Estratégia
CONCURSOS

Aul

Desenvolvimento de Sistemas p/ Polícia Federal (Perito - Área 03) Pós-Edital

Professor: Diego Carvalho, Equipe Informática e TI, Judah Reis

AULA 00

SUMÁRIO	PÁGINA
Apresentação	01
- Lógica de Programação	08
- Funções e Procedimentos	39
- Passagem de Parâmetros	45
- Linguagens de Programação	53
- Sistemas de Numeração	59
- Autômatos Finitos	73
Lista de Exercícios Comentados	79
Gabarito	94

O PROFESSOR...

PROF. DIEGO CARVALHO

FORMADO EM CIÊNCIA DA COMPUTAÇÃO PELA UNIVERSIDADE DE BRASÍLIA (UNB), PÓS-GRADUADO EM GESTÃO DE TECNOLOGIA DA INFORMAÇÃO NA ADMINISTRAÇÃO PÚBLICA E, ATUALMENTE, AUDITOR FEDERAL DE FINANÇAS E CONTROLE DA SECRETARIA DO TESOURO NACIONAL.

ESTRATÉGIA CONCURSOS

Já ministrei mais de 120 cursos de Tecnologia da Informação no Estratégia Concursos, incluindo concursos importantes como: TCU; PF; Senado e Câmara Federal; etc. Nossa equipe já possui vasta experiência em concursos, de forma que você não precise procurar mais nenhum outro material de estudos além dos nossos para fazer sua prova.

ENTRE EM CONTATO:

PROFESSORDIEGOCARVALHO@GMAIL.COM

FACEBOOK.COM/PROFESSORDIEGOCARVALHO

ATENÇÃO

ESSA AULA É APENAS UMA **DEMONSTRAÇÃO** DO MÉTODO, ESTILO E ESCRITA DO PROFESSOR. CASO ADQUIRA O CURSO, O RESTANTE DAS AULAS E EXERCÍCIOS SERÃO DISPONIBILIZADOS.

A AVALIAÇÃO DOS PROFESSORES...



Comentário sobre o curso
O curso está excelente para a preparação para o concurso do TCE-SP e outros que tem abordado os mesmos temas. Bem focado nos temas e apresenta os aspectos gerais de cada um deles. O professor comentou muitos exemplos e exercícios. Nem senti falta de vídeos.
Olá professor! de forma geral gostei do curso. Eis alguns pontos para melhorar: - o fato de limitar o tamanho dos parágrafos em poucas linhas facilita, mas acho que não deveriam ser quebrados no meio do mesmo assunto...algumas vezes me perdi... - a observação a respeito do modelo espiral, no qual alguns autores consideram incremental e outros não, poderia ser reforçado no tópico do modelo espiral, mais para o fim da aula (já não lembrava mais desta informação quando cheguei neste modelo...) Desejo muito sucesso!
Achei o material muito bom, objetivo e com muitos exercícios. São resumões sobre os asssuntos, então só vou saber se a profundida é adequada para concursos depois que fizer a prova, mas no geral me pareceu que cobriu bem os temas. Att, Nádia.
Muito bom o curso.
okjok
DADA A QUANTIDADE DE TEMAS ABORDADOS E EXPLICADOS, ACHEI UM CURSO BEM COMPACTO E PRÁTICO.
Apresentou um número de questões comentadas muito bom e com bons esclarecimentos.
Muito bom mesmo. Muitas questões do CESPE mesmo que o concurso fosse da Vunesp. Será que a FCC, FGV e Cesgranrio não tem questões desses assuntos? As questões CESPE são \"diferentes\"... Não tive problemas com isso porque foco o TCU mas fica a dica. Discursivas sobre BI e big data. Sugestões, por favor.
O curso foi excelente e atendeu perfeitamente o meu plano de estudo. Pois precisava de mobilidade e consegui estudar tanto no smartfone , tablet e notebook principalmente no status Off Line porque nem sempre tinha acesso a internet disponível. A didática do professor foi um ponto forte neste curso : A explicação da teoria com uma linguagem clara, cheias de questões comentadas e bom humor como se eu estivesse em uma aula presencial !! Os assuntos-chaves destacados em vermelho otimizou muito as minhas revisões!
Curso com um conteudo excelente, bem produtivo e rico nos detalhes passados que me ajudaram muito.
Gostei muito da linguagem com certeza compraria outros cursos
Não tenho referencia de outros cursos porque este é o primeiro que estou estudando pela internet mas pra mim foi muito bom. A linguagem não é cansativa e acredito que o que se estuda é somente o necessario. Uma coisa que não entendi é que as vezes havia algumas questões que havia a pergunta e a letra da resposta mas não havia as respostas para escolher. Acho que poderia melhorar isso ou então não entendi o conceito.
O professor tinha uma abordagem informal do assunto que facilitava a memorização.
Melhor professor de TI que já vi na área de concursos!

Comentário sobre o curso
Excelente curso! Excelente material, excelente professor, excelente equipe. Recomendo a todos!
"Excelente" é pouco para esse curso. Eu adquiri outras aulas para me preparar para esse concurso, mas nenhuma outra agregou tanto. Imagino o trabalho que deve ter dado para "compilar" todos esses assuntos. Além disso, colocá-los de forma tão didática em aulas relativamente pequenas. Isso é altíssima qualidade! O conteúdo foi muito bem elaborado! E eu reparei isso porque depois que estudei por ele consegui resolver muitas questões que procurei sobre esses assuntos. Questões que antes pareciam ser de "outro mundo" pra mim. A organização da aula também merece destaque... As figuras, os esquemas, as questões, as notas de rodapé e a evolução do conteúdo estudado de acordo com os itens no edital... Sem falar na didática! Quanto ao professor, sem palavras. Respondeu todas minhas dúvidas e sempre se mostrou bastante receptivo. Bom, não sei como irei me sair na prova domingo, mas devo MUITO da minha preparação ao professor Diego Carvalho. Ele nem imagina o quanto me ajudou. Com certeza recomendo esse curso!
Eu me considero uma pessoa leiga em TI, pois sou formado em Engenharia Mecânica e pos graduado em economia, além de nunca ter trabalhado em departamentos de TI nas empresas onde trabalhei. Por isso, apesar de ter tido dificuldades, achei muito adequada a linguagem adotada pelo professor. Os assuntos são muito complexos para serem ensinados a pessoas com pouca base de conhecimento na área, e ainda assim eu entendo que o curso tenha atingido seu objetivo de permitir a transmissão do conhecimento proposto aos alunos. Como pontos de melhoria, sugeriria: - utilização de videoaulas, pois diversos assuntos poderiam ter seu entendimento facilitado com a utilização de imagens - redistribuição na sequência de assuntos. Ainda que eu entenda que vários assuntos sejam interligados, o que dificulta a escolha sobre "o que ensinar primeiro", acho que determinados assuntos poderiam ter sua ordem invertida, ainda que isto não tenha impedido o entendimento da matéria. Muito obrigado. André.
NULL
Excelente curso, muito didático, material de alta qualidade.
Ok

Comentário sobre o curso
òtima didática do Professor Diego!!
"Sempre podemos melhorar" Está muito bom o curso, o pessoal da Estrategia está de parabéns. abs. Airton.
O curso ajuda demais a estruturar e ordenar o conteudo dos cursos, coisa quase impossível de fazer sozinho. Será o primeiro concurso que irei fazer para avaliar realmente a eficácia do curso. Sempre lemabrando que é muito importante e a exempliação da teoria com exemplos práticas reais e ou fictícios que ajudam a fixar o conteúdo, principalmente para aqueles cursos que tem provas discursivas ou questões abertas. Marco Costa
Aulas muito bem explicadas. Respostas muito rápidas. Conteúdo abrangente, focado e muito bem baseado bibliograficamente. Parabéns. Excelente curso.
Esta foi a matéria que mais gostei dentre as abordadas para o concurso do TRT. Gostei muito da linguagem que o professor utilizou e os exercícios no meio do conteúdo deixam a aula mais dinâmica!
Excelente material. Focado para o concurso e muita questão para fixação.
Melhor didática dos pacotes de TI pro TRT.
É uma pena que a última aula chegou tão perto da prova, mas eu entendo a complicação e é fato que o professor fez um trabalho muito bom. Vou revizar o último PDF amanhã, mas até o momento, de longe, foi o melhor material que eu encontrei sobre os outros assuntos. Uma sugestão de melhoria é o próprio site. O sistema de perguntas e respostas é horrível, feio e de baixíssima usabilidade.
Excelente didática! As apostilas dão vontade de ler até quando o assunto não é dos mais legais. Parabéns pelo curso!
O curso é excelente. Sugiro ao Professor lançar um curso regular de Java tratando todos assuntos (programação, JEE, etc.).
Muito bom o material, apostilas com muito exercícios.

O CURSO...

alinhando expectativas...

O curso que eu proponho abrangerá todo o conteúdo do meu cronograma, entretanto é impossível e inviável esgotar cada ponto do edital em uma aula escrita. Como se ministra Java em uma aula? Teríamos uma aula de 800 páginas e não chegaríamos nem perto de matar todo conteúdo! Imaginem agora cada ponto do Edital.



01

Portanto, vou direcioná-los pelo conteúdo da melhor maneira possível. O nosso foco é ter uma visão geral, mas objetiva do que de fato cai em prova e, não, elucubrações sobre cada tema. Meu foco aqui é te fazer passar! Eu sei como é complicado ler muita coisa (ainda mais de TI) e vocês têm outras disciplinas para estudar. Logo, vou ser simples e objetivo! Tranquilo? ;)



02

Além disso, o cronograma será seguido com a maior fidelidade possível, mas ele não é estático e poderá haver alterações no decorrer do curso. Eventualmente, posso tirar o conteúdo de uma aula e colocar em outra de forma que o estudo de vocês fique mais lógico, coeso e fácil de acompanhar; posso também inverter a ordem das aulas (adiantar uma aula e atrasar outra) – sem prejudicá-los.



03

Ademais, vamos usar questões de diversas bancas. Enfim, confiem em mim: o curso vai ajudar bastante! Qualquer dúvida, é só me chamar! Caso haja alguma reclamação, problema, sugestão, comentários, erros de digitação, etc, podem enviar para o nosso fórum que eu tento responder da maneira mais tempestiva possível. Ainda duvidam que PDF não dá certo com Concursos de TI?



04



CONHEÇA O 6º LUGAR – ISS/SALVADOR

www.youtube.com/watch?v=b1w4H3I6mC4#t=1678



CONHEÇA O 1º LUGAR – TRT/RJ

www.facebook.com/video.php?v=790616534367672



CONHEÇA O 3º LUGAR – ALERJ

www.youtube.com/watch?v=SZSvcy4QM8E



CONHEÇA O 1º LUGAR – DATAPREV

www.estrategiaconcursos.com.br/blog/entrevista-andre-furtado-aprovado-em-1o-lugar-no-concurso-dataprev-para-o-cargo-de-analistaarea-de-tecnologia-da-informacao



CONHEÇA O 2º LUGAR – ISS/SALVADOR

www.youtube.com/watch?v=vmUInIJ-aqQ



CONHEÇA NOSSO GRUPO NO FACEBOOK

www.facebook.com/groups/EstrategiaConcursosdeTI



AS AULAS E AS DICAS...

