objeto e, não, um arquivo executável. Ademais, um montador gera um arquivo de saída: o código-objeto!
Gabarito: E

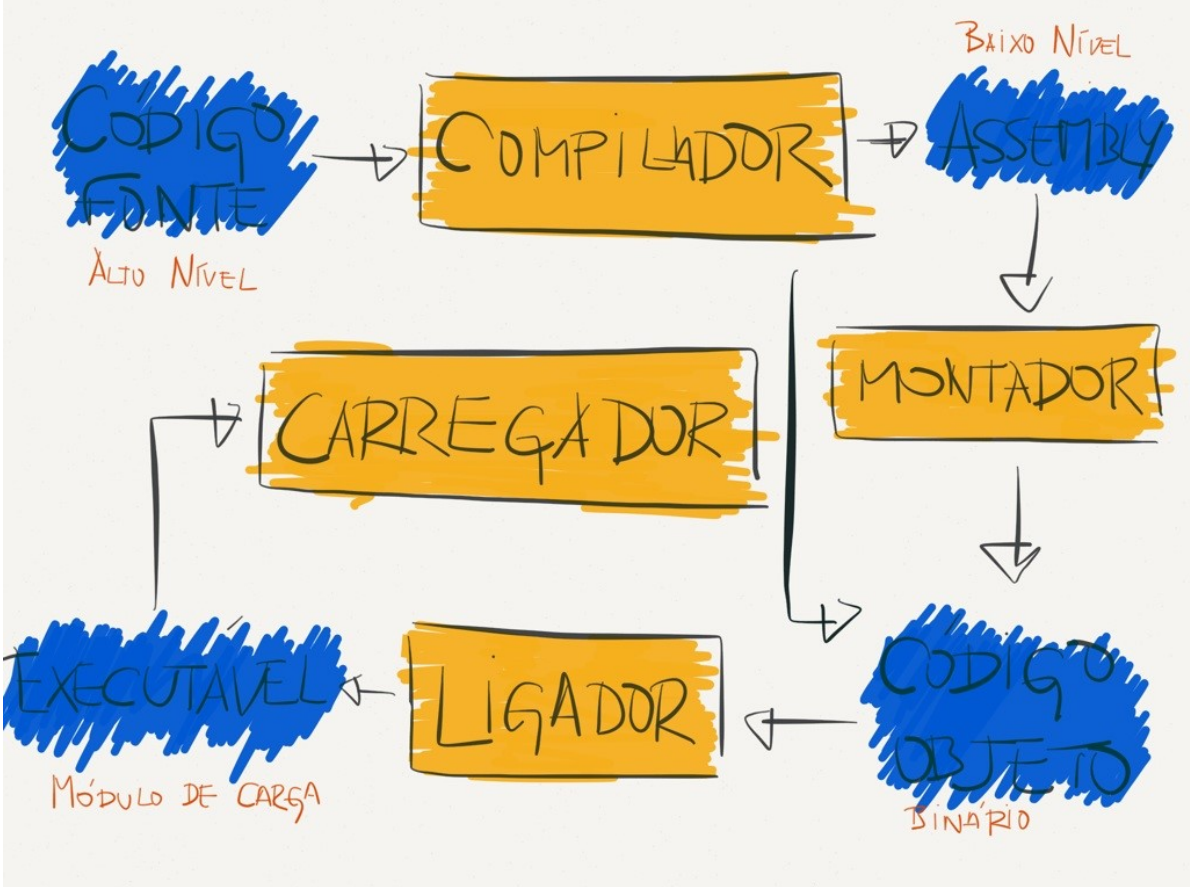
2. (CESPE - 2006 – DATAPREV - Analista de Sistemas) O compilador é parte integrante do kernel de sistemas operacionais.

Comentários:

Não, isso não faz sentido! Eu imagino que essa questão foi formulada porque o GCC faz parte do SO Linux – mas não é regra haver compiladores como parte do kernel de sistemas operacionais. Gabarito: E

3. (CESPE - 2014 – ANATEL - Analista de Sistemas) A compilação é o processo de análise de um programa escrito em linguagem de alto nível, dominando programa-fonte, e sua conversão em um programa equivalente, escrito em linguagem binária de máquina, denominado programa-objeto.

Comentários:



Conforme vimos em aula, está perfeito! O código-fonte entra no compilador e sai um código de montagem ou o próprio código (binário) de máquina. Gabarito: C

4. (CESPE - 2011 – ECT - Analista de Sistemas) Instruções em linguagem de máquina são apresentadas na forma de padrões de bits utilizados para representar as operações internas ao computador. A linguagem de montagem constitui uma versão da linguagem de máquina; cada instrução é representada por uma cadeia de texto que descreve o que a instrução faz. Nesse

processo, o montador é o elemento que converte instruções em linguagem de montagem para linguagem de máquina.

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, o código de montagem transforma a linguagem de montagem em código-objeto ou linguagem (binária) de máquina. Gabarito: C

5. (CESPE - 2007 – TCU - Analista de Sistemas) Uma das principais tarefas do ligador, programa responsável por unir módulos-objeto em um único módulo, denominado módulo de carga, é resolver referências internas e externas.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Essa questão foi anulada! O item sugere que o arquivo de saída do Ligador seja um

módulo pronto a ser executado, e, em algumas situações, isso não ocorre. Gabarito: X

6. (CESPE - 2007 – TCU - Analista de Sistemas) Um interpretador pode ser considerado como um programa que lê um conjunto de instruções e as executa passo a passo. Programas interpretados são, em geral, menores e mais facilmente mantidos, embora sejam mais lentos que os programas compilados.

Comentários:

Com tanto trabalho sendo feito "durante o voo", interpretadores geralmente são muito mais lentos do que compiladores. Pelo menos cinco dos seis passos requeridos por compiladores também devem ser feitos por interpretadores, e estes passos são realizados em "tempo real". Esta abordagem não dá oportunidade para otimização de código.

Conforme vimos em aula, ele é realmente mais lento. Gabarito: C

7. (CESPE - 2007 – TCU - Analista de Sistemas) A análise semântica — responsável por verificar se a estrutura gramatical do programa está correta, ou seja, se essa estrutura foi formada de acordo com as regras gramaticais da linguagem — é uma das tarefas realizadas pelo compilador.

Comentários:

Análise Sintática: também conhecida como Análise Gramatical ou Parsing, analisa uma sequência de entrada para determinar sua estrutura gramatical, segundo uma determinada gramática formal. Ela pega os tokens resultantes do processo de análise léxica e joga em uma estrutura hierárquica, como uma árvore. Assim como no português, verifica-se a estrutura do texto.

Conforme vimos em aula, quem verifica estrutura gramatical é a análise sintática e, não, análise semântica. Gabarito: E

8. (CESPE - 2007 – TSE - Analista de Sistemas – A) Um programa pode ser composto por partes independentemente carregadas e realocadas. Um ligador pode ser usado para resolver as referências aos símbolos externos às partes e para produzir um código executável.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o Ligador tem a função de conectar esses diversos elementos. Gabarito: C

9. (CESPE - 2007 – TSE - Analista de Sistemas – B) Um carregador transfere para a memória códigos a serem executados. Se for transferido um código objeto, tem que ser armazenado nos endereços definidos quando foi gerado, pois um código objeto não pode ser realocado.

Comentários:

Não, um código-objeto pode ser realocado por meio da utilização de carregadores realocáveis, i.e., pode alocar o programa em partes não-fixas da memória. Gabarito: E

10. (CESPE - 2007 – TSE - Analista de Sistemas – C) Para gerar um código objeto, um compilador precisa fazer a análise sintática e semântica de um programa. Para isso ser possível, a semântica da linguagem, mas não a sintaxe, é descrita na notação Backus-Naur Form (BNF).

Comentários:

Não, a Notação BNF é utilizada para descrever a Sintaxe! Gabarito: E

11. (CESPE - 2007 – TSE - Analista de Sistemas – D) Os interpretadores não analisam sintaticamente os códigos fonte uma vez que os traduzem para um formato interno. Por isso, um interpretador traduz um código em menos tempo que um compilador.

