Estratégia CONCURSOS

Aul

Tecnologia da Informação p/ Perito Polícia Federal (Área 03) - Parte II

Professor: Diego Carvalho, Equipe Informática e TI

AULA 00

4 Linguagens de programação. 4.1 Noções de linguagens procedurais: tipos de dados elementares e estruturados, funções e procedimentos. 4.3 Estruturas de controle de fluxo de execução. 4.4 Montadores, compiladores, ligadores e interpretadores. 5 Estruturas de dados e algoritmos. 5.1 Estruturas de dados: listas, filas, pilhas e árvores. 5.2 Métodos de acesso, busca, inserção e ordenação em estruturas de dados. 4.2 Noções de linguagens de programação orientadas a objetos: objetos, classes, herança, polimorfismo, sobrecarga de métodos. 4.5 Desenvolvimento web: Servlets, JSP, Ajax, PHP, ASP. 13.4 Análise de pontos de função. 13.5 Atos normativos do MPOG/SLTI: Instrução Normativa nº 2/2008 (alterada pela Instrução Normativa nº 3/2009); Instrução Normativa nº 4/2010. Engenharia Reversa. Técnicas e Ferramentas de Descompilação. Debuggers. Ofuscação de Código, Autômatos determinísticos e não- determinísticos, Desenvolvimento Seguro de Aplicações, Sistemas Operacionais Móveis. 7.5 Desenvolvimento seguro de aplicações: SDL, CLASP.

E aí, querem mais teoria? Mais exercícios? Tem muito mais! Essa é apenas a aula demonstrativa para que vocês conheçam o método e a escrita! Espero que vocês venham comigo... grande abraço ;)

A APRESENTAÇÃO...

Olá, pessoal. Sejam bem-vindos ao curso pré-edital para Perito Criminal da Polícia Federal! Esse é um dos concursos mais difíceis na área de tecnologia da informação. E nesses tempos de ajuste fiscal, cada concurso tem que ser levado muito a sério, com sangue nos olhos para conseguir sua tão desejada vaga. Corram, pessoal! Venham comigo ;)

DÚVIDAS DOS ALUNOS

TOP 5

1. Peço encarecidamente que leiam as instruções dessa primeira aula. Eu sei que é chato, mas assim nós alinhamos nossas expectativas a respeito do curso.

2. Essa é a Aula Demonstrativa (está disponível para todos na internet) – o restante do conteúdo estará disponível na Aula 01 (apenas para aqueles que adquirirem o curso).

3. Esse curso não possui vídeo-aulas! Estamos trabalhando para disponibilizá-las em breve, nesse primeiro semestre – talvez ainda não seja possível disponibilizá-las para esse curso.

4. Esse curso contempla somente aquilo que está em seu cronograma. Ele não contempla todo edital de tecnologia da informação, nem outras disciplinas, nem discursivas, estudos de caso, etc.

5. Existem questões de Múltipla Escolha (A, B, C, D, E) e existem questões de Certo/Errado (C, E). Quando não há itens para escolha na questão, é porque a questão é da Modalidade Certo/Errado.

O PROFESSOR...

Uma breve apresentação: meu nome é Diego Carvalho, bacharel em Ciência da Computação pela Universidade de Brasília, pós-graduado em Gestão de Tecnologia da Informação na Administração Pública e Analista de Finanças e Controle da Secretaria do Tesouro Nacional. Já passei por esses perrengues de concurseiro e sei de duas coisas: a estrada é difícil, mas o prêmio compensa! E muito!

www.facebook.com/professordiegocarvalho

ÁREA	ÓRGÃOS PARA OS QUAIS JÁ MINISTREI CURSOS			
Agência	ANCINE	ANTAQ	ANATEL	
Jurídica	TRT/1	TRT/2	TRT/3	TRT/4
	TJ/BA	CNMP	MP/PB	TRT/15
Legislativa	CÂMARA DOS DEPUTADOS			
Auditoria	TCE/RS	TCE/SP	TCE/CE	TCM/GO
	TCU	TCM/SP		
Fiscal	ISS/SP	ISS/BA		
Outros	CEF	DATAPREV	DEPEN	INMETRO

Neste curso, contaremos também com a Professora Mayara Rosa! Ela também é bacharel em Ciência da Computação na Universidade de Brasília (UnB), pós-graduada em Engenharia de Sistemas e recentemente tomou posse como Auditora Federal de Controle Externo no Tribunal de Contas da União (TCU). Ela é a nova professora do Estratégia Concursos! ;)

Galera, lá no site, nós – professores – temos algumas métricas para medir se o nosso desempenho nos cursos está bacana! Os alunos podem avaliar com notas e, inclusive, escrever anonimamente o que acharam do professor e do curso. Apresento abaixo o resultado de alguns cursos ministrados recentemente. Portanto, confiem em mim... vocês vão aprender muito com esse curso!

Comentário sobre o curso
Curso bom.
Muito bom o curso, abrange os tópicos cobrados conforme o edital.
Atende ao conteúdo pertinente ao edital!
Pontos positivos: -Marcações coloridas no texto para destacar as informações -Boa cobertura dos pontos do edital -Imagens ilustrativas -Ilustrações feitas pelo próprio professor (eu acho) que ficaram bem legais -Muitas questões resolvidas -Qualidade do conteúdo foi muito boa Pontos negativos: -Muita informalidade, pode ser só uma frescura minha, mas o "Galera" é usado em excesso, acabou prejudicando um pouco a seriedade da aula. -Muitas questões do CESPE, apesar de a banca ser FCC -Comentários muito sucintos em algumas questões: "Perfeito, é isso mesmo." Considerações adicionais: Não sei se o professor tinha experiência prévia no ramo, mas pelo menos para mim ficou clara a melhora das aulas ao longo do curso. Ao final realmente estava TOP, digno do Estratégia Concursos. No mais, eu gostaria de agradecer ao professor Diego por ter se disposto a dar essas aulas. Foram só 500 inscritos no concurso, é bastante provável que poucas pessoas tenham feito o curso, então a rentabilidade deve ter sido baixa. Tenho certeza que muito cursinho por aí cancelaria, mas mesmo assim ele foi adiante. Valeu! Essas aulas serão úteis não só para esse concurso. Vem aí o TCU TI e praticamente tudo do seu curso deve cair, então foi um ótimo investimento. Com certeza estudarei todo o material várias vezes!