Parágrafos pequenos Observem que os parágrafos têm, no máximo, cinco linhas. Isso serve para que a leitura não fique cansativa e para que vocês não desanimem no meio do material! Para tal, eu tento dividir as disciplinas de maneira que as aulas fiquem objetivas e pequenas (em termos de teoria), mas extensa (em termos de exercícios).	**Visão Geral** Não se atenham a detalhes antes de entender o básico. *Por que?* Ora, não há nada mais irritante do que ir para uma prova que vai cair, por exemplo, RUP, saber vários detalhes, mas não saber as fases e disciplinas. Portanto, caso estejam iniciando os estudos sobre uma matéria, foquem em saber o básico para depois se especializarem.
Destaques em vermelho Quase todos os parágrafos possuem alguma palavra ou frase destacada em negrito e em vermelho. Isso ocorre por suas razões: primeiro, para enfatizar alguma informação importante; segundo, para facilitar a leitura vertical, i.e., após uma primeira leitura, a segunda pode ser passando apenas pelos pontos em destaque.	**Façam muitos exercícios** Ler várias bibliografias é muito trabalhoso e, geralmente, não vale o custo-benefício. Acredito que o que funciona mesmo é entender o básico, depois fazer muitos exercícios e, eventualmente, caso encontrarem algo que não souberem, pesquisem-no separadamente. Além disso, você vai pegando as "manhas" da banca.
Linguagem natural Essa é uma aula para ser lida, o que por si só já pode ser cansativo. Tentarei colocar a linguagem mais coloquial possível, simulando uma conversa. Caso virem frases ou palavras em itálico, ou é uma palavra estrangeira ou é a simulação de uma conversa com vocês. *Tranquilo?*	**Façam resumos** Essa dica somente serve caso vocês tenham disponibilidade. Caso haja pouco tempo para estudar ou pouco tempo até a prova, não compensa! Se não, façam resumos organizados, pois eles economizarão um bom tempo de estudo em suas próximas provas e sempre que descobrirem novas informações, insiram-nas no resumo.
Diversas figuras Essas aulas estarão em constante evolução, sempre à procura de explicar as matérias de maneira mais compreensível e com novas informações/questões. Para tal, na minha opinião, é fundamental a utilização de figuras, gráficos, painéis, etc. Em minha experiência, é bem mais fácil memorizar a partir de imagens.	**Revisem antes da prova** Não adianta querer estudar coisas novas até o último minuto antes da prova e não revisar o que estudou há um mês. Vocês irão esquecer e irão se irritar na hora da prova por não lembrarem de conceitos simples. Tirem uma semana para revisar seus resumos, decorarem algumas coisas e, certamente, irão mais confiantes para a prova.
Fazer Exercícios Muitos exercícios é o meio pelo qual vocês se situarão. *Como assim, professor?* É na hora de fazer os exercícios que vocês descobrirão se estão bem ou mal e avaliarão se precisam estudar mais ou menos. Para tal, há um quadrinho ao final de cada bloco de exercícios para vocês anotarem a quantidade de questões respondidas corretamente ou incorretamente.	**Simulado Final** Ora, fazer um bloco de questões depois de estudar a teoria é tranquilo. No entanto, lembrem-se que a memória de vocês não é infinita e vocês têm um milhão de outras coisas para estudar e decorar. Portanto, se possível, ao fim do curso faremos um simulado com questões escolhidas que foram comentadas dentro das aulas.

LÓGICA DE PROGRAMAÇÃO

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

Professor, o que seria uma instrução? É um conjunto de regras ou normas definidas para a realização ou emprego de algo, indicando ao computador uma ação elementar a ser executada. Bem, esses conceitos são muito básicos e vocês já devem estar bastante acostumados, por isso vamos ver rapidamente. Um conjunto de instruções formam um algoritmo:

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



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

Professor, todos os programas que eu utilizo no meu computador são representações de algoritmos? Sim! *Inclusive o joguinho de Paciência que eu curto?* Sim! *Mas até mesmo os apps que eu utilizo no celular?* Eles também! Todos os softwares (de desktop, notebook, smartphone, tablet, geladeira, relógio, 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, eles entendem presença ou ausência de tensão elétrica), portanto é necessário representar esses algoritmos por meio de uma linguagem de programação.

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

Como assim, professor? Pessoal, um computador é uma grande calculadora. No entanto, ele é "burro", ele só calcula o que mandam-no 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. Todo software possui um código-fonte, que é um conjunto de palavras organizado de acordo com regras específicas, formando um ou mais algoritmos. Essas palavras que formam o algoritmo são escritas utilizando uma linguagem de programação. Esse código-fonte é traduzido e posteriormente executado pelo usuário.

Pessoal... se eu não souber uma linguagem de programação, eu posso escrever um algoritmo utilizando um pseudocódigo! *O que é isso, professor?* 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 pseudo-código, 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

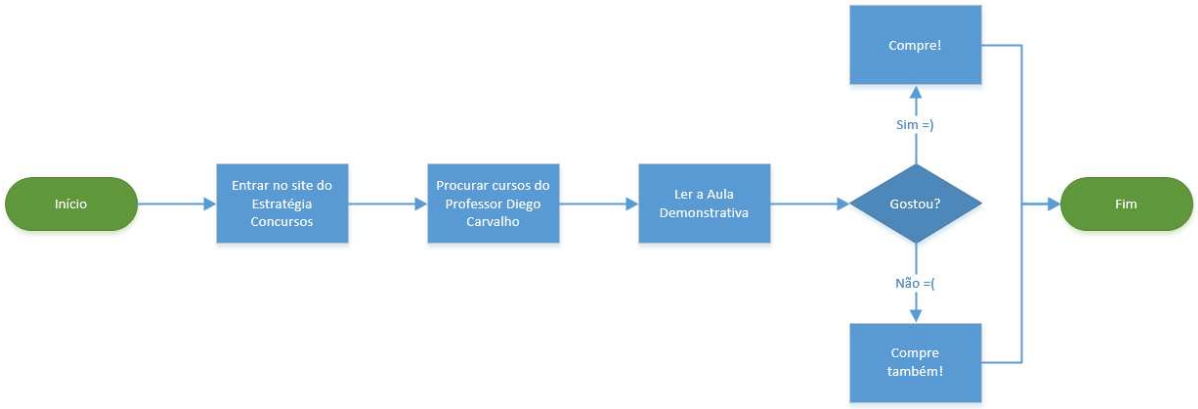
estruturas que têm significado pré-definido, na medida em que deve seguir um padrão. Emprega uma linguagem intermediária entre a linguagem natural e a linguagem de programação para descrever os algoritmos.

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.

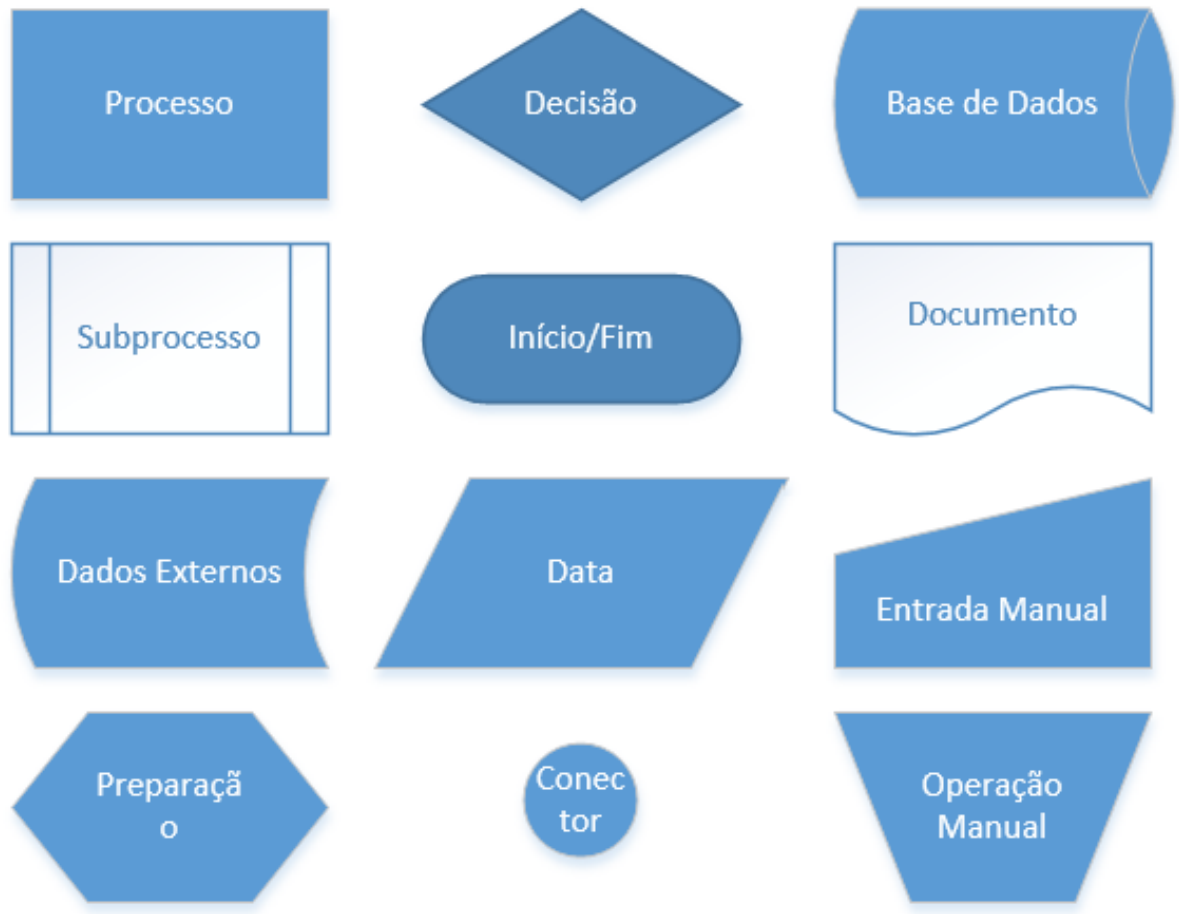
```
início
    <instruções>

    se <teste> então
        <instruções>
    senão
        <instruções>
    fim_se
fim
```

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



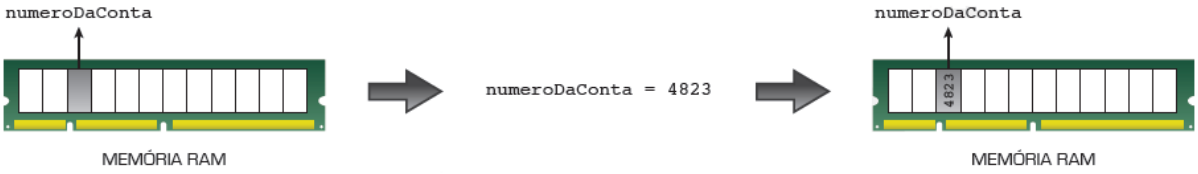
Os principais símbolos de um Fluxograma são:



Bem, nós vimos então que podemos representar algoritmos por meio de Linguagens de Programação, Pseudocódigo 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!

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 Portugol. *O que é isso, professor?* É um pseudocódigo escrito em português! A notação de atribuição é representada por uma seta apontando para a esquerda do valor para o identificador, podendo ser:

```
Variável ← Constante:  dataDeNascimento ← 1988
Variável ← Variável:   endereço ← cidade
Variável ← Expressão:  idade ← (anoAtual – anoDeNascimento)
```

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

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

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

    - Real: também conhecido como Float (Ponto Flutuante), são similares aos números reais da matemática, i.e., possuem parte fracionária. Ex: 3,141592

é a constante de PI; 9,81 m/s² de Aceleração Gravitacional; raiz quadrada de 7; etc.

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

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

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

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

Pessoal, quase todas as linguagens de programação possuem instruções para Leitura e Escrita de dados. A Escrita é uma Saída de Dados (Output) que busca mostrar informações ao usuário na tela do computador (Ex: Escrever (idade)). A Leitura é uma Entrada de Dados (Input) que busca ler dados do usuário por meio teclado (Ex: Ler (idade)). Para manipular dados, utilizamos operadores:

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

Operador	Símbolo	Prioridade
Multiplicação	*	2º
Divisão	/	2º
Adição	+	3º

[1] Uma música em .mp3, um texto em .pdf, uma imagem em jpg são todos tipos estruturados.
[2] Em operadores que possuem a mesma prioridade, o que aparecer primeiro deve ser priorizado! Além disso, parênteses possuem sempre a maior prioridade!