Comentários:

Os interpretadores realizam tradução mais rápida que compiladores; no entanto, programas compilados são executados mais rapidamente que programas interpretados. O item está incorreto porque interpretadores analisam sintaticamente os códigos fontes (eles realizam toda a fase de análise, mas não de síntese). Gabarito: E

12. (CESPE - 2008 – FUB - Analista de Sistemas) O compilador pode ser visto como um tradutor, uma vez que o mesmo transforma um programa de linguagem de alto nível para linguagem de baixo nível.

Comentários:

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?

Conforme vimos em aula, essa é uma definição mais ampla de compilação. Gabarito: C

13. (CESPE - 2008 – TRT/5 - Analista de Sistemas) As definições de variáveis no código de um programa não interferem na geração de um código-objeto porque as variáveis não possuem relação direta com a entrada e a saída de dados em um programa.

Comentários:

Como não? Claro que interferem! Entradas e Saídas de dados geralmente são armazenadas em uma variável. Gabarito: E

14. (CESPE - 2008 – ANAC - Analista de Sistemas) O compilador, em contraste com o montador, opera sobre uma linguagem de alto nível, enquanto o montador opera sobre uma linguagem de montagem.

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, é exatamente isso! O Compilador opera sobre uma linguagem de alto nível (entre outras) para traduzir para uma linguagem de máquina, já o Montador opera sobre uma linguagem de montagem (Assembly) para traduzir também para uma linguagem de máquina. Gabarito: C

15. (CESPE - 2008 – ANAC - Analista de Sistemas) A criação da tabela de símbolos constitui tarefa realizada pelo ligador.

Comentários:

Análise Léxica: nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

Conforme vimos em aula, a Tabela de Símbolos é criada por Compiladores durante a Análise Léxica. Gabarito: E

16. (CESPE - 2009 – FINEP - Analista de Sistemas – A) Um código escrito em linguagem de máquina (Assembly) deve ser compilado e ligado antes de ser executado.

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, Assembly é uma linguagem de montagem e, não, de máquina. Além disso, ela não é compilada, mas montada! Gabarito: E

17. (CESPE - 2009 – FINEP - Analista de Sistemas – B) A análise semântica é uma tarefa normalmente realizada pelo link-editor.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o Link-Editor (Ligador) realiza a ligação. Os Compiladores e Interpretadores realizam a análise semântica. Gabarito: E

18. (CESPE - 2009 – FINEP - Analista de Sistemas – C) A otimização de código, feita durante a fase de análise, é uma das tarefas do compilador.

Comentários:

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: Análise ou Front-End (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e Síntese ou Back-end (Otimização de Código Intermediário e Geração de Código Final).

Conforme vimos em aula. de fato, a otimização de código é uma das tarefas do compilador. No entanto, ela é realizada na fase de síntese e, não, de análise. Gabarito: E

19. (CESPE - 2009 – FINEP - Analista de Sistemas – D) Um interpretador é classificado como um tradutor, uma vez que analisa e executa o código. O compilador, por realizar análise e síntese do código, não é considerado um tradutor.

Comentários:

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?

Conforme vimos em aula, compilador também é um tradutor! Compiladores, Montadores e Interpretadores são tradutores. Gabarito: E

20. (CESPE - 2009 – FINEP - Analista de Sistemas – E) A construção da tabela de símbolos é atividade que pode ser iniciada durante a análise léxica.

Comentários:

Análise Léxica: nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

Conforme vimos em aula, é realmente na análise léxica. Gabarito: C

21. (CESPE - 2010 – INMETRO - Analista de Sistemas – B) Todo compilador de linguagem de programação de alto nível tem a responsabilidade de analisar o código fonte até a geração de código executável.

Comentários:

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: Análise ou Front-End (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e Síntese ou Back-end (Otimização de Código Intermediário e Geração de Código Final).

Conforme vimos em aula, a geração do código executável é feita na síntese, portanto ele tem a responsabilidade de analisar o código-fonte até antes da geração do código intermediário. Gabarito: E

22. (CESPE - 2010 – INMETRO - Analista de Sistemas – C) Carregadores são programas usados exclusivamente por linguagens de programação de alto nível, com o objetivo de transferir um módulo de carga para a memória.

Comentários:

São programas que transferem o código de máquina de um módulo objeto para a memória e encaminham o início de sua execução. Eles recebem o módulo de carga como entrada, transferem seu código para a memória e realizam apenas os ajustes de realocação de acordo com o endereço

base de memória. Um programa pode ser composto por partes independentemente carregadas e realocadas.

Não! Carregadores não são usados exclusivamente por linguagens de programação de alto nível.
Gabarito: E

23. (CESPE - 2010 – INMETRO - Analista de Sistemas – D) A declaração de variável "int 7g;" em um programa escrito na linguagem Java, leva a um erro de compilação detectado durante a análise sintática.

Comentários:

Lembrem-se que primeiro é feita a Análise Léxica, depois Análise Sintática e, por último, Análise Semântica. Nesse caso, já dará erro na Análise Léxica! Lembre-se que é nessa fase que se lê o código-fonte, caractere por caractere, e se divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos.

Quando começar a compilação, a análise léxica vai separar "int" e "7g". O Analisador Léxico lerá "7g", verificará se é uma palavra-chave, um literal, um separador, um operador, etc e, como não é, ele o classificará como um identificador. No entanto, identificadores não podem começar com número em Java, logo o compilador indicará um erro logo na Análise Léxica.

Lembrando que alguns números permitem, sim, sufixos – por exemplo: long (39832L), float (2.4f) e double (-7.832d). No entanto, a letra "g" não é um sufixo permitido em Java. Gabarito: E

24. (CESPE - 2010 – INMETRO - Analista de Sistemas – E) Em programas que usam funções disponíveis em bibliotecas, as referências a estas funções serão resolvidas pelo ligador. No caso de bibliotecas estáticas, o código objeto das funções é integrado ao módulo executável durante o processo de ligação.

Comentários:

Perfeito! Executáveis e Bibliotecas fazem referências mútuas conhecidas como ligações, tarefa tipicamente realizada por um ligador. Lembrando que Bibliotecas Dinâmicas são aquelas que podem ser compartilhadas com vários programas e Bibliotecas Estáticas são aquelas que podem ser ligadas somente a um programa individualmente. Gabarito: C

25. (CESPE - 2010 – INMETRO - Analista de Sistemas) A análise léxica/sintática de uma linguagem que tem palavras reservadas tende a ser mais complexa que a de linguagens que têm apenas palavras-chave usadas também como identificadores.

Comentários:

Não, isso não interfere na complexidade da análise léxica/sintática. Gabarito: E

26. (CESPE - 2010 – TRE/MT - Analista de Sistemas) Durante a compilação de um código-fonte, a fase do compilador que é responsável por produzir uma sequência de tokens é a:

a) análise léxica.
b) análise semântica.
c) análise sintática.
d) geração de código executável.
e) verificação de tipos.

Comentários:

Análise Léxica: nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

Conforme vimos em aula, a tokenização ocorre na Análise Léxica! Gabarito: A

27. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A) Compiladores são projetados para um tipo específico de hardware e de sistema operacional.

Comentários:

Não, compiladores não dependem do hardware. Gabarito: E

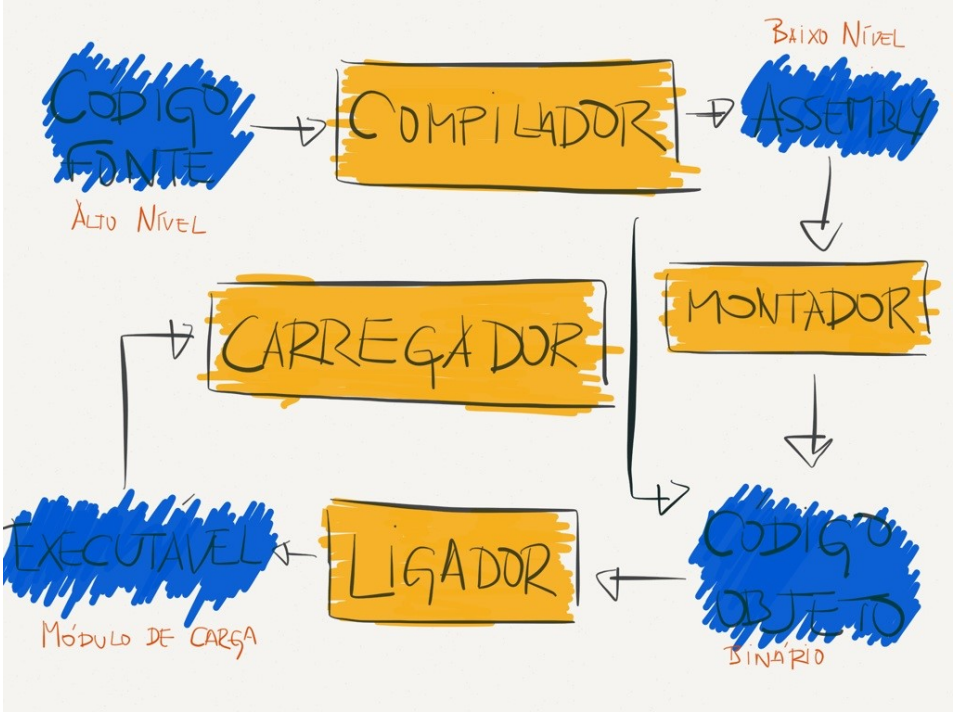
28. (CESPE - 2010 – TRE/MT - Analista de Sistemas – B) A interpretação nada mais é do que uma compilação cruzada.