Comentário sobre o curso
Fui colega do Diego no CF/STN mas durante o período do CF não conversei muito com ele. Soube que era professor de TI também além de concurseiro. Não conhecia o trabalho dele como professor mas me surpreendi positivamente, muito mesmo. A aula é elaborada com muito cuidado, criatividade e é uma aula leve de se estudar. Apesar de eu estar afastado da área de Engenharia de Software e Desenvolvimento já há algum tempo foi muito tranquilo acompanhar a exposição dele nas aulas. Parabéns Diego! Muito bom trabalho!!
ótimo curso
No começo estava meio cético com relação ao curso, mas quanto mais ia me aprofundando nas apostilas percebi que o professor fez um bom trabalho. No mais, gostaria de ter mais questões comentadas, e com comentários um pouco mais completos.
Otima didática com conteúdo e exercicios de diversas bancas logo em seguida. Quando tiver as videos aulas será melhor ainda. Continue assim ...
O curso me atendeu muito bem. Aspectos que gostei bastante: - Linguagem prática e objetiva - Sem enrolações - Sem copy/paste desnecessários no comentários de questões repetitivas (que cobram o mesmo conhecimento), explicou uma vez, ok, as outras basta pincelar de forma rápida. - Conteúdo bem direcionado. Não foi superficial mas também não desceu ao nível NASA. - Cumpriu o cronograma Como fatores negativos, apenas com relação aos pequenos erros encontrados vez ou outra nos comentários das questões ou gabaritos trocados. Entretanto, conforme já explicado pelo próprio professor, o curso está amadurecendo. Estou satisfeito. Tenho pretenção de adquirir novamente esse curso daqui a algum tempo e conferir a evolução, mas, cá entre nós: Quero mesmo é passar de uma vez agora e não precisar disso nunca mais :D Sucesso Diego. Att, Benjamin Pinto
Gostei muito, doo curso. Apenas senti falta dos vídeos para diversificar um pouco, não ficar tão teórico.

Comentário sobre o curso
Excelente professor na área de TI. Um dos melhores que já vi em todos os cursos que já realizei, seja presencial ou a distância. Realizaria qualquer outro curso ministrado pelo professor.
Professor muito bom! Aulas totalmente direcionadas no edital, clareza para escrever, boa seleção de exercícios e bons comentários. Passa segurança para o aluno e tem domínio do conteúdo.

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.

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
Existem algumas definições que só foram explicadas nos comentários dos exercícios, poderiam ser dadas antes. No geral o material é muito bom. Só faltou vídeos pra ilustrar melhor alguns conceitos de arquitetura.
O material do curso é ótimo, mas o que faltou foi um resumo no final de cada aula com tudo que foi falado. De resto, eu gostei, foi meu primeiro curso no Estratégia e pretendo voltar a fazer outros.
Excelente didática do Professor. Certamente se ele lançar outro curos vou adquirir. Como sugestão o Estratégia devia fazer mais parcerias com professores de TI. Essa área é carente em cursos bons iguais a este. A procura é grande.
Muito bom o curso e principalmente o comprometimento do professor para com os alunos.

Comentário sobre o curso
Muito bom o curso, contudo, senti falta das vídeo aulas complementares.
Gostei muito da forma com o professor respondeu as perguntas, da maioria do conetúdo apresentado. A única coisa que acredito tornar o curso perfeito são os vídeos, senti falta. O vídeo auxilia em um primeiro entendimento, para depois seguir lendo a apostila.
Estou iniciando no estudo de concursos para TI e gostei muito do curso. Não achei muito caro, isso me permitiu comprar outro curso no site. Estão de parabéns!
Ótimo curso, tudo bem explicado
Curso de excelente qualidade. Entretanto extremamente insuficiente para o concruso, mesmo com a Parte I.

O CONCURSO...

POLÍCIA FEDERAL – PERITO CRIMINAL ESPECIALIDADE: TECNOLOGIA DA INFORMAÇÃO	
	
REMUNERAÇÃO	VAGAS
R$16.830,85	-
	
EDITAL/AUTORIZAÇÃO:	
-	

O CURSO...

Antes de começar o curso, vamos alinhar algumas 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.

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?* ;)

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.

Além disso, 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?* Veja abaixo:

6º Lugar – ISS/Salvador
https://www.youtube.com/watch?v=b1w4H3l6mC4#t=1678

1º Lugar – TRT/RJ
https://www.facebook.com/video.php?v=790616534367672

2º Lugar – ISS/Salvador
https://www.youtube.com/watch?v=vmUInIJ-aqQ

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

O CRONOGRAMA...

Aula	Data	Tópicos do Edital
00	02/09	Aula Demonstrativa
01	04/09	4 Linguagens de programação. 4.1 Noções de linguagens procedurais: tipos de dados elementares e estruturados, funções e procedimentos. 4.3 Estruturas de controle de fluxo de execução. 4.4 Montadores, compiladores, ligadores e interpretadores.
02	06/09	5 Estruturas de dados e algoritmos. 5.1 Estruturas de dados: listas, filas, pilhas e árvores. 5.2 Métodos de acesso, busca, inserção e ordenação em estruturas de dados.
03	08/09	4.2 Noções de linguagens de programação orientadas a objetos: objetos, classes, herança, polimorfismo, sobrecarga de métodos.
04	10/09	4.5 Desenvolvimento web: Servlets, JSP, Ajax.
05	12/09	PHP e ASP.
06	14/09	13.4 Análise de pontos de função.
07	16/09	13.5 Atos normativos do MPOG/SLTI: Instrução Normativa nº 2/2008 (alterada pela Instrução Normativa nº 3/2009); Instrução Normativa nº 4/2010.
08	18/09	IN 04/2014 (Seleção do Fornecedor e Gestão Contratual)
09	20/09	Instrução Normativa nº 2/2008 (alterada pela Instrução Normativa nº 3/2009)
10	22/09	Engenharia Reversa. Técnicas e Ferramentas de Descompilação. Debuggers. Ofuscação de Código, Autômatos determinísticos e não-determinísticos, Desenvolvimento Seguro de Aplicações, Sistemas Operacionais Móveis. 7.5 Desenvolvimento seguro de aplicações: SDL, CLASP.

AS AULAS E AS DICAS...

1 – 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).	**2 – 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.
3 – 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.	**4 – 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.
5 – 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. Portanto, caso virem frases ou palavras em itálico, ou é uma palavra estrangeira ou é a simulação de uma conversa com vocês. *Pode dar um exemplo, professor?* Acabei de dar! :-)	**6 – 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.
7 – 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.	**8 – 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.
9 – 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.	**10 – 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.