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	>=
Menor ou igual a	<=

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

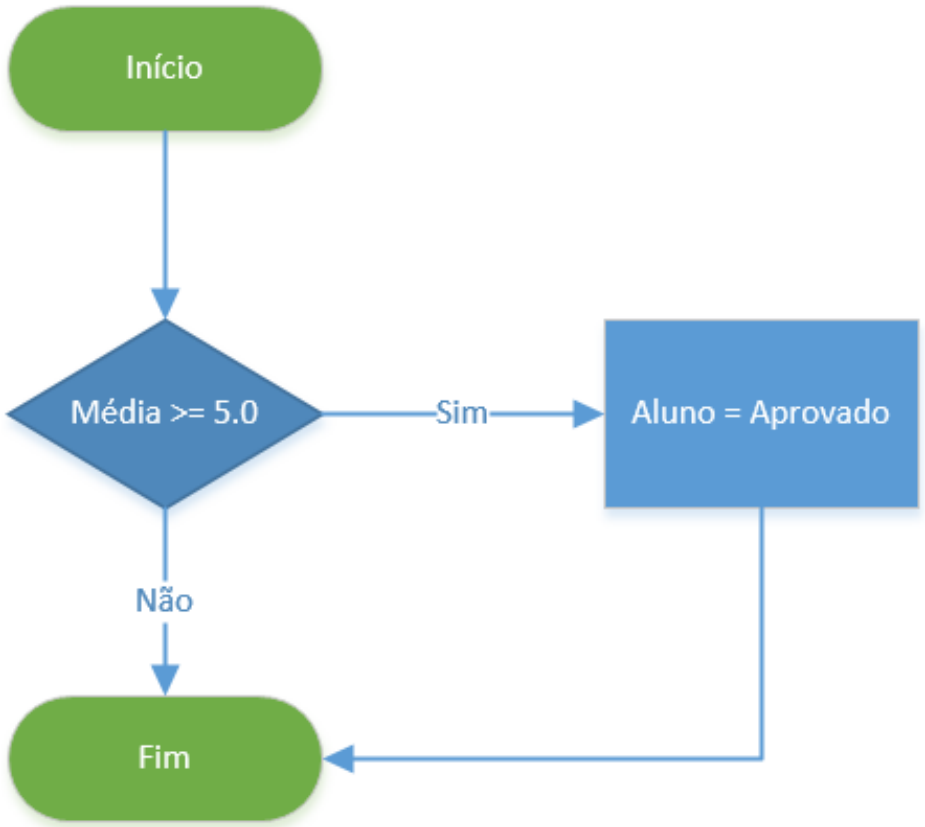
Operador/Símbolo
E/And
Ou/Or
Não/Not

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

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

CASO 1:

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

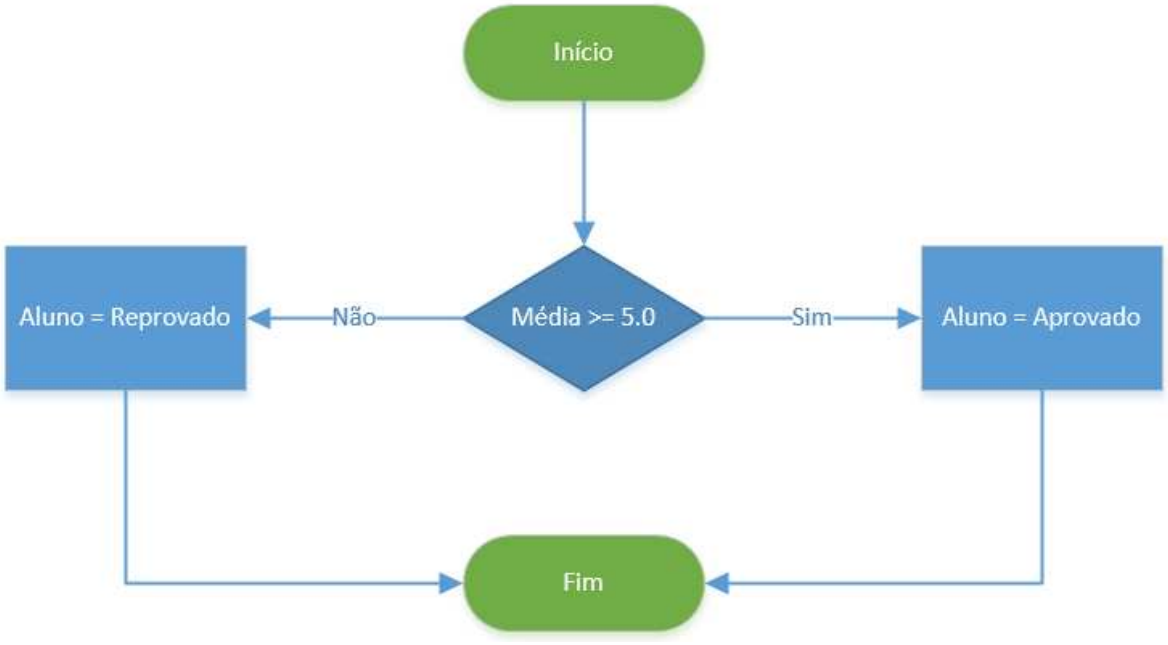


O primeiro caso, e mais simples, é uma estrutura de decisão com um teste (Média >= 5.0). O programa avalia se a variável Média tem o valor maior ou igual a 5.0, no caso de o teste ser verdadeiro a instrução "Aluno = Aprovado" é executada. Em caso negativo, o programa pula a instrução "Aluno = Aprovado"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

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

CASO 2:

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

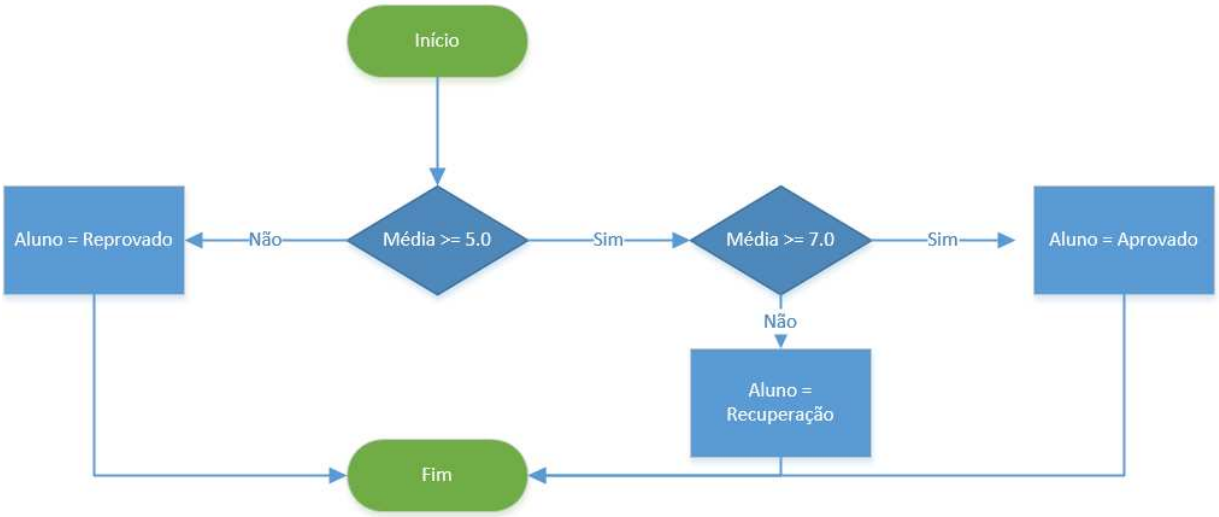


O segundo caso é só um pouquinho mais complexo, onde temos dois fluxos possíveis, um que passa pelo comando "Aluno = Aprovado" e outro que passa pelo "Aluno = Reprovado". Importante ressaltar que, nesse caso, sempre alguma dessas instruções serão executadas, pois, ou Média é maior ou igual a 5.0 ou é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

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

CASO 3:

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



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

O segundo teste avalia a expressão "Média >= 7.0". Chamamos esse tipo de utilização em diferentes níveis de estrutura de decisão (seleção) aninhada. Antes de aprendermos a nossa última estrutura de decisão (seleção), "Caso Selecione", temos

que nos atentar para o fato de que a condição testada nas estruturas de decisão não precisa ter uma expressão somente.

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

CASO 4:

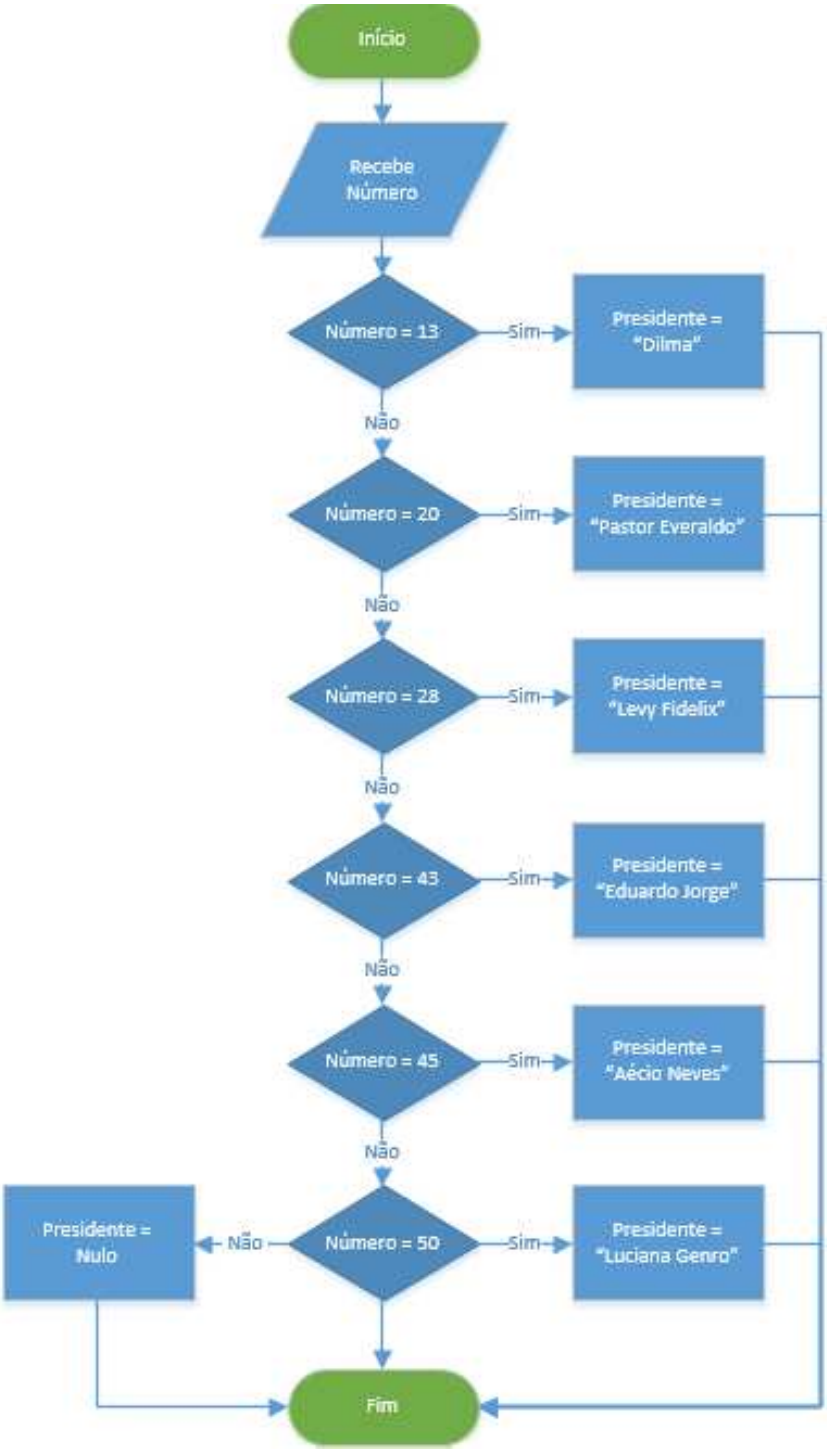
```
Selecione (Número)
      Caso 13: Presidente = "Dilma"
      Caso 20: Presidente = "Pastor Everaldo"
      Caso 28: Presidente = "Levy Fidelix"
      Caso 43: Presidente = "Eduardo Jorge"
      Caso 45: Presidente = "Aécio Neves"
      Caso 50: Presidente = "Luciana Genro"
      Caso Outro: Presidente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

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

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

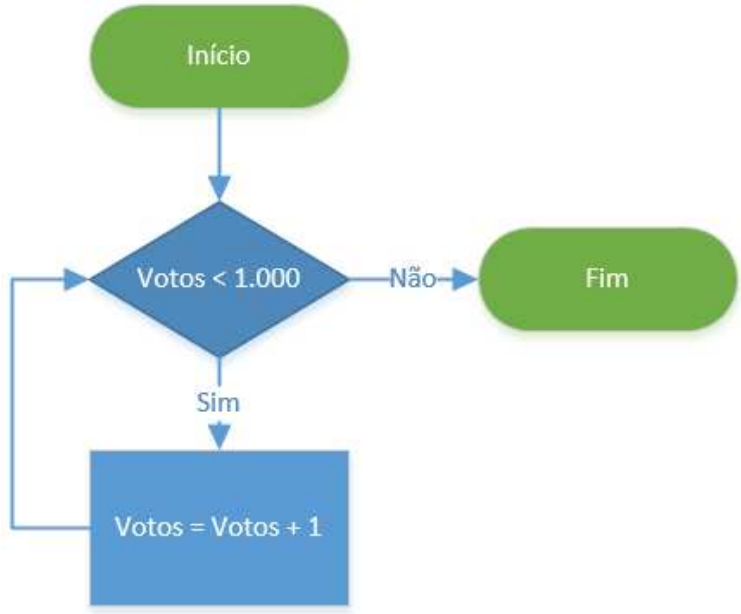
Agora veremos as famosas Estruturas de Repetição! Essas estruturas são utilizadas quando se deseja que um determinado conjunto de instruções ou comandos sejam executados por mais de uma vez. A quantidade pode ser definida, indefinida, ou enquanto um estado se mantenha ou seja alcançado. Parece confuso,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

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



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

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



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

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