Comentários:

A compilação cruzada é aquela capaz de produzir código executável em uma plataforma diferente da qual o compilador está sendo executado. Interpretadores não podem ser definidos como uma compilação cruzada. Gabarito: E

29. (CESPE - 2011 – FUB - Analista de Sistemas) Na programação empregando uma linguagem de alto nível, a utilização de um compilador implica o uso de um ligador e de um carregador para a correta execução do programa; por outro lado, a utilização de um interpretador, que simula a existência de um processador cujas instruções são aquelas da linguagem de alto nível empregada, torna desnecessárias as etapas de ligação e carga.

Comentários:



Conforme vimos em aula, a questão está correta. A interpretação torna desnecessárias as duas últimas etapas (Ligação e Carga). Gabarito: C

30. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Em programa escrito em linguagem de alto nível e traduzido por compilador, alguns comandos que fazem parte desse código são instruções da linguagem de programação, enquanto outros comandos são instruções típicas do compilador denominadas diretivas.

Comentários:

O pré-processador é um programa que recebe um texto e efetua conversões léxicas, tais como substituição de macros, inclusão condicional, inclusão de ficheiros e exclusão de comentários. É baseado em Diretivas de Compilação (Ex: #define, #include, etc), que são comandos que não são

compilados, dirigidos ao pré-processador e executado pelo compilador antes do processo de compilação em si.

Conforme vimos em aula, perfeito! Lembram-se das diretivas de compilação? Elas não são compiladas, são pré-processadas antes do processo de compilação em si. Gabarito: C

31. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Durante a execução de um programa que utiliza funções em uma biblioteca dinâmica, esta deve ser carregada em memória e ligada ao processo em tempo de execução.

Comentários:

Existem basicamente dois tipos de bibliotecas, as bibliotecas estáticas e as bibliotecas dinâmicas (ou compartilhadas). Ao compilar um programa que chama uma biblioteca estática, todo o código da biblioteca é copiado e inserido dentro do binário final. O termo "estática" se deve ao fato da linguagem se tornar uma parte estática do binário, não podendo ser compartilhada por outros programas.

Conforme vimos em aula, bibliotecas dinâmicas são ligadas em tempo de execução. Gabarito: C

32. (CESPE - 2011 – TRE/ES - Analista de Sistemas) O link-editor tem a função de vincular os dados de um programa aos programas de sistema e a outros programas de usuário.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o ligador tem a função de vincular diversos componentes. Gabarito: C

33. (CESPE - 2010 – TRF/4 - Analista de Sistemas - A) Interpretadores são programas que convertem códigos escritos em linguagem de alto nível para programas em linguagem de máquina.

Comentários:

Como linguagens compiladas, linguagens interpretadas também têm um relacionamento um-para-muitos entre comandos do código fonte e instruções executáveis de máquina (i.e., nós escrevemos um comando no código-fonte, mas ele pode se transformar em diversas instruções de máquina). E qual seria a diferença principal entre os compiladores e os interpretadores?

Conforme vimos em aula, a questão trata do compilador – ele realmente converte códigos escritos em uma linguagem de alto nível para programas em linguagem de máquina (ou código-objeto)! Já os interpretadores convertem um programa escrito em linguagem de programação de alto nível em código executável. Gabarito: E

34. (CESPE - 2010 – TRF/4 - Analista de Sistemas - B) Linguagens de alto nível cumprem tarefas mais substanciais com um número menor de comandos, mas exigem programas tradutores denominados compiladores para converter programas em linguagem de alto nível para linguagem de máquina.

Comentários:

Perfeito! Linguagens de alto nível, em geral, necessitam de menos comandos para realizar uma mesma tarefa que uma linguagem de baixo nível. Além disso, para converter programas em linguagens de alto nível para linguagem de máquina, é necessário um compilador. Gabarito: C

35. (CESPE - 2010 – TRF/4 - Analista de Sistemas - C) Um computador pode entender qualquer linguagem de máquina, pois a linguagem de máquina não é definida pelo projeto de hardware do computador.

Comentários:

Não! A Linguagem de Montagem é traduzida para uma Linguagem de Máquina (0 e 1) específica de cada processador. Apesar de ser um código binário, há diversas maneiras de interpretá-lo de acordo com o processador (hardware) para o qual ele foi escrito. Gabarito: E

36. (CESPE - 2010 – TRF/4 - Analista de Sistemas - D) Programadores podem escrever instruções em várias linguagens de programação e todas são entendidas diretamente pelos computadores sem a necessidade de tradução.

Comentários:

Claro que não! Elas precisam ser traduzidas, porque computadores só entendem 0 e 1 (0V e 5V). Gabarito: E

37. (CESPE - 2010 – TRF/4 - Analista de Sistemas - E) Softwares escritos em linguagens de máquina são portáveis.

Comentários:

Elas não são portáveis, porque eles dependem da arquitetura do processador. Gabarito: E

38. (CESPE - 2010 – BASA - Analista de Sistemas) Um interpretador é um programa que lê um código escrito em uma linguagem e o converte em um código equivalente em outra linguagem. No processo de conversão, o interpretador relata a presença de erros no código original.

Comentários:

Eu sinceramente não consigo encontrar erros nessa questão. A primeira parte está correta e vale tanto para compiladores quanto para interpretadores. A segunda parte também está correta e também vale para compiladores e interpretadores. Acredito que o examinador deve ter retirado essa frase originalmente falando de um compilador, mudou para interpretador e achou que ia ser errado, mas – para mim – ela continua correta.

Vamos por partes! Ambos leem um código escrito em uma linguagem e o convertem para outra linguagem equivalente. A diferença é que o compilador transforma uma linguagem em uma outra linguagem equivalente que posteriormente deve ser executada; já o interpretador transforma em uma linguagem em uma outra linguagem equivalente que é imediatamente executada. Logo, não há erro nessa primeira parte da questão.

Na segunda parte, ele diz que, no processo de conversão, o interpretador relata a presença de erros no código original. Ora, uma vez que interpretadores leem declarações singulares, eles exibem um erro de cada vez e você terá que consertar esse erro para interpretar a próxima declaração. Já os compiladores exibem todos os erros e alertas de uma vez e, sem consertar todos os erros, o programa não poderá ser executado. Para mim, não há contradição com o que foi dito!

A grande diferença é que o interpretador, em geral, vai se limitar a detectar erros somente de sintaxe da linguagem e dos tipos de variáveis, enquanto os compiladores detectam uma variedade maior de erros. Gabarito: E

39. (CESPE - 2010 – INMETRO - Analista de Sistemas) Um interpretador traduz um programa descrito no nível da linguagem para o nível da máquina, enquanto o compilador eleva a máquina ao nível da linguagem, para que o programa execute a partir da fonte.

Comentários:

Não, a questão claramente inverteu compiladores e interpretadores. Gabarito: E

40. (CESPE - 2011 – ECT - Analista de Sistemas) No programa em linguagem de alto nível, os interpretadores executam os passos definidos para cada instrução e produzem o mesmo

resultado que o do programa compilado. Entretanto, a execução de um programa em linguagem de alto nível com o uso de interpretadores é mais lenta que a execução de um programa compilado, uma vez que precisa examinar cada instrução no programa-fonte, à medida que ela ocorre, e desviar para a rotina que executa a instrução.