Bem, pessoal! É isso... sejam bem-vindos! Espero que vocês curtam e tenham uma leitura leve e despojada da aula, mas com muito foco, atenção e dedicação. Qualquer dúvida, podem entrar em contato comigo – ficarei feliz em ajudá-los. Bons estudos, estou torcendo por vocês! Fiquem agora com algumas mensagens de incentivo para animá-los ;-)

R$ 16.830,85

R$ 16.830,85

R$ 16.830,85

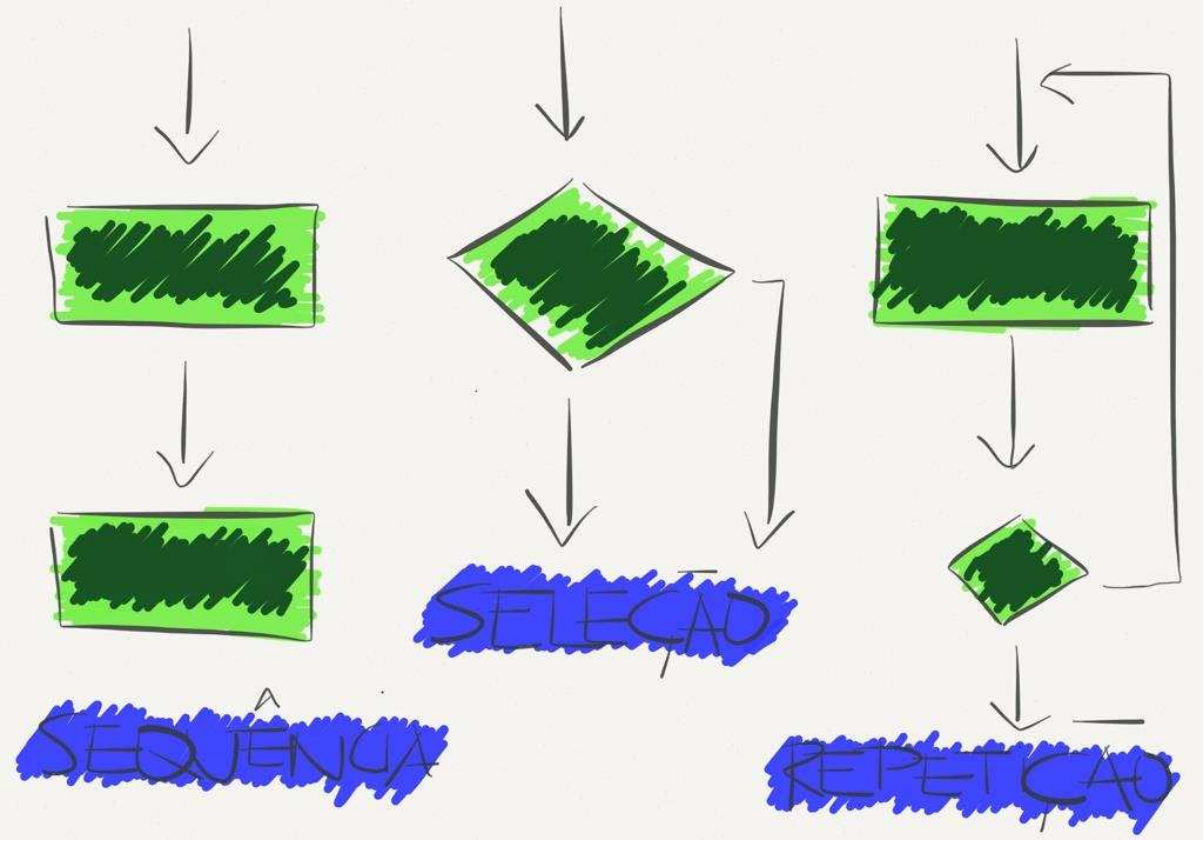
R$ 16.830,85

R$ 16.830,85

PARADIGMA ESTRUTURADO

O Paradigma Estruturado tem o objetivo de melhorar a clareza, qualidade e tempo de desenvolvimento de um programa de computador por meio da utilização extensa de sub-rotinas, estruturas de bloco e loops, em contraste à utilização de desvios incondicionais (famosos saltos ou *goto*). *Por que?* Porque eles dificultavam imensamente a leitura, compreensão e depuração do código.

Quem já programou em Assembly sabe que os saltos são bastante problemáticos e complexos. Portanto, o Paradigma Estruturado busca organizar o fluxo de controle de execução dos programas de maneira lógica, por meio da modularização do código em blocos aninhados de comando, oferecendo ao programador mais controle sobre o fluxo.



Agora vamos mais a fundo! As linguagens estruturadas reduzem qualquer programa a três estruturas principais: sequência, seleção e repetição. Sequencial, porque possui uma ordem de execução; seletiva, porque seleciona um trecho de código dentre diversas opções de fluxo; e repetitiva, porque realiza a iteração de diversos trechos de código.

```
/* Representação de um algoritmo genérico */

variavel 1;
variavel 2;
variavel 3;

Inicio

{
    comando 1;
    comando 2;
    comando 3;
    comando 4;
    comando 5;
}

Fim
```

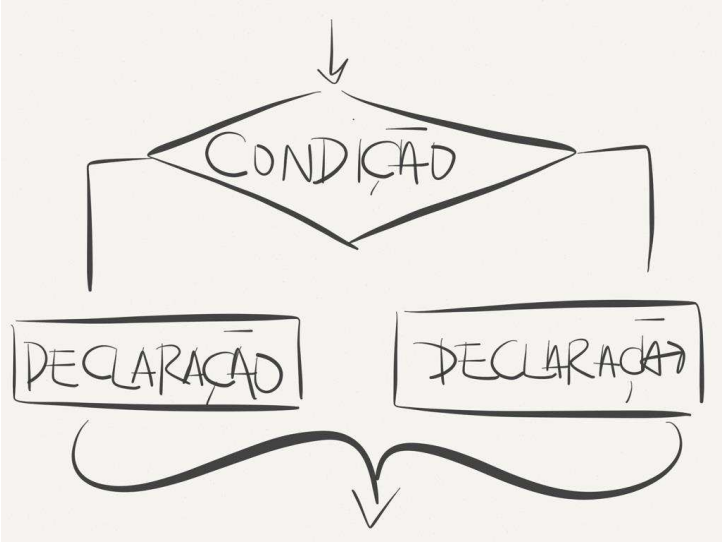
Bem, primeiro vamos ver a estrutura de um algoritmo em uma linguagem estruturada (na verdade, essa estrutura serve para diversos outros paradigmas), como mostra a imagem acima. Observem que, primeiro, há uma declaração de variáveis e, em seguida, há o corpo do algoritmo – essa estrutura é válida para todo e qualquer algoritmo estruturado: corpo e variáveis.

```
/* Representação de uma estrutura sequencial */

Inicio

{
    comando 1;
    comando 2;
    comando 3;
    comando 4;
    comando 5;
}

Fim
```

A Estrutura de Sequência realiza um conjunto de comandos sequencialmente ordenados, descendente e na ordem em que foram declarados, como mostra a imagem ao lado.