```
Votos = 500

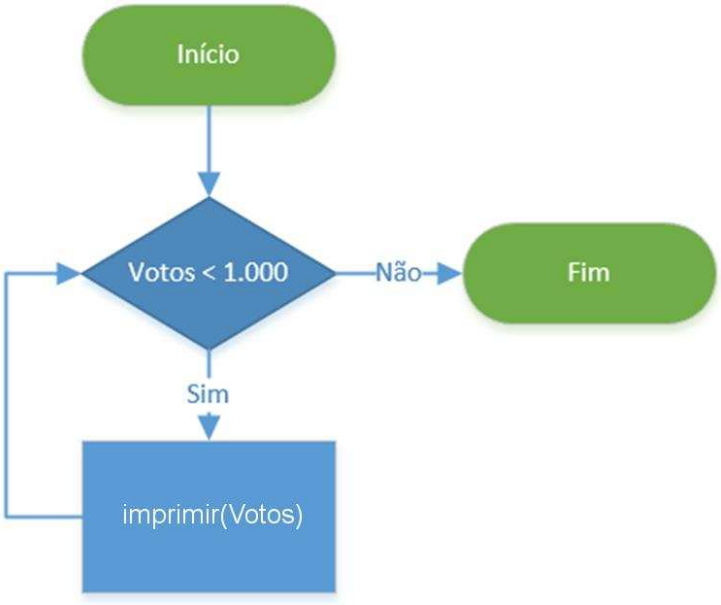
Enquanto (Votos < 1.000) Faça
      imprimir("Eu nunca mais vou sair daqui! Muahahahaha")
Fim-Enquanto
```

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

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

CASO 2: REPETIÇÃO COM VARIÁVEL DE CONTROLE

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



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

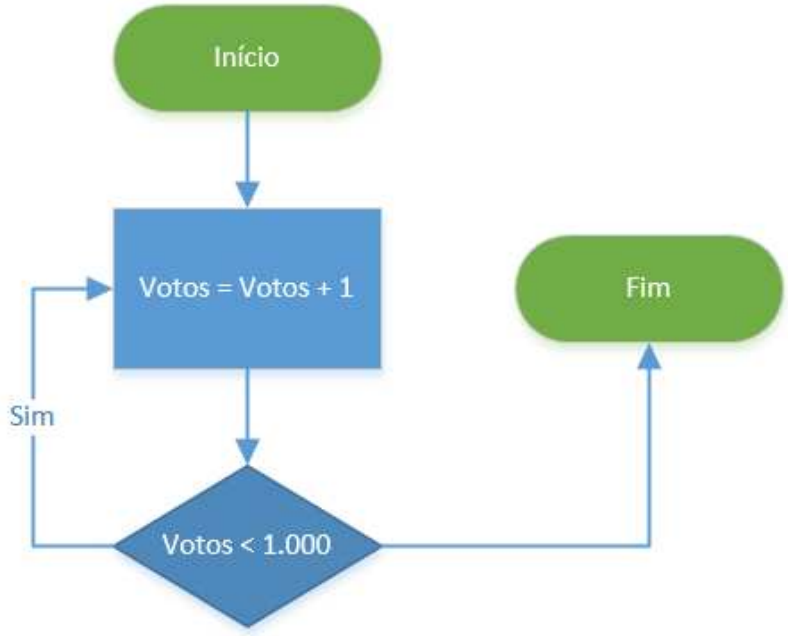
- Testa se Votos é igual a 1000;
- Se é igual, encerra a repetição;
- Se é menor, soma 1 à variável de controle "Votos";
- Mais uma repetição

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

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

```
Faça
```

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



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

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

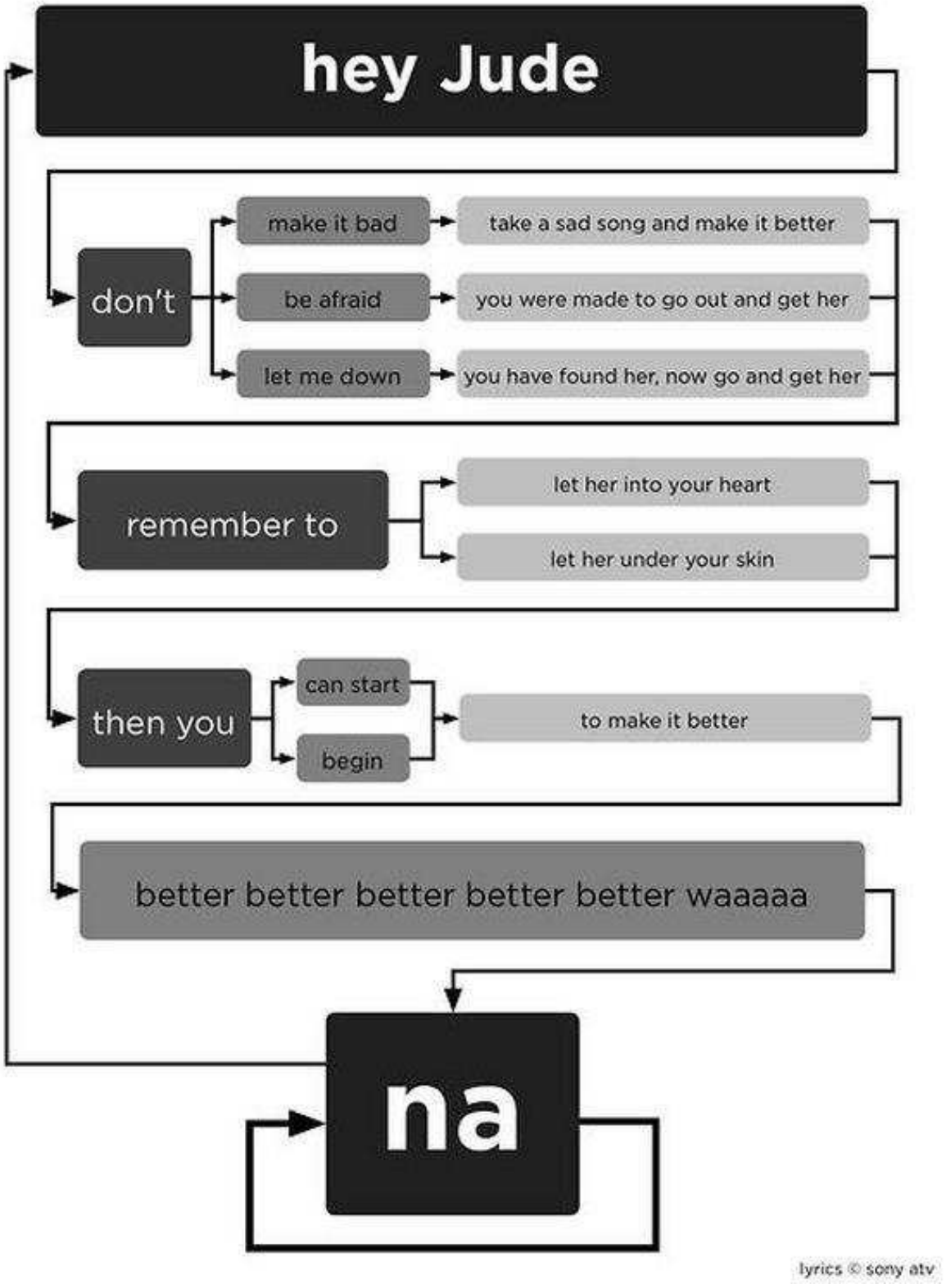
```
01 estrofe = 0
02
03 Enquanto (estrofe < 3) Repetir
04   estrofe == estrofe + 1
05   imprimir("Hey Jude, don't ")
06
07   Se (estrofe == 1) Então
08     imprimir("make it bad")
09     imprimir("take a sad song and make it better")
10   Senão Se (estrofe == 2) Então
11     imprimir("be afraid")
12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

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

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

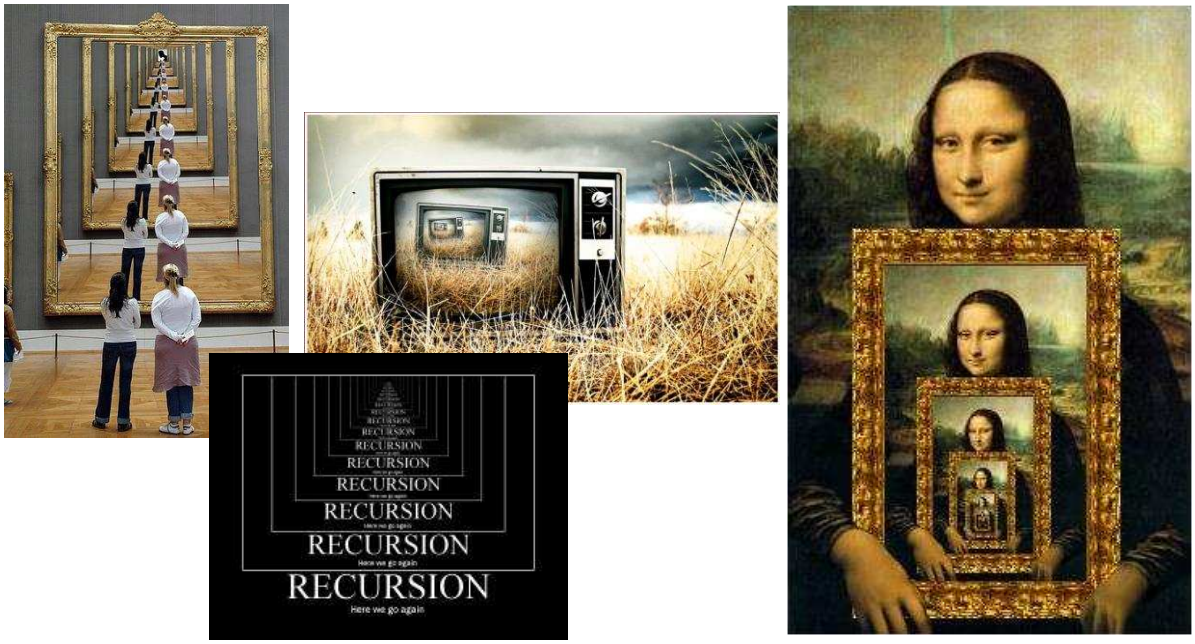
Após o primeiro "Se Então Senão", temos outro, que testa se estrofe é par ou ímpar (linha 28). Essa estrutura também é composta, pois possui um bloco "Senão". Por fim, temos uma estrutura de seleção (decisão) simples (linha 35), ou seja, somente com bloco "Se Então" e, dentro desse, encontramos uma estrutura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