Comentários:

Perfeito, essa é uma questão para decorar! Muito bem escrita e boa para sedimentar alguns conceitos! Gabarito: C

41. (CESPE - 2009 – TCE/AC - Analista de Sistemas - A) Os programas escritos em linguagem de programação de alto nível precisam ser convertidos em programas de máquina, sendo o linker um tipo de software básico que efetua essa tradução.

Comentários:

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o Linker (Ligador) não efetua tradução alguma! Quem faz isso é o Compilador! Gabarito: E

42. (CESPE - 2007 – TCU - Analista de Sistemas) Um montador é considerado um software de sistema, responsável pela tradução de uma linguagem de alto nível para uma linguagem de baixo nível (linguagem simbólica).

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, montadores não são softwares de sistema e não realizam a tradução de linguagens de alto nível para uma linguagem de baixo nível. Na verdade, ele traduz linguagens de baixo nível para linguagem de máquina. Gabarito: E

43. (CESPE - 2008 – FUB - Analista de Sistemas) O montador realiza a tarefa de combinar módulos objetos em um módulo de carga.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, é o Ligador que é responsável por receber como entrada os diversos módulos e conectá-los, gerando como saída um único módulo de carga. Gabarito: E

44. (CESPE - 2008 – FUB - Analista de Sistemas – C) Para que um programa possa ser executado, seu código de máquina deve estar presente na memória. O ligador é o programa do sistema responsável por transferir o código de máquina de um módulo objeto para a memória e encaminhar o início de sua execução.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o ligador é o responsável por interligar os diversos módulos de um programa para gerar o programa que será posteriormente carregado para a memória. O carregador é responsável por transferir o código de máquina de um módulo objeto para a memória e encaminhar o início de sua execução. Gabarito: E

45. (CESPE - 2006 – DETRAN - Analista de Sistemas - D) O termo linguagem de máquina é usado para denominar linguagens do tipo assembler ou C++.

Comentários:

Não, Linguagem de Máquina é código binário (0 e 1). Assembler não é linguagem, é um programa chamado Montador. Assembly é uma linguagem de montagem e C++ é uma linguagem de alto nível orientada a objetos. Gabarito: E

QUESTÕES COMENTADAS – COMPILADORES E INTERPRETADORES - MULTIBANCAS

1. (FCC – 2016 – 14ª Região (RO e AC) - Analista Judiciário - Tecnologia da Informação) A compilação é o processo de tradução de um programa escrito em uma linguagem fonte em um programa equivalente em linguagem de máquina. Nesse processo, o programa fonte normalmente passa pelas fases:

   I. Identificação de sequências de caracteres de entrada e produção de uma sequência de elementos de saída, os tokens. Nesta fase, verifica-se se cada caractere do programa fonte pertence ao alfabeto da linguagem, identificando os tokens e desprezando comentários e espaços em branco. Os tokens constituem classes de símbolos, tais como palavras reservadas, delimitadores, identificadores etc.

   II. Identificação de sequências de símbolos que constituem estruturas como expressões e comandos, através de uma varredura, ou parsing, da representação interna do programa fonte, produzindo uma estrutura em árvore, chamada árvore de derivação.

   III. Verificação das estruturas quanto ao sentido, ou seja, se o programa não possui erros de significado. Por exemplo, verifica se um identificador declarado como variável é utilizado como tal, se existe compatibilidade entre operandos e operadores em expressões etc.

Os itens I, II e III referem-se, correta e respectivamente, às fases:

a) Análise Léxica - Análise Sintática - Análise Semântica.

b) Interpretação - Análise Sintática - Montagem.

c) Busca Binária - Montagem Léxica - Análise Semântica.

d) Classificação - Análise Léxica - Montagem.

e) Identificação Inicial - Análise Estrutural - Geração de Código.

Comentários:

I – Refere a identificação de tokens, claramente Análise Léxica, como vimos, nessa etapa se lê o código-fonte, caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens;

II – identificação de expressões e comandos, a partir dos tokens identificados na fase anterior (análise léxica) ocorrem na análise sintática;

III – semântica = sentido. Nessa fase, após a identificação dos tokens e da análise sintática, que avalia a corretude da construção da linguagem; Gabarito: A

2. (FCC – 2014 – TCE-GO - Analista de Controle Externo - Tecnologia da Informação) Compiladores, montadores e ligadores são softwares que convertem programas de um formato de código (entrada) para um mais próximo ao formato executável compreendido pela máquina (saída). Os ligadores geram como saída:

a) programas objeto.

b) bibliotecas de programas semicompilados.

c) programas em formato bytecode.

d) programas executáveis em linguagem de máquina.

e) programas compilados em código intermediário, mas ainda não executáveis.

Comentários:

Como vimos na aula teórica, os ligadores transformam código objeto em (binário) em código executável em linguagem de máquina. Gabarito: D

3. (FCC – 2014 – SABESP - Tecnólogo - Sistemas) Uma sub-rotina, escrita numa linguagem de programação, que chama a si mesma, direta ou indiretamente, é dita I. O uso de II geralmente permite uma descrição mais clara e concisa dos algoritmos, especialmente quando o problema tem esta característica por natureza, como é o caso III , por exemplo. Um compilador implementa este tipo de sub-rotina por meio de uma IV , na qual são armazenados os dados usados em cada chamada da sub-rotina que ainda não terminou de processar.

As lacunas são correta e, respectivamente, preenchidas por:

a)

I	II	III	IV
recursiva	recursividade	do fatorial	pilha

b)

autoexecutável	autoexecuções	das árvores	pilha encadeada

c)

loop	*looping*	da pesquisa binária	árvore de decisão

d)

recursiva	recursão	dos vetores	fila

e)

loop	*looping*	da série de Fibonacci	árvore binária

Comentários:

Vimos nas aulas anteriores que uma rotina que chama a si mesma é dita "recursiva" (lacuna I)! O uso da "recursividade" deixa o código mais limpo, como vimos (nem sempre mais eficiente!!), lacuna II, portanto. Exemplo de recursividade que aparecem nos itens são: fatorial, série de Fibonacci e pesquisas binárias, ou seja, qualquer delas poderia preencher a lacuna III. A recursividade é implementada utilizando a estrutura de dados "pilha". Isto porque a execução dos níveis de recursividade se dá na ordem inversa das chamadas, ou seja, LIFO. Preenchendo todas as lacunas temos: recursiva, recursividade, fatorial e pilha. Gabarito: A

4. (FCC – 2012 – TJ-PE – Técnico Judiciário) No contexto do módulo executável de um programa de computador, menor tempo de execução, menor consumo de memória, maior tempo na execução de loop's, e menor dificuldade de identificação de erros estão associados, respectivamente, aos métodos

a) compilação, interpretação, compilação, interpretação.

b) interpretação, interpretação, interpretação, compilação.

c) interpretação, interpretação, compilação, compilação.

d) interpretação, compilação, interpretação, compilação.

e) compilação, compilação, interpretação, interpretação.

Comentários:

São características dos compiladores: ter execução mais rápida (daí a vantagem de ter uma etapa anterior) e menor consumo de memória (código compilado em arquivo executável é mais eficiente).

Um código já compilado, porém, dificulta descobrir onde está o erro no código-fonte, o que é facilitado por linguagens interpretadas, pois não há código intermediário. Da mesma forma, por não ter uma análise prévia, como é o caso de linguagens compiladas, as interpretadas têm um tempo de execução em loops maior, pois é mais difícil realizar otimizações sem uma etapa anterior para fazê-lo; Gabarito: E

5. (FCC – 2012 – TRE-SP – Analista Judiciário - Análise de Sistemas) Analise o texto:

Na compilação, a análise consiste em três fases. Em uma das fases, os caracteres ou tokens são agrupados hierarquicamente em coleções aninhadas com significado coletivo. Essa fase envolve o agrupamento dos tokens do programa fonte em frases gramaticais, que são usadas pelo compilador, a fim de sintetizar a saída. Usualmente, as frases gramaticais do programa fonte são representadas por uma árvore gramatical.

A fase citada no texto é conhecida como análise:

a) sintática.

b) semântica.

c) léxica.

d) binária.

e) linear.