Esses comandos podem ser: uma leitura, uma gravação, uma atribuição, etc. O foco aqui é apenas demonstrar a sequência ordenada de execução dos comandos.

A Estrutura de Seleção (ou Decisão ou ainda Condicional) realiza um conjunto de comandos dependendo da veracidade da condição imposta, como mostra a imagem abaixo. Pode ser de três modos: por seleção simples ou composta; por meio de um operador ternário; ou por meio de uma seleção de múltipla escolha.

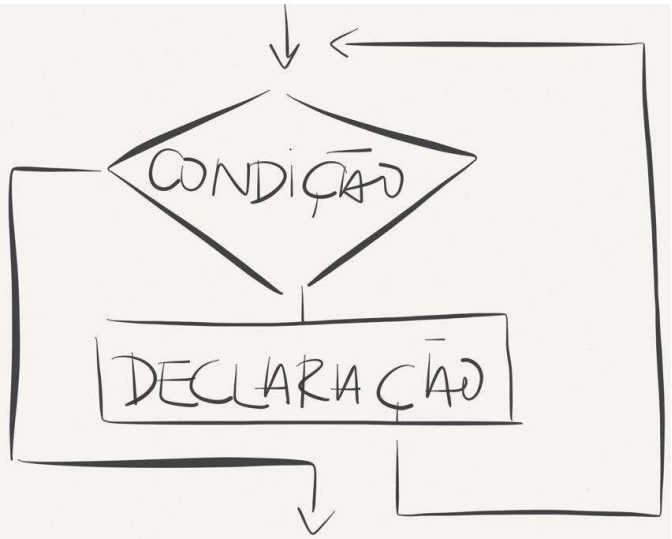



```
/* Representação de uma estrutura condicional por seleção simples/composta */

Inicio
    Se (condição 1) Então
        (comandos)
    Senão
        (comandos)
    Fim-Se
Fim

/* Representação de uma estrutura condicional por meio de um operador ternário */

(condição) ? (comandos do Então):(comandos do Senão)

/* Representação de uma estrutura condicional por meio de uma seleção de múltipla escolha */

Escolha a

    Caso 1:
        (comandos)
    Caso 2:
        (comandos)
    ...
    Caso Contrário
        (comandos)

Fim-Escolha
```

No primeiro caso, é bem simples! Exemplo: se eu tenho mais de 18 anos, então sou maior de idade; se não tenho mais de 18 anos, então sou menor de idade. *Tranquilo, né?!* Já quanto se utiliza um operador ternário, funciona de modo similar, i.e., a condição é ter ou não mais de 18 anos. Se eu tiver, sou maior de idade e se eu não tiver, sou menor de idade.

No último caso, há uma seleção com múltipla escolha. Caso eu tenha até 3 anos, então sou um bebê; caso eu tenha entre 4 e 12 anos, então sou uma criança; caso eu tenha entre 13 e 18 anos, então sou um adolescente; caso eu tenha entre 19 e 59

anos, então sou um adulto; por fim, caso contrário, i.e., eu tenha mais de 60 anos, então sou um idoso.



Por fim, vamos falar sobre a Estrutura de Repetição. Ela repete ou itera conjuntos de comandos dependendo da veracidade da condição imposta, como mostra a imagem abaixo.

Pode ser de três modos: por repetição pré-testada; por repetição pós-testada; ou por meio de repetição com variável de controle.

```
/* Representação de uma estrutura de repetição pré-testada */

Enquanto (condição) Faça
    (comandos)
Fim-Enquanto

/* Representação de uma estrutura de repetição pós-testada */

Repita
    (comandos)
Até (condição)

/* Representação de uma estrutura de repetição com variável de controle */

Para x=1 até 100 Faça
    (comandos)
Fim-Para
```

No primeiro caso, inicialmente testa-se a condição e, caso ela seja verdadeira, entra-se no bloco de comandos, saindo – apenas – quando a condição se tornar falsa. No segundo caso, entra-se no bloco de comandos, realizam-se todos os passos até testar a condição. Da mesma forma, caso ela seja falsa, o fluxo sai do bloco de comandos.

Por fim, na estrutura de repetição com variável de controle, utiliza-se uma variável que controlará a quantidade de iterações do bloco de comandos, isto é, nesse caso, há uma quantidade definida de repetições e, portanto, quando chegar à contagem máxima, o controle de fluxo sai do bloco de comandos e parte para a próxima instrução do programa.

Continuando...! Há outro conceito fundamental quando se fala no paradigma estruturado: a modularidade. *Bem, vocês já imaginaram um programa gigantesco em um único arquivo sequencial? Já pensaram como seria difícil de ler, debugar, etc?* Então... para tal, divide-se o programa em módulos ou subprogramas com entradas e saídas bem definidas.

Qual a vantagem de modularizar um programa? Bem, como cada bloco realiza uma tarefa específica, isso favorece a legibilidade, compreensão, manutenção, organização e reusabilidade do software. Esse último aspecto é muito importante, pois cada módulo pode ser arquivado formando uma biblioteca de subprogramas a serem utilizados futuramente.

Quanto à Entrada de Dados, ela é conhecida como Argumento; já a Saída de Dados é conhecida como Valor de Retorno. Pessoal, mas deve-se ter bom senso! Em geral, não há necessidade de módulos em programas muito pequenos – é perda de tempo. *E, professor, quais são os tipos de módulos?* Bem, os dois principais são: Função e Procedimento.