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


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

RECURSIVIDADE

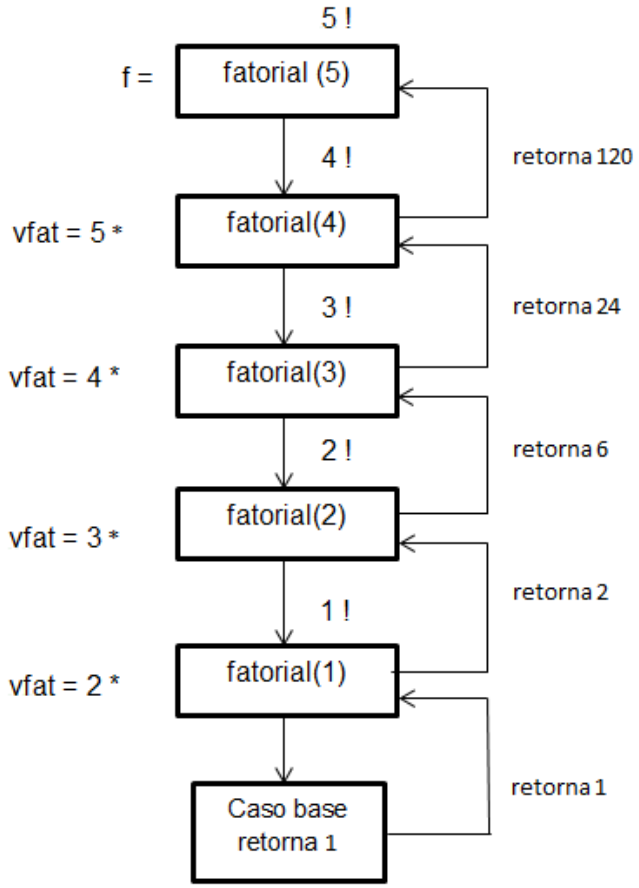
Por fim, vamos falar sobre recursividade! *O que é isso, professor?* Formalmente, é uma maneira de solucionar um problema por meio da instanciação de instâncias menores do mesmo problema. Grosso modo, a recursão é uma técnica em que uma função chama a si mesma. Eu gosto de pensar em recursividade por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



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

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

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



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

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



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

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

Comentários:

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

Gabarito: B

---

2. (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

---

3. (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

---

4. (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: C

---

5. (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.

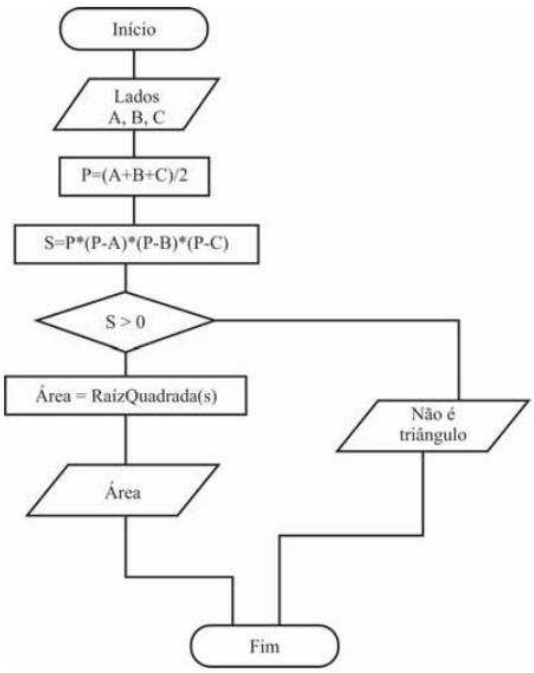
Comentários:

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

Gabarito: C

---

6. **(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: E

---

7. (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: E

---

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

Comentários:

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

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

Gabarito: C

---

9. (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: C

---

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

Comentários:

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

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

Gabarito: E

---

11. (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: C

---

12. **(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

13. (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

14. (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

15. (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

16. (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

17. (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

---

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

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

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

Comentários:

Antes do loop, nossas variáveis valem:

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

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

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

```
Quarta instrução:       Saída = 1;
Logo, temos que:        i = 2; f = 1; a = 1; b = 1;

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

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

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

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

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

Gabarito: E

---

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

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

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

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

Comentários:

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

Segundo o início do código temos:

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

1ª Chamada da função FF

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

FF retorna AUX * FF(AUX-1), ou seja, FF retorna 4 * FF(3)

2ª Chamada da função FF

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

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

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

3ª Chamada da função FF

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

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

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

Voltando à 2ª Chamada de FF

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

Voltando à 1ª chamada de FF

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

Gabarito: C

---

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

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

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

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

Comentários:

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

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

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

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

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

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

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

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

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

Gabarito: B

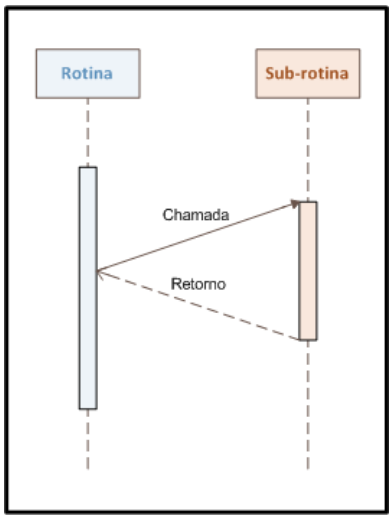
ACERTEI	ERREI

FUNÇÕES E PROCEDIMENTOS

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

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

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



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

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

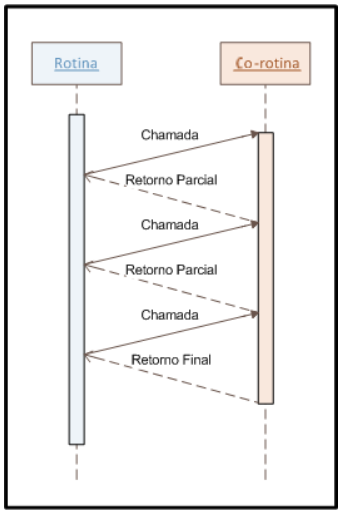
Algumas das vantagens na utilização de sub-rotinas durante a programação são: redução de código duplicado; possibilidade de reutilizar o mesmo código sem grandes alterações em outros programas; decomposição de problemas grandes em pequenas partes; melhorar a interpretação visual; esconder ou regular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

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

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

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

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



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

Infelizmente, eu não encontrei questões sobre esse tema.



Se alguém encontrar, eu posso gentilmente comentá-las.



PASSAGEM DE PARÂMETROS

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

```
#include <stdio.h>

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

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

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

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

    return();
}
```

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

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

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

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

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

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

	return();
}
```

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

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

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

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

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

Infelizmente, eu não encontrei questões sobre esse tema.



Se alguém encontrar, eu posso gentilmente comentá-las.



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

```
f(x)
x ← x + 1
devolva x
```

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

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

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

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

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

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

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

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

Comentários:

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

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

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

Gabarito: C

---

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

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

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

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

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

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

Comentários:

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

Gabarito: C

---

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

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

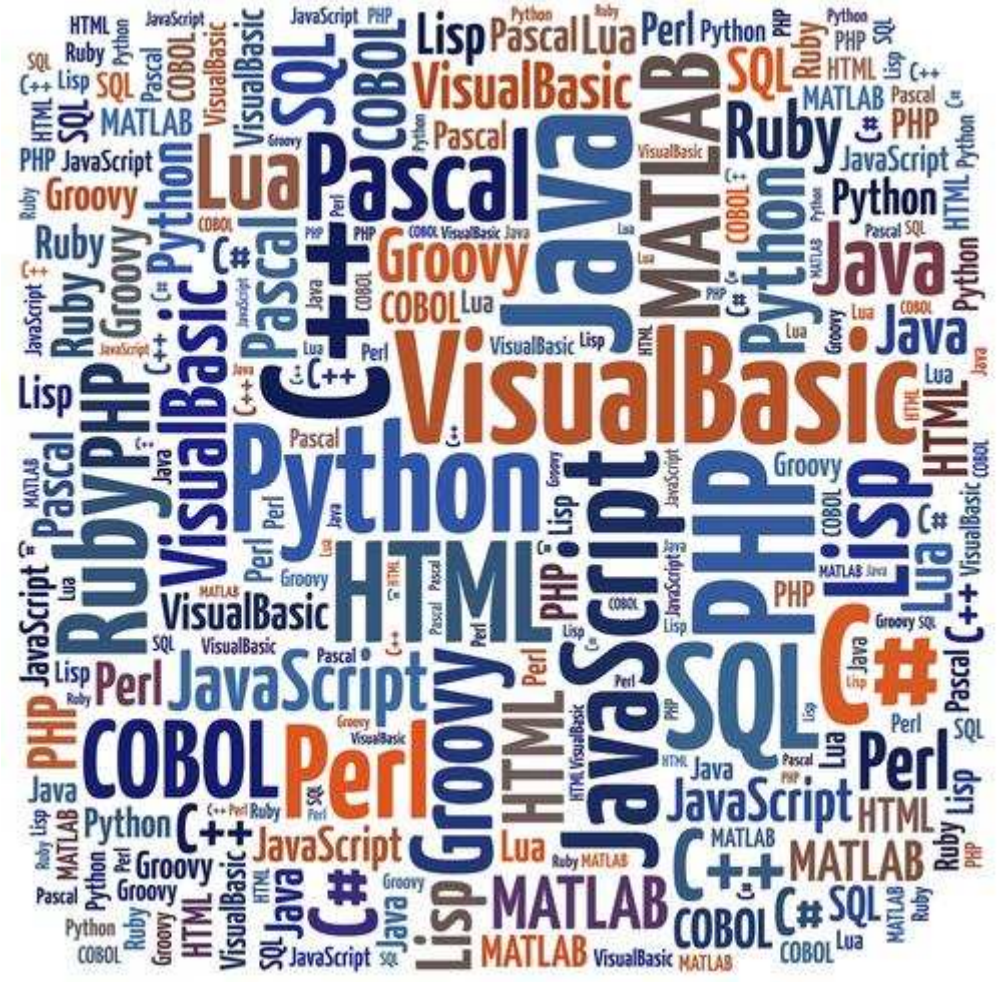
Comentários:

Galera, observem a pegadinha da questão: ele manda 6 (seis) como parâmetro e no retorno da rotina o valor é 6! (seis fatorial). Observe que o valor foi modificado, logo não pode ter sido uma passagem por valor - foi uma passagem por referência. Caso o retorno fosse 6 (seis), a passagem provavelmente seria por valor.