Comentários:

Quando a questão comenta sobre organização de tokens em estruturas, sabemos que se trata da análise sintática. Não poderia ser análise léxica, pois esta é responsável pela identificação dos tokens, nem da semântica, que analisa se o código faz sentido com base em regras semânticas da linguagem. Gabarito: A

6. (FGV – 2014 – SUSAM – Analista de Sistemas) Programa destinado a transformar um código escrito em linguagem de alto nível em uma linguagem Assembly é o:

a) debugger.

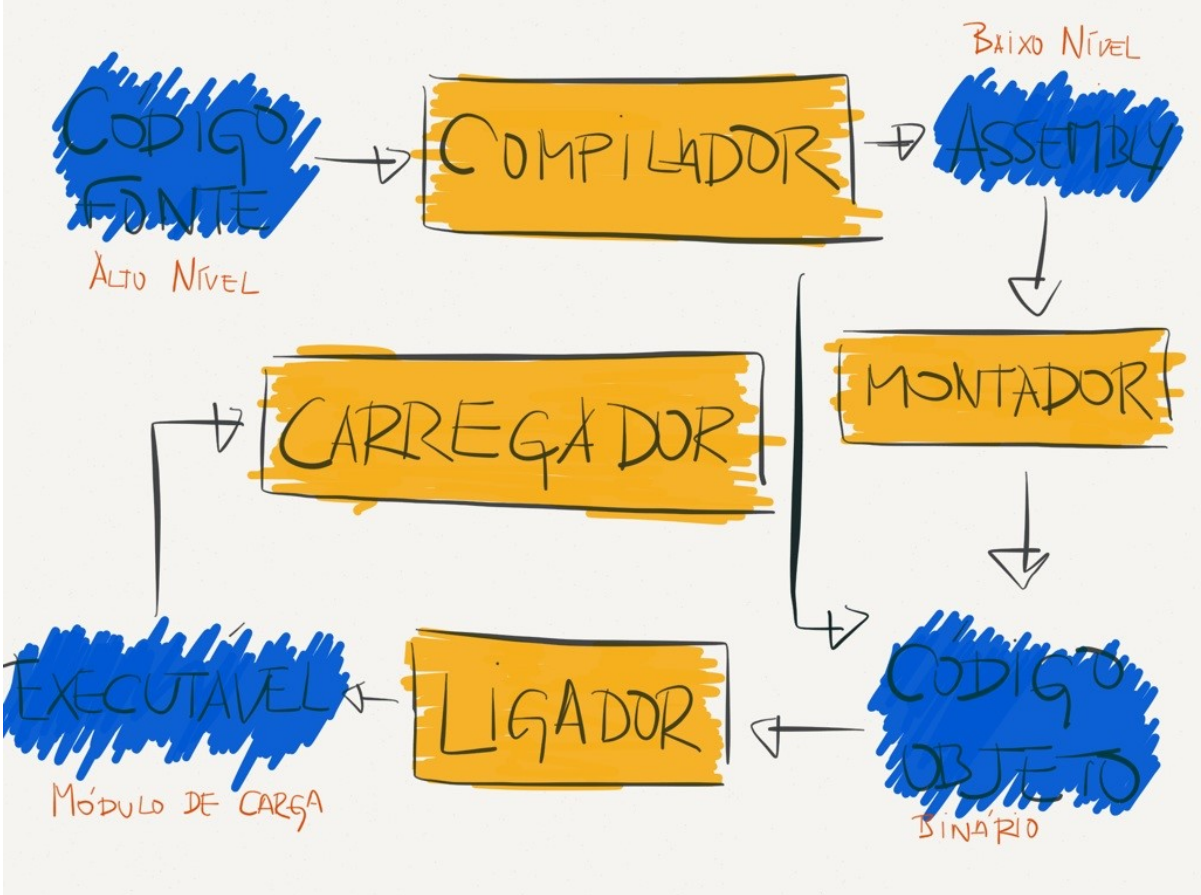
b) compilador.

c) montador.

d) fortran.

e) otimizador.

Comentários:



Conforme vimos em aula, trata-se do compilador. Gabarito: B

7. (FGV – 2015 – TCE/SE – Analista de Sistemas) O módulo de análise léxica de um compilador tem por objetivo:

a) verificar se o programa fonte obedece às regras da gramática da linguagem;

b) agrupar os caracteres do programa fonte em unidades denominadas tokens;

c) gerar o código objeto correspondente à tradução do programa fonte para alguma forma intermediária de representação;

d) construir as árvores sintáticas dos diversos comandos do programa fonte;

e) eliminar comandos supérfluos do programa fonte.

Comentários:

Análise Léxica: nesta fase, um analisador léxico (ou scanner) lê o código-fonte, caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Ele descarta comentários, espaços em branco, marcas de formatação e produz uma sequência de palavras-chaves, pontuações e nomes. Terminada essa fase, vamos para a próxima!

Conforme vimos em aula, trata-se do agrupamento de unidades em tokens.Gabarito: B

8. (FGV – 2010 – DETRAN/RN – Analista de Sistemas) As etapas realizadas durante a programação em uma linguagem de alto nível, para se gerar um código executável, são:

a) Programa fonte, compilação, interpretação, ligação, código executável.

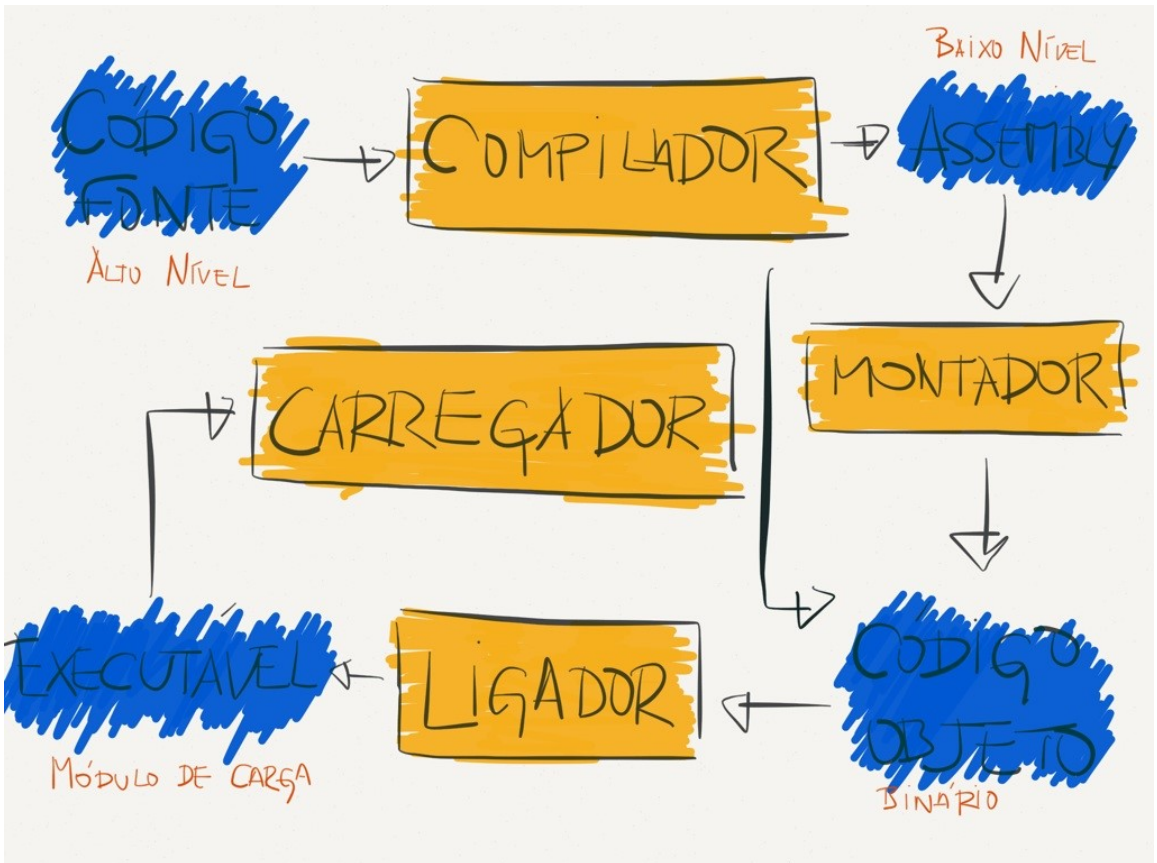
b) Programa fonte, compilação, código-objeto, ligação, código executável.

c) Programa fonte, interpretação, código-objeto, ligação, código executável.

d) Programa fonte, montagem, compilação, código-objeto, código executável.

e) Programa fonte, montagem, código-objeto, ligação, interpretação.