Em geral, a maior diferença entre ambos é que a Função sempre retorna algum valor e o Procedimento não retorna nenhum valor – apenas executa alguma ação. *Poxa, professor! Como eu vou lembrar disso na hora da prova?* Basta lembrar de uma função matemática. Você dá o valor de x e ele retorna f(x). Por exemplo: $f(x) = x^2 + x + 10$. Ora, para $x = 0$, $f(0) = 10$. *Tranquilo, não?!*

IMPORTANTE:

Pessoal, já recebi algumas perguntas de alunos dizendo que já viram procedimentos com retorno e funções sem retorno. Galera, isso é uma diferença teórica, uma diferença conceitual. Na prática, pode-se praticamente tudo em programação. Bacana?

1. (CESPE - 2004 – SESPA/PA - Analista de Sistemas) A programação estruturada é uma filosofia de projeto procedimental que restringe o número e o tipo de construções lógicas usadas para representar o detalhe do algoritmo.

Comentários:

Perfeito, são elas: sequência, seleção e repetição.

Gabarito: C

---

2. (CESPE - 2004 – STJ - Analista de Sistemas) Existem três construções fundamentais para a programação estruturada: seqüência, condição e repetição. Qualquer programa, independentemente da área de aplicação ou complexidade técnica, pode ser projetado e implementado usando apenas essas três construções lógicas. No entanto, o uso dogmático de apenas essas três construções pode algumas vezes causar dificuldades práticas.

Comentários:

Perfeito, é exatamente isso!

Gabarito: C

---

3. (CESPE - 2009 – ANAC - Analista de Sistemas) Tipos abstratos de dados só podem ser definidos em linguagens que implementam o paradigma de programação estruturada.

Comentários:

Na verdade, tipos abstratos de dados são definidas em linguagens orientadas a objetos.

Gabarito: E

---

4. (CESPE - 2010 – BASA - Analista de Sistemas) Na programação estruturada, existem estruturas de sequência, de decisão e de iteração. No primeiro tipo, uma tarefa é executada após a outra, linearmente. No segundo, a partir de um teste lógico, determinado trecho de código é executado, ou não. No terceiro, a partir de um teste lógico, determinado trecho de código é repetido por um número finito de vezes.

Comentários:

Perfeito, é exatamente isso! Cuidado com o final: ele não diz que é obrigatório que seja repetido um número finito de vezes, às vezes não há critério de parada! Ele apenas disse que na estrutura de iteração, a partir de um trecho lógico, um trecho pode ser repetido um número finito de vezes.

Gabarito: C

---

5. (CESPE - 2011 – EBC - Analista de Sistemas) Um programa que possui somente um ponto de entrada e somente um ponto de saída pode ser considerado estruturado.

Comentários:

Ué, sim! Pode ser considerado estruturado... esse "pode" ajuda a responder a questão! Se você julgasse esse item como errado, você estaria dizendo que em nenhuma hipótese um programa com somente um ponto de entrada e comente um ponto de saída poderia ser considerado estruturado, no entanto há essa possibilidade.

Gabarito: C

---

6. (CESPE - 2011 – EBC - Analista de Sistemas) Em programação estruturada, por meio do mecanismo de seleção, é possível testar determinada condição e estabelecer ações a serem realizadas.

Comentários:

Perfeito, é a estrutura de seleção!

Gabarito: C

7. (CESPE - 2011 – EBC - Analista de Sistemas) O mecanismo de iteração pode ser utilizado para sequenciar comandos, controlando a execução do programa.

Comentários:

Não, é o mecanismo de sequência que pode ser utilizado para sequenciar comandos.

Gabarito: E

8. (CESPE - 2013 – MPU - Analista de Sistemas) Mediante a utilização da técnica de programação estruturada, é possível obter programas mais legíveis e, consequentemente, menos suscetíveis a erros, e também definir e melhorar o grau de coesão entre as funções de um programa.

Comentários:

A questão não disse com quem ela está comparando. Assumindo que era com o paradigma de programação anterior, que utilizada bastantes desvios incondicionais, é correto dizer que melhora a legibilidade, depuração e grau de coesão.

Gabarito: C

9. (CESPE - 2013 – TRT/17 - Analista de Sistemas) Os módulos, também denominados de funções, rotinas ou procedimentos, são empregados para dividir um programa grande em partes menores, o que permite a realização, de forma individual, do desenvolvimento, do teste e da revisão, sem alterar o funcionamento do programa.

Comentários:

Perfeito, é exatamente isso!

Gabarito: C

10. (FCC - 2004 – TRF/4 - Analista de Sistemas) A técnica de programação estruturada contém uma estrutura básica adicional, originada pela estrutura Seleção, que é denominada:

a) Dountil.
b) Dowhile.
c) Case.
d) Sequence.
e) If-then-else.

Comentários:

A estrutura mais comum é If-Then-Else, a estrutura básica adicional é o Case.

Gabarito: C

---

11. (FCC - 2005 – TRT/11 - Analista de Sistemas) Na técnica de programação estruturada, a construção lógica denominada seleção é aplicada com as estruturas:

a) Do-while e repeat-until.
b) do-while e if-then-else.
c) case e do-while.
d) case e if-then-else.
e) if-then-else e repeat-until.

Comentários:

Trata-se da quarta opção: case e if-then-else.

Gabarito: D

---

12. (FCC - 2006 – AR/CE - Analista de Sistemas) Em programação estruturada, uma lógica de seleção é implementada apenas pela estrutura:

a) case.
b) do-while.
c) case e do-while.
d) case e if-then-else.
e) Case e do-while.

Comentários:

Trata-se da quarta opção: case e if-then-else.

Gabarito: D

---

13. (FCC - 2009 – TJ/PA - Analista de Sistemas) Na programação estruturada é adequado e fundamental:

a) o conhecimento e uso de construções de sequência, repetição e seleção.
b) o uso da mais baixa coesão possível entre tarefas de um módulo.
c) o uso do mais alto acoplamento possível entre módulos.
d) construir módulos cujo escopo de efeito não esteja no seu alcance de controle.
e) aumentar a redundância das interfaces modulares sempre que possível.

Comentários:

(a) Perfeito; (b) Não, alta coesão; (c) Baixo acoplamento; (d) Não, esteja no seu alcace de controle; (e) Não, reduzir a redundância.

Gabarito: A

---

14. (ESAF - 2008 – STN - Analista de Sistemas) Para resolver um determinado problema, um programador tem em mente como deve ser o programa principal que, por sua vez, controlará todas as outras tarefas distribuídas em sub-rotinas, para as quais deverá desenvolver os respectivos algoritmos. Este cenário exemplifica o conceito de programação:

a) Estruturada.
b) Orientada a Objetos.
c) Funcional.
d) Numérica.
e) Orientada a Aspectos.