Gabarito: D

ACERTEI	ERREI

LINGUAGENS DE PROGRAMAÇÃO



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

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

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

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

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

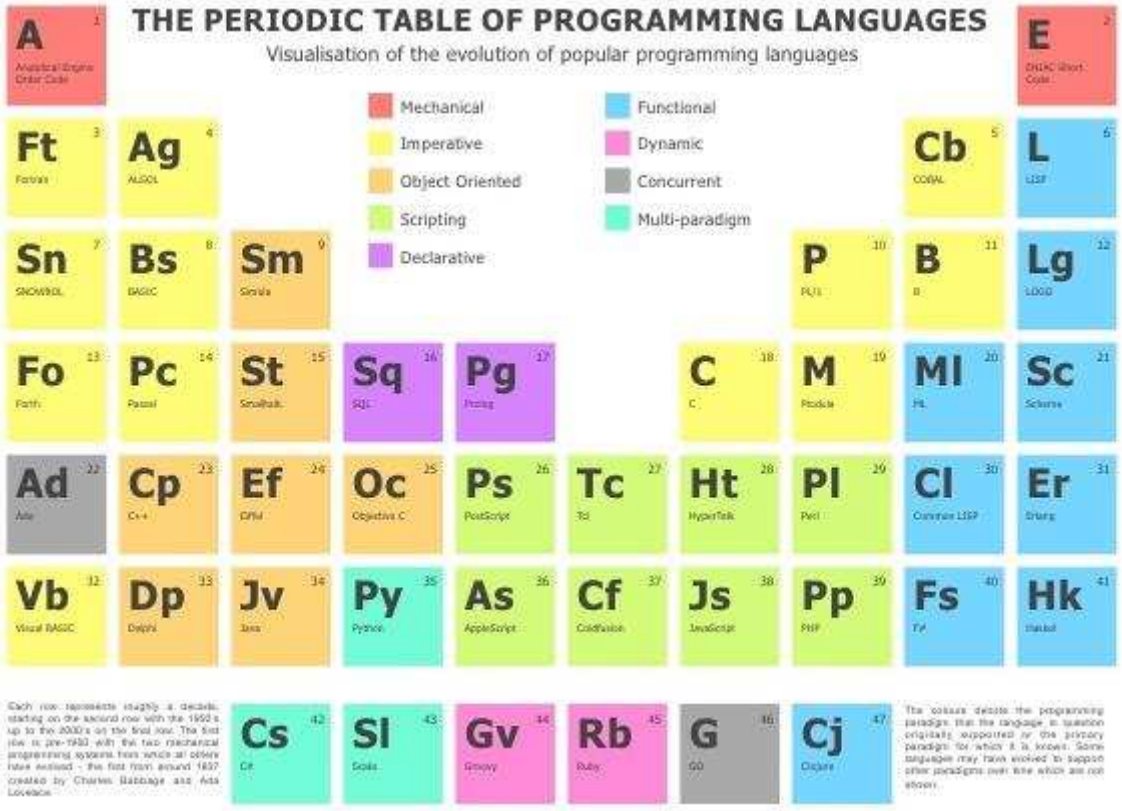
Linguagem de Alto Nível: linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana. Desse modo, as linguagens de alto nível não estão diretamente relacionadas à arquitetura do computador. O programador de uma linguagem de alto nível não precisa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

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



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

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

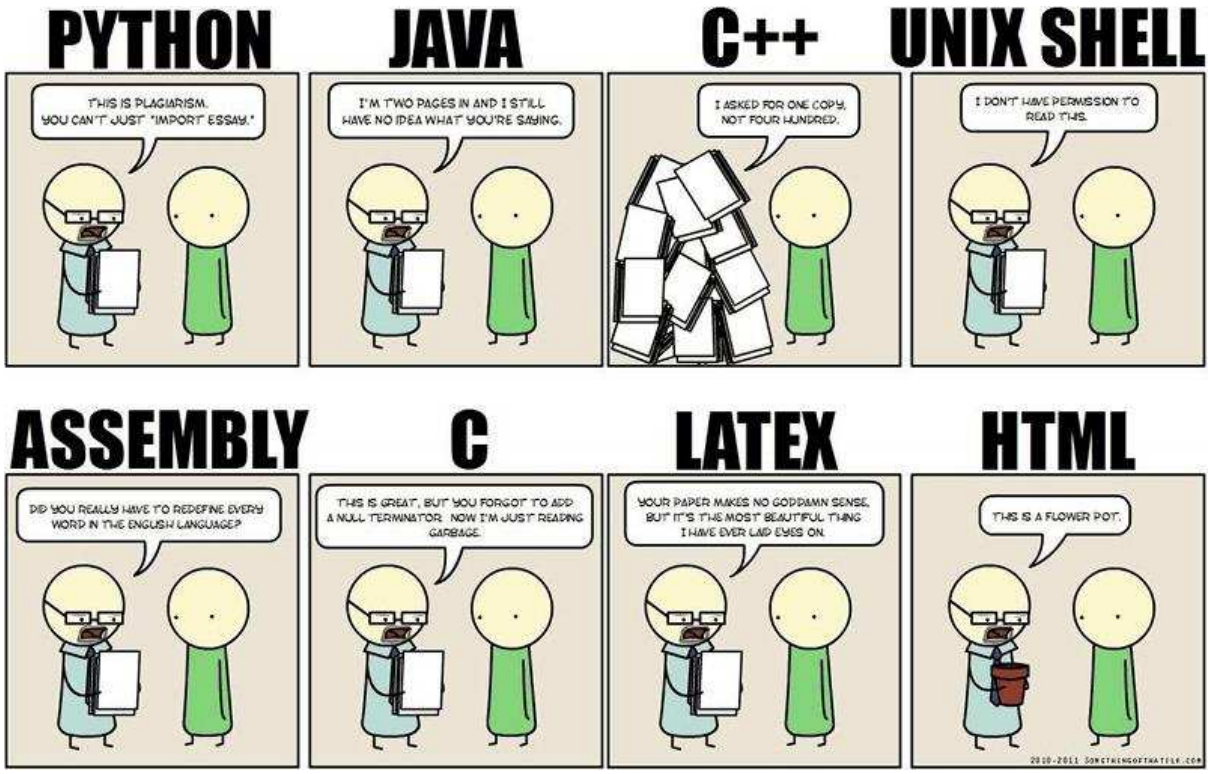


Galera, eu usei algumas dessas linguagens na faculdade! Por exemplo: usei FORTRAN na disciplina de Física Experimental e usei LISP na disciplina de Lógica Computacional. Quanto ao COBOL, eu nunca programei, mas tenho certeza que entre vocês tem alguém que trabalhou ou trabalha com algum sistema legado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

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

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

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



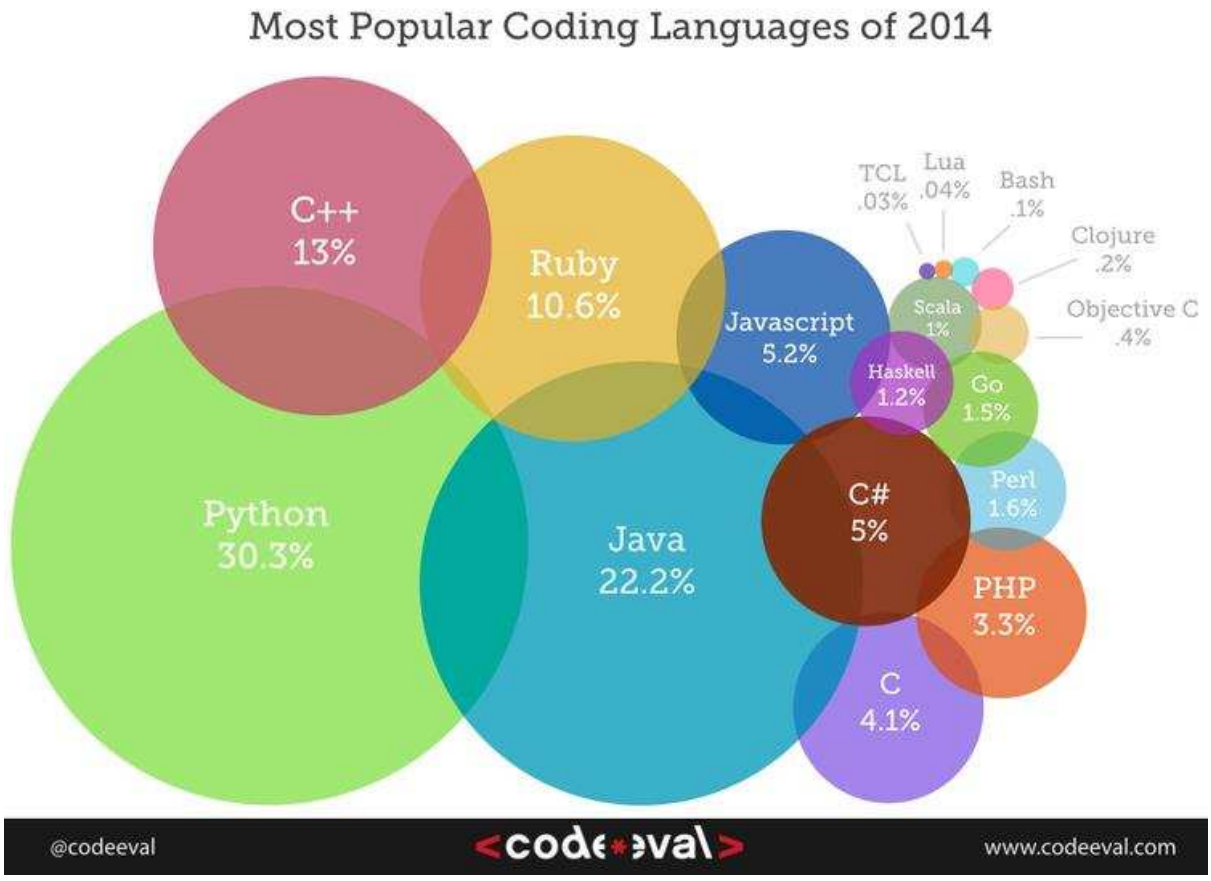
Todas estas linguagens foram criadas entre 1950 e 1960, e marcaram o início do desenvolvimento das linguagens de programação. Algumas destas linguagens, embora em versões mais recentes, são ainda utilizadas por vários programadores. O período compreendido entre o final dos anos 1960 à década de 1970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

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

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

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

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



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

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

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

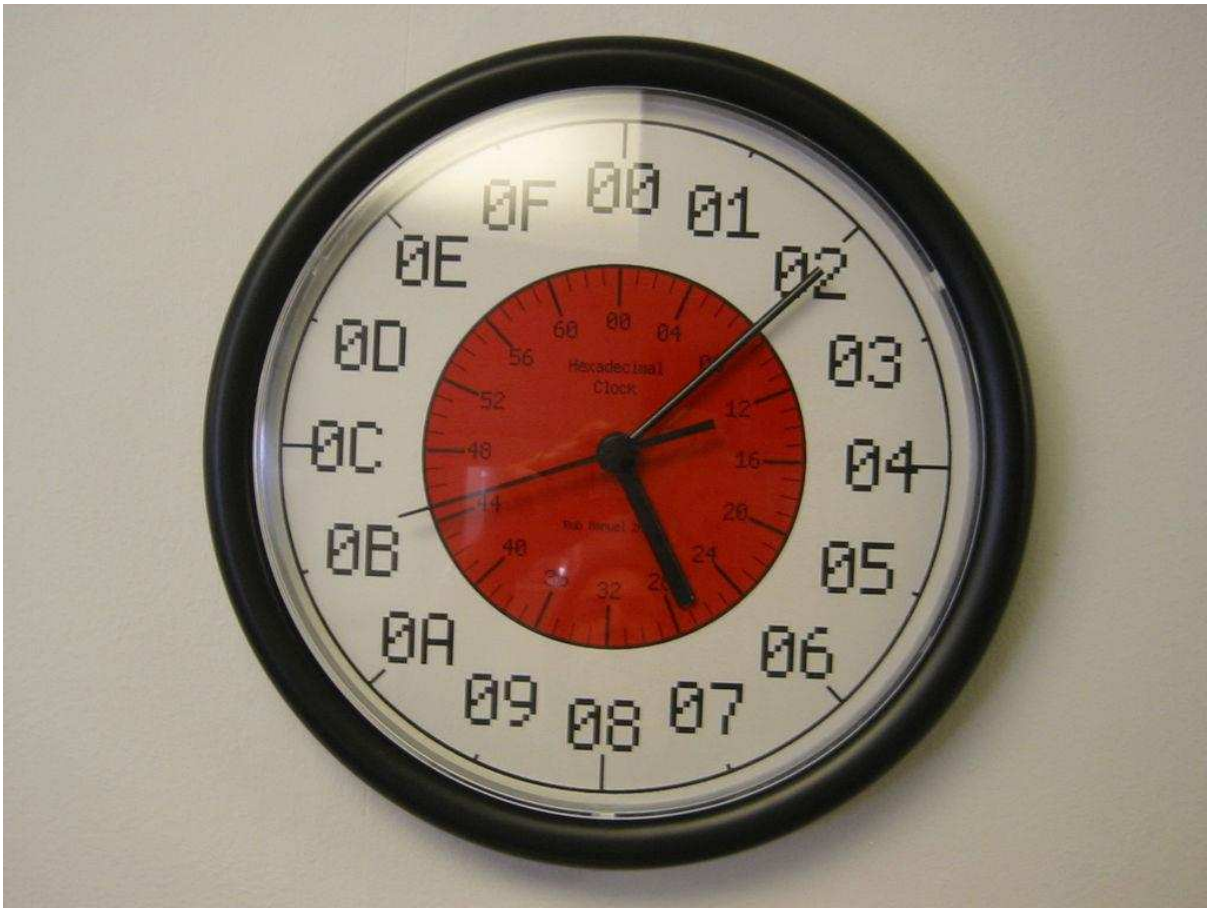
Em 1990, surgiu o Haskell, que era uma linguagem funcional de propósito geral. Em 1991, surgiu o Python, que é a queridinha de muita gente hoje em dia – utilizada no Google! Nesse mesmo ano, também veio o Java e revolucionou com o mundo todo da programação! Em 1993, veio o Ruby, que é outra queridinha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