Comentários:



Conforme eu disse em aula, a maioria dos compiladores contém um montador. Dessa forma, podemos afirmar que ordem é: programa fonte (código-fonte) → compilação → código-objeto → ligação → código executável. Gabarito: B

9. (FGV – 2010 – DETRAN/RN – Analista de Sistemas) São funções realizadas pelo módulo front-end de um compilador:

a) Análise léxica e análise lógica.

b) Análise sintática, análise léxica e análise lógica.

c) Análise sintática e análise lógica.

d) Análise semântica, análise léxica e análise sintática.

e) Análise semântica e análise lógica.

Comentários:

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: Análise ou Front-End (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e Síntese ou Back-end (Otimização de Código Intermediário e Geração de Código Final).

Conforme vimos em aula, o front-end inclui análise semântica, léxica e sintática. Gabarito: D

LISTA DE QUESTÕES – COMPILADORES E INTERPRETADORES - CEBRASPE

1. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Uma diferença fundamental entre um compilador e um montador é que o compilador gera um arquivo executável a partir de um arquivo texto com o programa, enquanto o montador executa diretamente a descrição assembler, sem gerar arquivo na saída.

2. (CESPE - 2006 – DATAPREV - Analista de Sistemas) O compilador é parte integrante do kernel de sistemas operacionais.

3. (CESPE - 2014 – ANATEL - Analista de Sistemas) A compilação é o processo de análise de um programa escrito em linguagem de alto nível, dominando programa-fonte, e sua conversão em um programa equivalente, escrito em linguagem binária de máquina, denominado programa-objeto.

4. (CESPE - 2011 – ECT - Analista de Sistemas) Instruções em linguagem de máquina são apresentadas na forma de padrões de bits utilizados para representar as operações internas ao computador. A linguagem de montagem constitui uma versão da linguagem de máquina; cada instrução é representada por uma cadeia de texto que descreve o que a instrução faz. Nesse processo, o montador é o elemento que converte instruções em linguagem de montagem para linguagem de máquina.

5. (CESPE - 2007 – TCU - Analista de Sistemas) Uma das principais tarefas do ligador, programa responsável por unir módulos-objeto em um único módulo, denominado módulo de carga, é resolver referências internas e externas.

6. (CESPE - 2007 – TCU - Analista de Sistemas) Um interpretador pode ser considerado como um programa que lê um conjunto de instruções e as executa passo a passo. Programas interpretados são, em geral, menores e mais facilmente mantidos, embora sejam mais lentos que os programas compilados.

7. (CESPE - 2007 – TCU - Analista de Sistemas) A análise semântica — responsável por verificar se a estrutura gramatical do programa está correta, ou seja, se essa estrutura foi formada de acordo com as regras gramaticais da linguagem — é uma das tarefas realizadas pelo compilador.

8. (CESPE - 2007 – TSE - Analista de Sistemas – A) Um programa pode ser composto por partes independentemente carregadas e realocadas. Um ligador pode ser usado para resolver as referências aos símbolos externos às partes e para produzir um código executável.

9. (CESPE - 2007 – TSE - Analista de Sistemas – B) Um carregador transfere para a memória códigos a serem executados. Se for transferido um código objeto, tem que ser armazenado nos endereços definidos quando foi gerado, pois um código objeto não pode ser realocado.

10. (CESPE - 2007 – TSE - Analista de Sistemas – C) Para gerar um código objeto, um compilador precisa fazer a análise sintática e semântica de um programa. Para isso ser possível, a semântica da linguagem, mas não a sintaxe, é descrita na notação Backus-Naur Form (BNF).

11. (CESPE - 2007 – TSE - Analista de Sistemas – D) Os interpretadores não analisam sintaticamente os códigos fonte uma vez que os traduzem para um formato interno. Por isso, um interpretador traduz um código em menos tempo que um compilador.

12. (CESPE - 2008 – FUB - Analista de Sistemas) O compilador pode ser visto como um tradutor, uma vez que o mesmo transforma um programa de linguagem de alto nível para linguagem de baixo nível.

13. (CESPE - 2008 – TRT/5 - Analista de Sistemas) As definições de variáveis no código de um programa não interferem na geração de um código-objeto porque as variáveis não possuem relação direta com a entrada e a saída de dados em um programa.

14. (CESPE - 2008 – ANAC - Analista de Sistemas) O compilador, em contraste com o montador, opera sobre uma linguagem de alto nível, enquanto o montador opera sobre uma linguagem de montagem.

15. (CESPE - 2008 – ANAC - Analista de Sistemas) A criação da tabela de símbolos constitui tarefa realizada pelo ligador.

16. (CESPE - 2009 – FINEP - Analista de Sistemas – A) Um código escrito em linguagem de máquina (Assembly) deve ser compilado e ligado antes de ser executado.

17. (CESPE - 2009 – FINEP - Analista de Sistemas – B) A análise semântica é uma tarefa normalmente realizada pelo link-editor.

18. (CESPE - 2009 – FINEP - Analista de Sistemas – C) A otimização de código, feita durante a fase de análise, é uma das tarefas do compilador.

19. (CESPE - 2009 – FINEP - Analista de Sistemas – D) Um interpretador é classificado como um tradutor, uma vez que analisa e executa o código. O compilador, por realizar análise e síntese do código, não é considerado um tradutor.

20. (CESPE - 2009 – FINEP - Analista de Sistemas – E) A construção da tabela de símbolos é atividade que pode ser iniciada durante a análise léxica.

21. (CESPE - 2010 – INMETRO - Analista de Sistemas – B) Todo compilador de linguagem de programação de alto nível tem a responsabilidade de analisar o código fonte até a geração de código executável.

22. (CESPE - 2010 – INMETRO - Analista de Sistemas – C) Carregadores são programas usados exclusivamente por linguagens de programação de alto nível, com o objetivo de transferir um módulo de carga para a memória.

23. (CESPE - 2010 – INMETRO - Analista de Sistemas – D) A declaração de variável "int 7g;" em um programa escrito na linguagem Java, leva a um erro de compilação detectado durante a análise sintática.

24. (CESPE - 2010 – INMETRO - Analista de Sistemas – E) Em programas que usam funções disponíveis em bibliotecas, as referências a estas funções serão resolvidas pelo ligador. No caso de bibliotecas estáticas, o código objeto das funções é integrado ao módulo executável durante o processo de ligação.

25. (CESPE - 2010 – INMETRO - Analista de Sistemas) A análise léxica/sintática de uma linguagem que tem palavras reservadas tende a ser mais complexa que a de linguagens que têm apenas palavras-chave usadas também como identificadores.

26. (CESPE - 2010 – TRE/MT - Analista de Sistemas) Durante a compilação de um código-fonte, a fase do compilador que é responsável por produzir uma sequência de tokens é a:

a) análise léxica.

b) análise semântica.

c) análise sintática.

d) geração de código executável.

e) verificação de tipos.

27. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A) Compiladores são projetados para um tipo específico de hardware e de sistema operacional.

28. (CESPE - 2010 – TRE/MT - Analista de Sistemas – B) A interpretação nada mais é do que uma compilação cruzada.

29. (CESPE - 2011 – FUB - Analista de Sistemas) Na programação empregando uma linguagem de alto nível, a utilização de um compilador implica o uso de um ligador e de um carregador para a correta execução do programa; por outro lado, a utilização de um interpretador, que simula a existência de um processador cujas instruções são aquelas da linguagem de alto nível empregada, torna desnecessárias as etapas de ligação e carga.

30. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Em programa escrito em linguagem de alto nível e traduzido por compilador, alguns comandos que fazem parte desse código são instruções da linguagem de programação, enquanto outros comandos são instruções típicas do compilador denominadas diretivas.

31. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Durante a execução de um programa que utiliza funções em uma biblioteca dinâmica, esta deve ser carregada em memória e ligada ao processo em tempo de execução.

32. (CESPE - 2011 – TRE/ES - Analista de Sistemas) O link-editor tem a função de vincular os dados de um programa aos programas de sistema e a outros programas de usuário.

33. (CESPE - 2010 – TRF/4 - Analista de Sistemas - A) Interpretadores são programas que convertem códigos escritos em linguagem de alto nível para programas em linguagem de máquina.