Comentários:

Continuando...! Há outro conceito fundamental quando se fala no paradigma estruturado: a modularidade. Bem, vocês já imaginaram um programa gigantesco em um único arquivo sequencial? Já pensaram como seria difícil de ler, debugar, etc?

Então... para tal, divide-se o programa em módulos ou subprogramas com entradas e saídas bem definidas.

Trata-se da Programação Estruturada!

Gabarito: A

---

15. (Instituto Cidades - 2012 - TCM-GO - Auditor de Controle Externo – Informática – I) A programação estruturada é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Comentários:

Na verdade, isso é a Programação Orientada a Objetos.

Gabarito: E

---

16. (CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação) Considere que, em um sistema, seja necessário montar uma lista de opções e criar uma rotina para calcular a média das notas dos alunos. Nessa situação, é correto escolher um procedimento para a primeira ação e uma função para a segunda.

Comentários:

Montar uma lista de opções precisa de algum retorno? Não, então é um procedimento. *Calcular a média das notas dos alunos precisa de algum retorno?* Sim, desde que ele calcule e retorne esse valor – trata-se de uma função!

Gabarito: C

---

17. (FEPESE - 2010 - SEFAZ-SC - Auditor Fiscal da Receita Estadual - Parte III - Tecnologia da Informação) A programação estruturada é caracterizada por quais conceitos?

a) exceções, entrada e saída.
b) objeto, função e interação.
c) procedimento, função e argumento.
d) sequência, polimorfismo e hierarquia.

e) sequência, seleção e iteração.

Comentários:

Agora vamos mais a fundo! As linguagens estruturadas reduzem qualquer programa a três estruturas principais: sequência, seleção e repetição. Sequencial, porque possui uma ordem de execução; seletiva, porque seleciona um trecho de código dentre diversas opções de fluxo; e repetitiva, porque realiza a iteração de diversos trechos de código.

Iteração é sinônimo de Repetição.

Gabarito: E

---

18. (ESAF - 2009 - ANA - Analista Administrativo - Tecnologia da Informação - Desenvolvimento) Na programação estruturada, são necessários apenas três blocos de formas de controle para implementar algoritmos. São eles:

a) seleção, repetição e aninhamento.
b) empilhamento, aninhamento e operação.
c) sequência, aninhamento e seleção.
d) sequência, seleção e repetição.
e) função, operação e programa.

Comentários:

Agora vamos mais a fundo! As linguagens estruturadas reduzem qualquer programa a três estruturas principais: sequência, seleção e repetição. Sequencial, porque possui uma ordem de execução; seletiva, porque seleciona um trecho de código dentre diversas opções de fluxo; e repetitiva, porque realiza a iteração de diversos trechos de código.

Trata-se de Sequência, Seleção e Repetição.

Gabarito: D

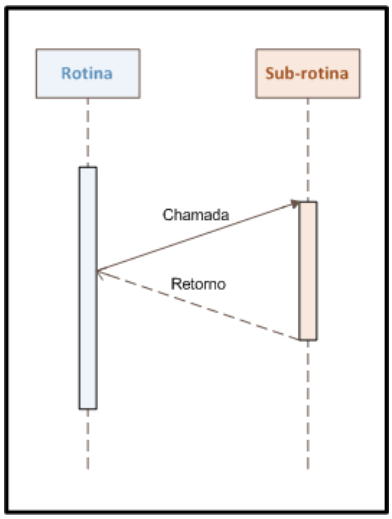
ACERTEI	ERREI

FUNÇÕES E PROCEDIMENTOS

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

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

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



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

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

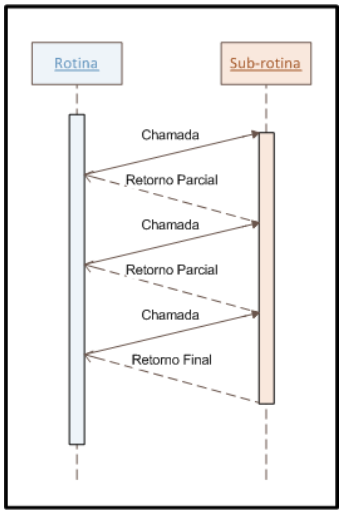
Algumas das vantagens na utilização de sub-rotinas durante a programação são: redução de código duplicado; possibilidade de reutilizar o mesmo código sem grandes alterações em outros programas; decomposição de problemas grandes em pequenas partes; melhorar a interpretação visual; esconder ou regular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

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

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

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

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



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

Vamos falar agora um pouco 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. Em Java, por exemplo, a passagem de parâmetros é sempre por valor! *Bacana?*

COMPILADORES

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! *Bacana?*

CÓDIGO-FONTE (SOURCE CODE)

O código-fonte é um conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica. Atualmente, com a diversificação de linguagens, o código pode ser escrito de forma totalmente modular, podendo um mesmo conjunto de códigos ser compartilhado por diversos programas e, até mesmo, linguagens.

CÓDIGO-OBJETO (OBJECT CODE)

O código-objeto representa a versão compilada e montada de um arquivo de código-fonte. Ele não é diretamente um arquivo executável, na medida em que ele deve ser ligado com outros códigos-objeto para criar um executável. Trata-se de uma porção do código de máquina[1], mas que contém ainda tabelas de símbolos, constantes, strings e outras referências.

PRÉ-PROCESSADOR (PREPROCESSOR)

O pré-processador é um programa que recebe um texto e efetua conversões léxicas, tais como substituição de macros, inclusão condicional, inclusão de ficheiros e exclusão de comentários. É baseado em Diretivas de Compilação (Ex: #define, #include, etc), que são comandos que não são compilados, dirigidos ao pré-processador e executado pelo compilador antes do processo de compilação em si.

Normalmente ele é responsável por mudanças no código fonte destinadas de acordo com decisões tomadas em tempo de compilação. Por exemplo, um programa em C permite instruções condicionais para o pré-processador que podem

[1] Alguns chamam o código-objeto de Código (Binário) de Máquina.

incluir ou não parte do código caso uma assertiva lógica seja verdadeira ou falsa, ou simplesmente um termo esteja definido ou não.

O uso de pré-processadores tem vindo a ser cada vez menos comum à medida que as linguagens recentes fornecem características mais abstratas em vez de características orientadas lexicalmente. Há também linguagens recentes que tem pouca ou nenhuma funcionalidade, como por exemplo a linguagem Java, que não possui um pré-processador (mas quem programou C na faculdade usou muito!).