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

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



SISTEMAS DE NUMERAÇÃO E ARITMÉTICA COMPUTACIONAL



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

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

Outros sistemas muito utilizados são o octal e o hexadecimal, também pelo fato de utilizarem bases que são potências de 2 ($2^3 = 8$ e $2^4 = 16$). Nós esperamos que, ao final dessa aula, cada aluno entenda a piadinha infame que diz que: "*Só existem 10*

tipos de pessoas no mundo: aquelas que entendem binário e aquelas que não". Se não entenderem, perguntem! :)

Os sistemas numéricos possuem características em comum na representação dos números. A partir da compreensão dessas regras simples, se torna bem mais fácil converter números de um sistema para outro. A partir de um número XYZ, ABC qualquer, em num sistema numérico com base n qualquer, temos que as regras a seguir se aplicarão sempre:

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

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

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

ou ainda, aplicando as potências:

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

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

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

O número 110,111 é, portanto: (4+2) = 6 na parte inteira, e (1/2+1/4+1/8) = 0,875 na parte fracionária. Assim, 110,111 em binário (base 2) é o mesmo que 6,875 em decimal (base 10). O sistema decimal utiliza 10 algarismos diferentes para representar os

números, de 0 a 9. É o sistema que melhor conhecemos, talvez por termos 10 dedos nas mãos.

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

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

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

Em hexadecimal, precisamos de letras para representar os números dez, onze, doze, treze, quatorze e quinze, ou seja, A, B, C, D, E e F respectivamente. Os números hexadecimais são precedidos de "0x" (um zero e um xis) para indicar que a representação é de um número na base 16, por exemplo, 0xFA. Saibam que conversões entre sistemas numéricos são bastante cobrados pelas bancas. *Ok?*

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

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

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

- Primeiramente buscamos a maior potência de 2 que seja menor que o número 486;

- Lembremos que as potências de dois são 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, etc;

- 256 é menor que 486, e esta é a maior potência de 2 menor que o número decomposto – o próximo seria 512 que seria maior que 486;

- *E o que é 486?* Galera, 486 = 256 + 230. Ou seja, 256 (maior potência de 2 menor que 486) + 230 = 486;

- Temos que continuar a decomposição, pois o que sobrou não é uma potência de 2. Então, vamos decompor 230.

- Busquemos a maior potência de 2 que seja menor que o número 230 (como fizemos acima);

- *Que número é esse?* É o 128, uma vez que o próximo número (256) é maior que o número 230.

- *E o que é 230?* É 128 + 102. E seguimos: 102 pode ser decomposto como 64 + 38; 38 pode ser decomposto como 32 + 6; 6 = 4 + 2; e fim!

- Logo, 486 pode ser decomposto como 486 = 256 + 128 + 64 + 32 + 4 + 2, ou ainda 486 = $1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^1 + 0 \times 2^0$

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

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

O número decimal (base 10) 486 convertido em binário (base 2), portanto, é 111100110. A conversão de octal (base 8) para binário (base 2) se dá a partir da tradução de cada algarismo em octal para um conjunto de três bits ($2^3 = 8$

possibilidades), começando sempre pela direita. Desse modo, o número 25173 em octal pode ser convertido em binário como se segue:

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

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

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

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

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

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

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

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

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

Binário	110	011	011	001
Octal	6	3	3	1

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

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

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

Binário	0011	0101	1101
Hexadecimal	3	5	D

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

Octal	6	3	4
Binário	110	011	100

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

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

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

Convertendo o número hexadecimal 1FA em binário, temos:

Hexadecimal	9	F	A
Binário	1001	1111	1010

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

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

O número 9FA em hexadecimal é igual a 100111111010 em binário que, por sua vez, é igual ao número em decimal: $2^{11} + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1 = 2048 + 256 + 128 + 64 + 32 + 16 + 8 + 2 = 2554$ Desse modo, chegamos ao fim da aula teórica em que cobrimos os principais sistemas de numeração e as suas conversões. Vamos praticar um pouco com exercícios *E a piada, todo mundo entendeu?* :)



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

a) 11,5.
b) 9,3.
c) 11,3.
d) 9,5.
e) 9,625.

Comentários:

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

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

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

Gabarito: E

---

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

   a) $8F.2_{16}$
   b) $DF.1_{16}$
   c) $CE.4_{16}$
   d) $CF.4_{16}$

Comentários:

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

Binário	1100	1111,	0100
Hexadecimal	C	F,	4

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

Gabarito: D

---

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

   a) 10011100 e 167

b) 10111010 e 167
c) 10011100 e 156
d) 10111010 e 156

Comentários:

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

Hexadecimal	9	C
Binário	1001	1100

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

Gabarito: C

---

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

Comentários:

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

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

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

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

Gabarito: E

---

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

a) 65,20.
b) 47,60.
c) 129,70.
d) 242,15.
e) 39,34.

Comentários:

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

Hexadecimal	2	F	3	C
Binário	0010	1111	0011	1100

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

Gabarito: B

---

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

a) 2F6 h
b) 3E7 h
c) 4D6 h
d) 5F7 h
e) 6B6 h

Comentários:

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

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

Binário	0011	1110	0111
Hexadecimal	3	E	7

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

Gabarito: B

---

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

a) F8
b) AF
c) FF
d) FA
e) FB

Comentários:

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

Gabarito: D

---

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

a) C3F
b) C37F0
c) C3F7
d) EF3C
e) FE3CA

Comentários:

O número binário 1100001111110111, dividido de quatro em quatro bits, sempre pela direita, resulta em: 1100 0011 1111 0111. Convertendo cada 4 bits num algarismo hexadecimal temos 1100 = C, 0011 = 3, 1111 = F e 0111 = 7, ou seja, o número em hexadecimal é C3F7.

Gabarito: C

---

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

a) 175, AE e 10101110
b) 175, EF e 10110000
c) 175, AF e 10110000
d) 191, EA e 10110000
e) 191, FA e 10101110

Comentários:

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

Gabarito: C

---

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

a) 11111010 e FA.
b) 11111010 e FE.
c) 11111010 e FC.
d) 11111110 e FE.
e) 11111110 e FA.

Comentários:

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

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

Lembrando que a conversão passando pelo binário é bem mais rápida e simples, pois não requer divisões que podem ser complexas na hora da prova. Esse método direto é somente para ilustrar que é possível converter decimal para qualquer base a partir do método da decomposição. Finalmente, convertendo 250 em decimal

para base 2 (binário) temos a decomposição: 250 = 128 + 64 + 32 + 16 + 8 + 2 = $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$. 250 em binário, portanto, é 11111010.

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

Gabarito: A

ACERTEI	ERREI



AUTÔMATOS DETERMINÍSTICOS E NÃO-DETERMINÍSTICOS

Bem, eu preciso desabafar uma coisa: eu tinha pavor dessa disciplina na faculdade – ela era famosa por ser muito difícil! E, realmente, se trata de um assunto bastante complexo, mas aquilo era faculdade e isso aqui é concurso! Então fiquem tranquilos! *Sabe quantas questões já caíram em concursos?* Apenas duas! Em toda a história dos concursos, só encontrei duas questões...

Logo, levem isso em consideração na hora de sopesar o que vocês têm que estudar mais e o que vocês têm que estudar menos. *Bacana?* Vamos lá... a palavra "*autômato*" é uma latinização da palavra grega *αὐτόματον*, autômato, significando "*agindo pela vontade própria*". É mais comumente descrito como máquinas que se movem sem a ajuda de eletricidade (Ex: Relógio de Parede Cuco).

Ok, mas nosso assunto aqui é TI! No nosso contexto, um autômato é definido como sendo um modelo matemático de uma máquina de estados finitos. *O que é uma máquina de estados finitos?* É um modelo matemático usado para representar programas de computadores, circuitos lógicos, etc. O conceito é concebido como uma máquina abstrata que deve estar em um de seus finitos estados.

A máquina está em apenas um estado por vez, este estado é chamado de estado atual. Um estado armazena informações sobre o passado, isto é, ele reflete as mudanças desde a entrada num estado, no início do sistema, até o momento presente. Uma transição indica uma mudança de estado e é descrita por uma condição que precisa ser realizada para que a transição ocorra.

Uma ação é a descrição de uma atividade que deve ser realizada num determinado momento. Máquinas de estado finito podem modelar um grande número de problemas, entre os quais a automação de design eletrônico, projeto de protocolo de comunicação, análise e outras aplicações de engenharia. Enfim, existem centenas, dezenas, milhões de exemplos por aí!

Até agora nós vimos que uma máquina de estados finitos é um modelo matemático usado para modelar problemas, projetar programas de computador e circuitos lógicos digitais, entre outros. Trata-se de uma máquina abstrata que possui um número finito de estados e diversas transições entre esses estados. A ideia é *quebrar* o comportamento de um objeto em estado ou "*pedaços*" facilmente gerenciáveis.

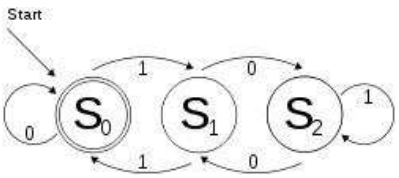
Por que utilizar essa abordagem? Ora, porque fica mais fácil de depurar (código modularizável); porque geralmente não necessita de muito processamento (if-else); e porque se trata de um modelo flexível (novos estados e transições podem ser facilmente adicionados). Um carro, por exemplo, pode ser modelado possuindo apenas dois estados: ligado ou desligado.

Se eu ligo a ignição, o carro é ligado; se eu desligo a ignição, há uma transição de estados, e o carro é desligado. *Quem aí já estudou UML?* Vocês devem se lembrar que lá existe um diagrama chamado Diagrama de Máquina de Estados ou Transição de Estados. Pois é, esse diagrama serve para modelar abstratamente os estados finitos de uma máquina ou de um sistema. Continuando...

Um autômato funciona como um reconhecedor de uma determinada linguagem e serve para modelar uma máquina ou, se quiserem, um computador simples. É usado, por exemplo, em editores de texto para reconhecer padrões. Um autômato pode ser representado de diversas formas, sendo uma das mais comuns a utilização de um grafo dirigido ou grafo orientado.

Pois bem, mas qual a diferença entre um autômato determinístico e um autômato não determinístico? Um autômato finito determinístico — também chamado máquina de estados finita determinística (AFD) — é uma Máquina de estados finita que aceita ou rejeita cadeias de símbolos gerando um único ramo de computação para cada cadeia de entrada.