34. (CESPE - 2010 – TRF/4 - Analista de Sistemas - B) Linguagens de alto nível cumprem tarefas mais substanciais com um número menor de comandos, mas exigem programas tradutores

denominados compiladores para converter programas em linguagem de alto nível para linguagem de máquina.

35. (CESPE - 2010 – TRF/4 - Analista de Sistemas - C) Um computador pode entender qualquer linguagem de máquina, pois a linguagem de máquina não é definida pelo projeto de hardware do computador.

36. (CESPE - 2010 – TRF/4 - Analista de Sistemas - D) Programadores podem escrever instruções em várias linguagens de programação e todas são entendidas diretamente pelos computadores sem a necessidade de tradução.

37. (CESPE - 2010 – TRF/4 - Analista de Sistemas - E) Softwares escritos em linguagens de máquina são portáveis.

38. (CESPE - 2010 – BASA - Analista de Sistemas) Um interpretador é um programa que lê um código escrito em uma linguagem e o converte em um código equivalente em outra linguagem. No processo de conversão, o interpretador relata a presença de erros no código original.

39. (CESPE - 2010 – INMETRO - Analista de Sistemas) Um interpretador traduz um programa descrito no nível da linguagem para o nível da máquina, enquanto o compilador eleva a máquina ao nível da linguagem, para que o programa execute a partir da fonte.

40. (CESPE - 2011 – ECT - Analista de Sistemas) No programa em linguagem de alto nível, os interpretadores executam os passos definidos para cada instrução e produzem o mesmo resultado que o do programa compilado. Entretanto, a execução de um programa em linguagem de alto nível com o uso de interpretadores é mais lenta que a execução de um programa compilado, uma vez que precisa examinar cada instrução no programa-fonte, à medida que ela ocorre, e desviar para a rotina que executa a instrução.

41. (CESPE - 2009 – TCE/AC - Analista de Sistemas - A) Os programas escritos em linguagem de programação de alto nível precisam ser convertidos em programas de máquina, sendo o linker um tipo de software básico que efetua essa tradução.

42. (CESPE - 2007 – TCU - Analista de Sistemas) Um montador é considerado um software de sistema, responsável pela tradução de uma linguagem de alto nível para uma linguagem de baixo nível (linguagem simbólica).

43. (CESPE - 2008 – FUB - Analista de Sistemas) O montador realiza a tarefa de combinar módulos objetos em um módulo de carga.

44. (CESPE - 2008 – FUB - Analista de Sistemas – C) Para que um programa possa ser executado, seu código de máquina deve estar presente na memória. O ligador é o programa do sistema responsável por transferir o código de máquina de um módulo objeto para a memória e encaminhar o início de sua execução.

45. (CESPE - 2006 – DETRAN - Analista de Sistemas - D) O termo linguagem de máquina é usado para denominar linguagens do tipo assembler ou C++.

GABARITO

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

LISTA DE QUESTÕES – COMPILADORES E INTERPRETADORES - MULTIBANCAS

1. (FCC – 2016 – 14ª Região (RO e AC) - Analista Judiciário - Tecnologia da Informação) A compilação é o processo de tradução de um programa escrito em uma linguagem fonte em um programa equivalente em linguagem de máquina. Nesse processo, o programa fonte normalmente passa pelas fases:

   I. Identificação de sequências de caracteres de entrada e produção de uma sequência de elementos de saída, os tokens. Nesta fase, verifica-se se cada caractere do programa fonte pertence ao alfabeto da linguagem, identificando os tokens e desprezando comentários e espaços em branco. Os tokens constituem classes de símbolos, tais como palavras reservadas, delimitadores, identificadores etc.

   II. Identificação de sequências de símbolos que constituem estruturas como expressões e comandos, através de uma varredura, ou parsing, da representação interna do programa fonte, produzindo uma estrutura em árvore, chamada árvore de derivação.

   III. Verificação das estruturas quanto ao sentido, ou seja, se o programa não possui erros de significado. Por exemplo, verifica se um identificador declarado como variável é utilizado como tal, se existe compatibilidade entre operandos e operadores em expressões etc.

Os itens I, II e III referem-se, correta e respectivamente, às fases

a) Análise Léxica - Análise Sintática - Análise Semântica.

b) Interpretação - Análise Sintática - Montagem.

c) Busca Binária - Montagem Léxica - Análise Semântica.

d) Classificação - Análise Léxica - Montagem.

e) Identificação Inicial - Análise Estrutural - Geração de Código.

2. (FCC – 2014 – TCE-GO - Analista de Controle Externo - Tecnologia da Informação) Compiladores, montadores e ligadores são softwares que convertem programas de um formato de código (entrada) para um mais próximo ao formato executável compreendido pela máquina (saída). Os ligadores geram como saída:

a) programas objeto.

b) bibliotecas de programas semicompilados.

c) programas em formato bytecode.

d) programas executáveis em linguagem de máquina.

e) programas compilados em código intermediário, mas ainda não executáveis.

3. (FCC – 2014 – SABESP - Tecnólogo - Sistemas) Uma sub-rotina, escrita numa linguagem de programação, que chama a si mesma, direta ou indiretamente, é dita I. O uso de II geralmente permite uma descrição mais clara e concisa dos algoritmos, especialmente quando o problema tem esta característica por natureza, como é o caso III , por exemplo. Um compilador implementa este tipo de sub-rotina por meio de uma IV , na qual são armazenados os dados usados em cada chamada da sub-rotina que ainda não terminou de processar.

   As lacunas são correta e, respectivamente, preenchidas por:

   a)

I	II	III	IV
recursiva	recursividade	do fatorial	pilha

   b)

autoexecutável	autoexecuções	das árvores	pilha encadeada

   c)

loop	*looping*	da pesquisa binária	árvore de decisão

   d)

recursiva	recursão	dos vetores	fila

   e)

loop	*looping*	da série de Fibonacci	árvore binária

4. (FCC – 2012 – TJ-PE – Técnico Judiciário) No contexto do módulo executável de um programa de computador, menor tempo de execução, menor consumo de memória, maior tempo na execução de loop's, e menor dificuldade de identificação de erros estão associados, respectivamente, aos métodos

   a) compilação, interpretação, compilação, interpretação.

   b) interpretação, interpretação, interpretação, compilação.

   c) interpretação, interpretação, compilação, compilação.

   d) interpretação, compilação, interpretação, compilação.

   e) compilação, compilação, interpretação, interpretação.

5. (FCC – 2012 – TRE-SP – Analista Judiciário - Análise de Sistemas) Analise o texto:

   Na compilação, a análise consiste em três fases. Em uma das fases, os caracteres ou tokens são agrupados hierarquicamente em coleções aninhadas com significado coletivo. Essa fase envolve o agrupamento dos tokens do programa fonte em frases gramaticais, que são usadas pelo

compilador, a fim de sintetizar a saída. Usualmente, as frases gramaticais do programa fonte são representadas por uma árvore gramatical.

A fase citada no texto é conhecida como análise:

a) sintática.

b) semântica.

c) léxica.

d) binária.

e) linear.

6. (FGV – 2014 – SUSAM – Analista de Sistemas) Programa destinado a transformar um código escrito em linguagem de alto nível em uma linguagem Assembly é o:

a) debugger.

b) compilador.

c) montador.

d) fortran.

e) otimizador.

7. (FGV – 2015 – TCE/SE – Analista de Sistemas) O módulo de análise léxica de um compilador tem por objetivo:

a) verificar se o programa fonte obedece às regras da gramática da linguagem;

b) agrupar os caracteres do programa fonte em unidades denominadas tokens;

c) gerar o código objeto correspondente à tradução do programa fonte para alguma forma intermediária de representação;

d) construir as árvores sintáticas dos diversos comandos do programa fonte;

e) eliminar comandos supérfluos do programa fonte.

8. (FGV – 2010 – DETRAN/RN – Analista de Sistemas) As etapas realizadas durante a programação em uma linguagem de alto nível, para se gerar um código executável, são:

a) Programa fonte, compilação, interpretação, ligação, código executável.

b) Programa fonte, compilação, código-objeto, ligação, código executável.

c) Programa fonte, interpretação, código-objeto, ligação, código executável.

d) Programa fonte, montagem, compilação, código-objeto, código executável.

e) Programa fonte, montagem, código-objeto, ligação, interpretação.