COMPILADOR (COMPILER)

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: Análise ou Front-End (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e Síntese ou Back-end (Otimização de Código Intermediário e Geração de Código Final).

- Análise Léxica: nesta fase, um analisador léxico (ou *Scanner*) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados *tokens*, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

  Em português, nós dividimos um texto em substantivos, verbos, adjetivos, etc; em linguagens de programação, nós dividimos um texto em palavras-reservadas, identificadores, operadores, pontuação, números, etc. A Análise Léxica é uma forma de verificar um alfabeto, ou seja, conseguimos verificar se existe ou não algum caractere que não faz parte do alfabeto.

  Encontrar um erro léxico é um pouco difícil! Em geral, só haverá erro se o scanner achar um caractere que não pertence ao alfabeto da linguagem. Nesse caso, ou o projetista esqueceu de incluir o caractere no alfabeto ou realmente o usuário utilizou um caractere indevido. Se o programador escreve "fi" em vez de "if", o scanner não dará erro, visto que "fi" pode ser um identificador válido.

  Ex: *Eu vou passar nesse concurso!* Os tokens obtidos foram: "Eu"; "vou"; "passar"; "nesse"; "concurso"; "!".

- Análise Sintática: também conhecida como Análise Gramatical ou *Parsing*, analisa uma sequência de entrada para determinar sua estrutura gramatical, segundo

uma determinada gramática formal. Ela pega os tokens resultantes do processo de análise léxica e joga em uma estrutura hierárquica, como uma árvore. Assim como no português, verifica-se a estrutura do texto[2].

Assim como na linguagem natural, nas linguagens de programação se espera que os símbolos sejam dispostos de uma forma lógica uns em relação aos outros, tal como as palavras se juntam para formar expressões, orações e frases – isso é sintaxe. Um erro sintático, portanto, é um caso em que as "frases" do programa estão mal formuladas, aquilo que comumente chamamos de "erro gramatical".

Erros sintáticos são bastante comuns, entre eles nós podemos mencionar: parênteses que abrem, mas não fecham; dois números um ao lado do outro sem nenhum operador entre eles; duas instruções sem um ponto-e-vírgula entre elas; uma palavra-chave sendo usada numa posição inesperada. *Quem aí nunca esqueceu de fechar parênteses?*

O analisador sintático consulta a tabela de símbolos para verificar a presença de variáveis definidas pelo programador. Se o analisador encontra uma variável para a qual não existe descrição na tabela de símbolos, ele emite uma mensagem de erro. O analisador sintático também detecta construções ilegais como A = B + C = D.

No entanto, o que o analisador sintático não faz é verificar se os operadores = ou + são válidos para as variáveis A, B, C e D – quem faz isso é o analisador semântico. Logo, nós podemos afirmar que o analisador sintático tem uma certa preocupação com a operação, mas não com os operandos. E mais: os erros de sintaxe são sempre detectados em tempo de compilação/parse.

Ex: *string nome = "Diego"*; ora, se eu declarar um valor com aspas-duplas, preciso fechá-lo com aspas duplas e, não, simples.

- Análise Semântica: nesta fase, verificam-se erros semânticos, i.e., erros de sentido. Entre as principais atividades, estão a checagem de tipos, verificação de fluxos de controle e verificação de unicidade de declaração de variáveis. Essa fase também é encarregada de analisar a utilização dos identificadores e de ligar cada uma delas a sua declaração.

[2] Uma árvore de análise sintática de uma gramática de atributos é a árvore de análise sintática baseada em sua gramática BNF associada, com um conjunto possivelmente vazio de valores de atributos anexado a cada nó.

A semântica refere-se ao significado daquilo que se quer dizer. Da mesma forma que uma frase em linguagem natural pode estar gramaticalmente correta, mas não fazer o menor sentido, também as instruções dadas ao computador podem estar bem formatadas, mas não fazer aquilo que o programador quer - ou mesmo nada de útil ou ainda possível.

Erros semânticos também são bastante comuns, entre eles nós podemos mencionar: dividir um número por uma string; criar uma classe que herda de si mesma; usar o operador ^ achando que é de exponenciação, mas na verdade é um ou exclusivo; ou dividir zero por zero. Erros semânticos podem ser detectados tanto durante a compilação quanto durante a execução.

O Analisador Semântico usa a árvore de análise como entrada e verifica se os tipos de dados são apropriados usando informação da tabela de símbolos. O analisador semântico também faz as promoções adequadas de tipos de dados, tais como mudar um valor ou uma variável do tipo inteiro para ponto-flutuante, se tais promoções são suportadas pelas regras da linguagem.

Ex: int x = "Diego"; ora, eu não posso declarar uma variável do tipo inteiro e atribuir um valor textual.

- Geração de Código Intermediário: nesta fase, ocorre a transformação da árvore sintática em uma representação intermediária do código fonte. Esta linguagem intermediária é mais próxima da linguagem objeto do que o código fonte, mas ainda permite uma manipulação mais fácil do que se o código Assembly ou código de máquina fosse utilizado.

- Otimização de Código Intermediário: nesta fase, examina-se o código intermediário produzido durante a fase anterior com objetivo de produzir, através de algumas técnicas, um código que execute com bastante eficiência o programa. Utilizam-se técnicas que detectam padrões dentro do código produzido e os substitui por códigos mais eficientes. Simples assim!

- Geração de Código Final: chegamos à última fase do processo de compilação com a geração do código de montagem (ainda não é o código-objeto, apesar de muitos compiladores já realizarem a montagem). Um código-objeto não é imediatamente executável, visto que ainda há código binário a ser incluído no programa, tais como bibliotecas[3].

[3] Essa é a única fase da síntese que é obrigatória; as duas anteriores são opcionais.

MONTADOR (ASSEMBLER)

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

- Cross-assembler: é executado em um computador com um processador diferente daquele para o qual se está gerando código.

- Macro-assembler: dispõe de recursos de macro, efetuando a expansão do código cada vez que uma macro for encontrada.

- Micro-assembler: permite a escrita de micro-instruções, definindo-se assim um conjunto de instruções de um processador microprogramável.