Um Autômato Finito Determinístico (AFD) é definido por uma quíntupla M = (S , Q, d, q_0, F), em que:



- S: Alfabeto de símbolos finitos de entrada;
- Q: Conjunto finito de estados possíveis para M;
- d: Função transição ou função programa definida em Q x S → Q;
- q_0: Estado inicial de M, sendo $q_0 \in Q$;
- F: Conjunto de estados finais, tal que $F \subseteq Q$;

A imagem acima representa um autômato finito determinístico através de um Diagrama de Transição de Estados. Nesse autômato, há três estados: **S_0**, **S_1** e **S_2**. A entrada é constituída por uma sequência finita de caracteres 1's e 0's. Para cada

estado da máquina, existe um arco de transição levando a um outro estado para ambos caracteres do alfabeto de entrada.

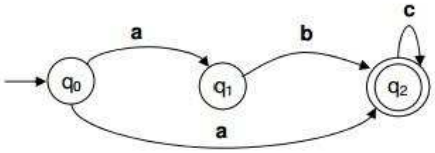
Isso significa que, em um dado estado, após a leitura de cada símbolo, a máquina determinística transita para um único estado referente à aresta associada ao símbolo. Por exemplo, esteja o autômato atualmente no estado S_0 e o símbolo de entrada para aquela instância seja um '1', então ele salta deterministicamente para o estado S1.

Todo Autômato Finito Determinístico (AFD) possui um estado inicial (denotado graficamente por uma seta de origem anônima) onde a sua computação começa e um conjunto de estados finais (denotados graficamente por um círculo de borda dupla) o qual indica a aceitação da cadeia de entrada. *E onde esses autômatos são utilizados, professor?*

AFDs são utilizados para modelar softwares que validam entradas de usuário tal como um e-mail em um servidor de correio eletrônico, reconhecem exatamente o conjunto de Linguagens Regulares que são, dentre outras coisas, úteis para a realização de Análise Léxica e reconhecimento de padrões. *Professor, por que ele se chama autômato finito? E por que autômato determinístico?*

Um autômato é dito finito, porque o conjunto de estados possíveis (Q) é finito – em nossa imagem, temos apenas três estados. Um autômato é determinístico quando dado o estado atual de M, ao ler um determinado símbolo de entrada, existe apenas um próximo estado possível. *Tudo entendido?* Ok, mas também existem Autômatos Finitos Não-Determinísticos!

Um AFN, similarmente a um AFD, lê uma cadeia de símbolos de entrada. Para cada símbolo da entrada há uma transição para um novo estado, até que todos os símbolos de entrada sejam lidos, porém existe pelo menos um estado tal que ao ler um mesmo símbolo há mais de uma possibilidade de estado destino. Assim, o próximo estado é um elemento do conjunto das partes dos estados.



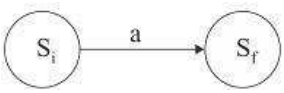
No autômato acima, existem duas transições de estado possíveis ao ler o símbolo "a", estando o autômato no estado q_0. Uma cadeia é aceita por um AFN se, testando-se todas as transições possíveis à medida que se lê a cadeia, o AFN para

em um estado final. Assim, o não-determinismo do próximo estado pode ser interpretado como um teste de todas as possibilidades.

Por outro lado, uma cadeia é rejeitada por um AFN se nenhum caminho de transições leva o autômato a um estado final após ler toda a cadeia. Bem, é isso...

CAIU na prova!
EXERCÍCIOS CESPE

1. (CESPE – 2013 – PF – Analista de Sistemas) Autômatos finitos são usualmente apresentados na forma de um grafo dirigido. A figura abaixo representa uma transição que pode ocorrer se o autômato estiver em um estado S_i e se o símbolo da string de entrada for a. Caso a entrada para o autômato seja a string prova, é correto afirmar que ocorrerá a transição de S_i para S_f.



Comentários:

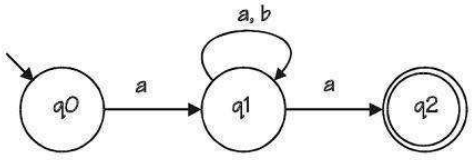
Um autômato é dito finito, porque o conjunto de estados possíveis (Q) é finito – em nossa imagem, temos apenas três estados. Um autômato é determinístico quando dado o estado atual de M, ao ler um determinado símbolo de entrada, existe apenas um próximo estado possível. Tudo entendido? Ok, mas também existem Autômatos Finitos Não-Determinísticos!

Bem, essa questão é extremamente mal formulada! *Cadê o estado inicial?* Já seria motivo para anulação! *Cadê o(s) estado(s) final?* Outro motivo para anulação! *Cadê o alfabeto de símbolos que são aceitos como entrada?* Mais um motivo para anulação! Aparentemente, só há uma transição: $S_i \rightarrow S_f$ quando se lê o caractere *"a"*. *Cadê as transições para "p", "r", "o", "v"?*

Ignorando os outros motivos de anulação, como não há transições mapeadas para esses outros caracteres, logo não há que se falar que ocorrerá transição entre os dois estados, na medida em que a cadeia já será rejeitada no primeiro caractere por falta de transição. Enfim, questão muito mal feita! E isso é, de certa forma, surpreendente, visto que estamos falando da Polícia Federal!

Gabarito: C

2. (CESPE – 2010 – INMETRO – Analista de Sistemas) Considerando a figura acima, que representa um autômato finito não determinístico, e a cadeia de entrada babaa, assinale a opção correta.



J. L. Rangel e L. C. Guedes. **Linguagens formais**.

a) O autômato lê o primeiro valor da cadeia de entrada e q0 o transforma em a.

b) Ao aceitar a cadeia de entrada, o autômato é levado, sucessivamente, a um estado final q2.

c) O autômato pode aceitar a cadeia de entrada oferecida e adivinhar que a cadeia acabou.

d) O autômato pode aceitar a cadeia de entrada oferecida, desde que q0 seja equivalente a q1.

e) O autômato não pode aceitar a cadeia de entrada oferecida.

Comentários:

Perceba que a cadeia de entrada é "*babaa*". Ora, já no primeiro caractere, não é possível tomar nenhuma transição, logo a entrada está rejeitada. Explicando melhor: o estado inicial q_0 aceita apenas a entrada "a", logo esse autômato não aceita nenhuma cadeia que se inicia com caractere diferente de "a", portanto o autômato não pode aceitar a cadeia de entrada oferecida. *Simples, não?*

Gabarito: E

ACERTEI	ERREI

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
LÓGICA DE PROGRAMAÇÃO

1. (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

2. (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.

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

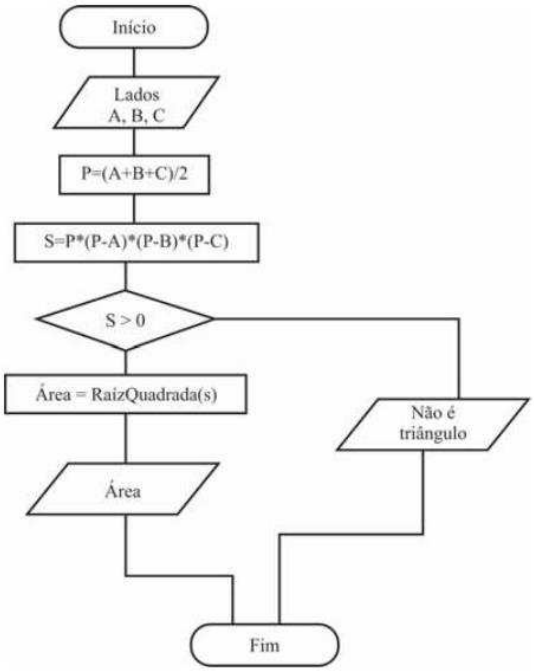
   a) apresenta outra função como resultado.
   b) aponta para um objeto.
   c) aponta para uma variável.
   d) chama uma outra função.
   e) pode chamar a si mesma.

4. (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.

5. (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.

6. (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.



7. (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.

8. (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.

9. (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.

10. (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.

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

12. (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

13. (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:

c) Para n de 0 até 6 faça imprimir(n);
d) Para n de 0 até 7 faça imprimir(n);

14. (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.

15. (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).

16. **(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
...
```

17. (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 + 1;
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.

18. (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

19. (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.

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

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

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

a) 12345
b) 1245

c) 3
d) 0124
e) 1234

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
PASSAGENS DE PARÂMETROS

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

```
f(x)
x ← x + 1
devolva x
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
SISTEMAS DE NUMERAÇÃO E ARITMÉTICA COMPUTACIONAL

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

   a) 11,5.
   b) 9,3.
   c) 11,3.
   d) 9,5.
   e) 9,625.

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

   a) $8F.2_{16}$
   b) $DF.1_{16}$
   c) $CE.4_{16}$
   d) $CF.4_{16}$

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

   a) 10011100 e 167
   b) 10111010 e 167
   c) 10011100 e 156
   d) 10111010 e 156

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

5. (FCC - 2015 – DPE-RR – Engenheiro Eletrônico ou Mecatrônico) Um comerciante de peças de bicicleta utiliza-se da numeração hexadecimal como código para representar o preço de custo dos seus artigos, de maneira que, quando um

cliente pede um desconto, ele sabe até que valor pode chegar. Para tanto, representou a parte inteira por um número hexadecimal, e os centésimos de real por outro número sempre com dois algarismos. Assim, o preço de custo de um produto que apresenta o código 2F3C é, em reais,

a) 65,20.
b) 47,60.
c) 129,70.
d) 242,15.
e) 39,34.

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

a) 2F6 h
b) 3E7 h
c) 4D6 h
d) 5F7 h
e) 6B6 h

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

a) F8
b) AF
c) FF
d) FA
e) FB

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

a) C3F
b) C37F0
c) C3F7
d) EF3C
e) FE3CA

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

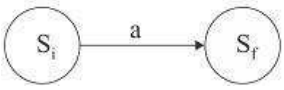
a) 175, AE e 10101110
b) 175, EF e 10110000
c) 175, AF e 10110000
d) 191, EA e 10110000
e) 191, FA e 10101110

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

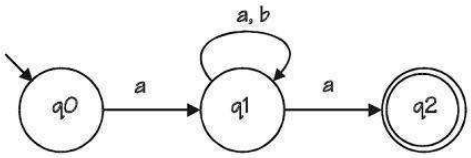
a) 11111010 e FA.
b) 11111010 e FE.
c) 11111010 e FC.
d) 11111110 e FE.
e) 11111110 e FA.

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
AUTÔMATOS FINITOS

1. (CESPE – 2013 – PF – Analista de Sistemas) Autômatos finitos são usualmente apresentados na forma de um grafo dirigido. A figura abaixo representa uma transição que pode ocorrer se o autômato estiver em um estado S_i e se o símbolo da string de entrada for a. Caso a entrada para o autômato seja a string prova, é correto afirmar que ocorrerá a transição de S_i para S_f.



2. (CESPE – 2010 – INMETRO – Analista de Sistemas) Considerando a figura acima, que representa um autômato finito não determinístico, e a cadeia de entrada babaa, assinale a opção correta.



J. L. Rangel e L. C. Guedes. **Linguagens formais**.

a) O autômato lê o primeiro valor da cadeia de entrada e q0 o transforma em a.

b) Ao aceitar a cadeia de entrada, o autômato é levado, sucessivamente, a um estado final q2.

c) O autômato pode aceitar a cadeia de entrada oferecida e adivinhar que a cadeia acabou.

d) O autômato pode aceitar a cadeia de entrada oferecida, desde que q0 seja equivalente a q1.

e) O autômato não pode aceitar a cadeia de entrada oferecida.

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
LÓGICA DE PROGRAMAÇÃO

1	2	3	4	5	6	7	8	9	10
B	A	E	C	C	E	E	C	C	E
11	12	13	14	15	16	17	18	19	20
C	B	A	D	C	E	D	E	C	B

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
PASSAGEM DE PARÂMETROS

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

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
SISTEMAS DE NUMERAÇÃO E ARITMÉTICA COMPUTACIONAL

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

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
AUTÔMATOS FINITOS

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