9. (FGV – 2010 – DETRAN/RN – Analista de Sistemas) São funções realizadas pelo módulo front-end de um compilador:

a) Análise léxica e análise lógica.

b) Análise sintática, análise léxica e análise lógica.

c) Análise sintática e análise lógica.

d) Análise semântica, análise léxica e análise sintática.

e) Análise semântica e análise lógica.

GABARITO

1. A
2. D
3. A
4. E
5. A
6. B
7. B
8. B
9. D

DEPURADORES

Depuradores! Como viver sem eles? Segundo Hoglund e McGraw (2006), um depurador é aquele programa que se conecta a outro programa e passa a controlá-lo, garantindo analisar o fluxo lógico durante a sua execução. Por meio dele, é possível rastrear códigos, percorrer sua execução através de breakpoints (pontos de parada) e verificar o estado das variáveis na memória, em tempo de execução.

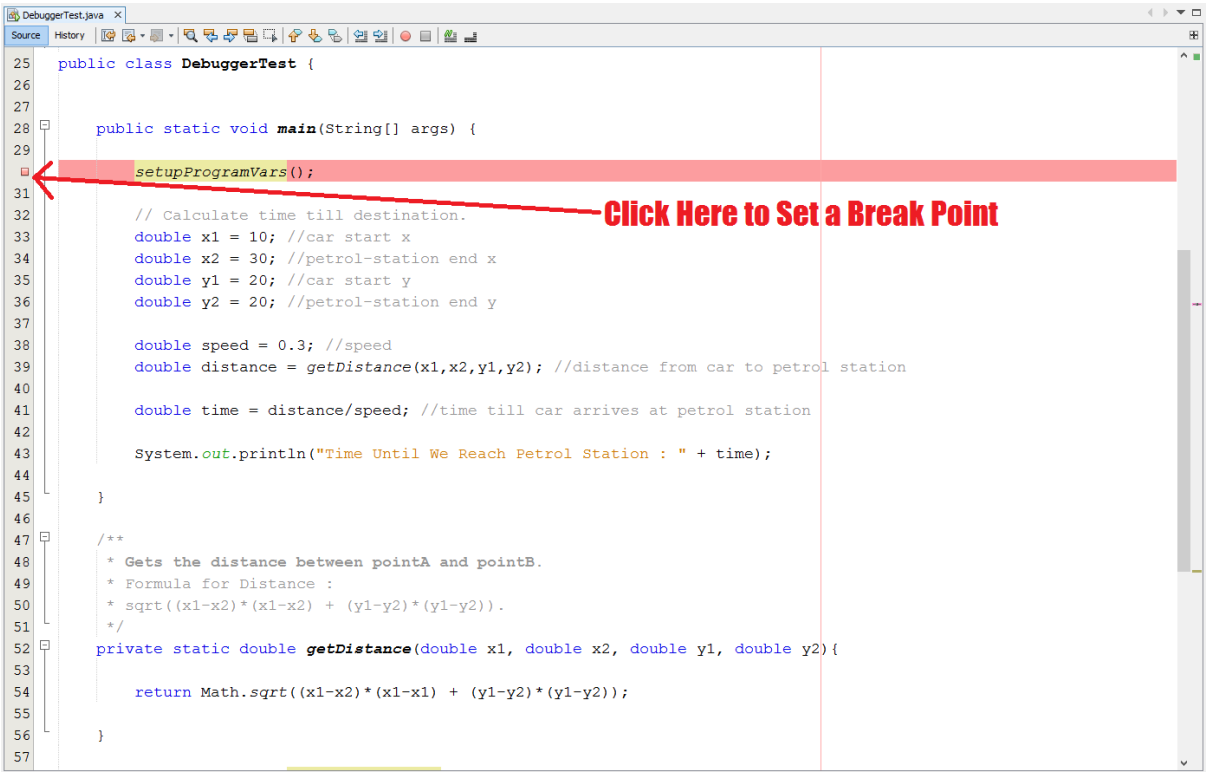
Outra definição afirma que um depurador pode ser definido como um ambiente especializado para controlar e monitorar a execução de um programa, sendo, portanto, uma ferramenta de software utilizada por programadores no desenvolvimento para identificar e corrigir defeitos (bugs) dentro de um programa. Tudo certo até aqui?

Para auxiliar o depurador, informações descrevendo símbolos e tipos no programa, bem como informações para mapear linhas de código-fonte e o código binário pode ser fornecido por um compilador e/ou um interpretador. Essas informações extras permitem que o programador examine os tipos, variáveis e estruturas de dados por nome e siga a execução do programa através do código-fonte.

Essas informações geralmente são referidas como informações de depuração. Com um depurador, os programadores podem "andar passo-a-passo" nas instruções de código do programa, uma de cada vez, enquanto as instruções de máquina correspondentes estão sendo executadas. Assim, podem monitorar mudanças simultaneamente em algumas variáveis, o que ajuda a identificar e corrigir erros.

Com um depurador, um programador também pode definir um ponto de interrupção em uma posição específica do programa. Quando o programa é executado, o depurador irá parar a execução do programa em qualquer ponto de interrupção encontrado, e pode exibir vários elementos de programação, tais como variáveis de software, por exemplo, para auxiliar no processo de depuração.

Vocês entenderam isso? Muitas vezes, o programador passa boa parte de seu tempo procurando erros. O Depurador é essencial para ajudar a encontrá-los! Por que? Porque eu consigo inserir um ponto de interrupção (breakpoint) em qualquer linha de código e visualizar informações de variáveis, entre outras coisas! Quem inventou isso deveria receber um Prêmio Nobel! Vamos ver um exemplo...



Depuradores também podem permitir que os programadores definam pontos de interrupção condicionais (breakpoints condicionais). Um ponto de interrupção condicional é um ponto de interrupção que tem uma expressão booleana associada. Ao encontrá-lo, um depurador só irá parar a execução do programa se a expressão booleana associada for avaliada pelo depurador como verdadeira. É isso...

QUESTÕES COMENTADAS - DEPURADORES - CEBRASPE

1. (CESPE - 2012 - Banco da Amazônia - Análise de Sistemas) Um depurador é definido como um ambiente especializado para controlar e monitorar a execução de um programa. A sua funcionalidade básica consiste na inserção de pontos de parada no código, de forma que, quando o programa esteja parado, o valor corrente das variáveis possa ser verificado.

Comentários:

Outra definição afirma que um depurador pode ser definido como um ambiente especializado para controlar e monitorar a execução de um programa, sendo, portanto, uma ferramenta de software utilizada por programadores no desenvolvimento para identificar e corrigir defeitos (bugs) dentro de um programa. Tudo certo até aqui?

Conforme vimos em aula, a questão está correta! Observe que ela cita também os Pontos de Parada (Breakpoints) como uma forma de visualizar o valor corrente de variáveis. Gabarito: C

2. (CESPE - 2016 – TRE/PI - Analista - Administração de Sistemas – Letra B) Um depurador não permite acompanhar a execução de um programa instrução por instrução. Essa tarefa é executada pelo interpretador.

Comentários:

Essas informações geralmente são referidas como informações de depuração. Com um depurador, os programadores podem "andar passo-a-passo" nas instruções de código do programa, uma de cada vez, enquanto as instruções de máquina correspondentes estão sendo executadas. Assim, podem monitorar mudanças simultaneamente em algumas variáveis, o que ajuda a identificar e corrigir erros.

Conforme vimos em aula, o depurador permite – sim – acompanhar a execução de um programa, instrução por instrução, passo por passo. Além disso, essa tarefa é executada pelo Depurador e, não, pelo Interpretador! Gabarito: E

LISTA DE QUESTÕES - DEPURADORES - CEBRASPE

1. (CESPE - 2012 - Banco da Amazônia - Análise de Sistemas) Um depurador é definido como um ambiente especializado para controlar e monitorar a execução de um programa. A sua funcionalidade básica consiste na inserção de pontos de parada no código, de forma que, quando o programa esteja parado, o valor corrente das variáveis possa ser verificado.

2. (CESPE - 2016 – TRE/PI - Analista - Administração de Sistemas – Letra B) Um depurador não permite acompanhar a execução de um programa instrução por instrução. Essa tarefa é executada pelo interpretador.

GABARITO

1. C
2. E

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.