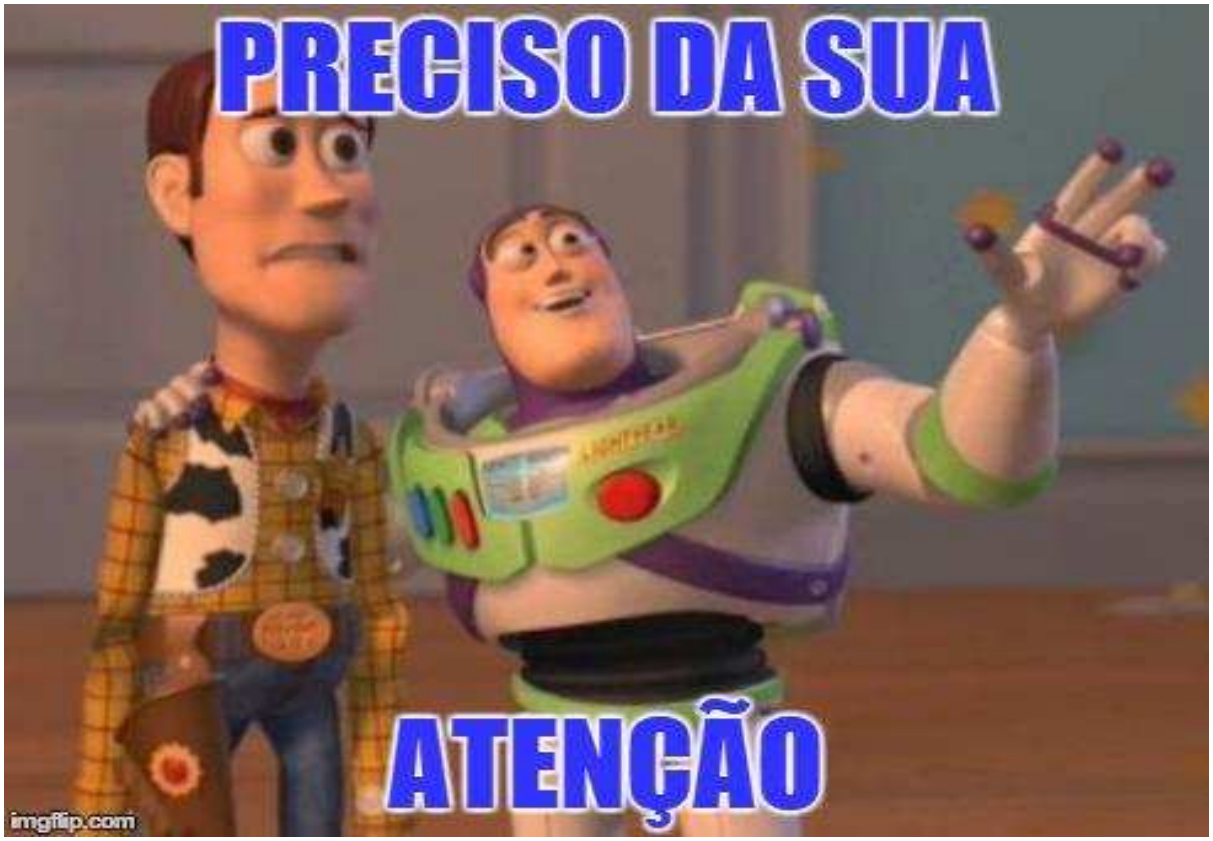
- Meta-assembler: é um assembler que pode montar programas para vários processadores diferentes.

- Assembler de um passo: varre o programa-fonte apenas uma vez, gerando o código (deve existir alguma forma de se revolver as referências adiante).

- Assembler de dois passos: varre o programa-fonte duas vezes para gerar o código, podendo assim resolver automaticamente as referências adiante.

A Linguagem de Montagem é traduzida para uma Linguagem de Máquina (Binária). Ambas são dependentes do hardware! *Como assim, professor?* Assembly é uma linguagem de programação, todavia cada família de processadores possui sua própria linguagem de montagem particular (Ex: INTEL, x86, MIPS, ARM, SPARC). Por essa razão, não se trata de uma linguagem portável.

A Montagem é um tipo de tradução, assim como a compilação ou a interpretação. Logo, ela pode existir por si só sem que haja compilação alguma. Eu vos-pergunto: *sendo rigoroso, um compilador traduz um código-fonte em um código-objeto?* Na teoria, ele traduz um código-fonte em um código de montagem, que depois vai à um montador para se transformar em código-objeto.



Em tese, compiladores produzem um código de montagem, que é passado a um montador para processamento posterior. No entanto, na prática, a maioria dos compiladores realizam a tarefa do montador, produzindo um código de máquina relocável, que pode ser passado diretamente para um ligador e/ou carregador, i.e., já entregam um código-objeto!

LIGADOR (LINKER OU LINK-EDITOR)

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

CARREGADOR (LOADER)

São programas que transferem o código de máquina de um módulo objeto para a memória e encaminham o início de sua execução. Eles recebem o módulo de carga como entrada, transferem seu código para a memória e realizam apenas os ajustes de realocação de acordo com o endereço base de memória. Um programa pode ser composto por partes independentemente carregadas e realocadas.

Existem dois tipos básicos de carregador: carregadores binários e realocáveis. Os carregadores binários (ou carregadores absolutos) são mais simples e apenas copiam o arquivo em formato binário para a memória, de tal forma que o arquivo executável é simplesmente uma imagem binária do programa em execução na memória.

Um programa que usa carregadores absolutos é associado com localizações específicas de memória, e por isso deve sempre ser carregado na mesma área de memória para serem executados corretamente (Ex: programas com extensão .COM). O carregador realocável pode ser colocado em qualquer local da memória para execução.

O programa executável realocável é semelhante ao programa executável absoluto, exceto que os endereços são todos relativos a zero (não são absolutos) e a informação de quais endereços relativos devem ser alterados quando o programa for colocado em execução estão junto com o arquivo executável (Ex: programas com extensão .EXE)[4].

BIBLIOTECAS

Bibliotecas são uma coleção de funções e definições utilizadas no desenvolvimento de software para algum propósito específico. Elas provêm serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular. O Ligador é o responsável, em geral, por fazer referências entre os executáveis e as bibliotecas.

É muito comum, ao precisar resolver determinado problema, utilizar-se de uma biblioteca escrita por outro programador. Isso pode ser vantajoso, pois reduz a necessidade de criar códigos, mas também pode ser perigoso, visto que pode-se optar por uma biblioteca mal implementada (ou até mesmo maliciosa) e acabar tendo grandes dores de cabeça.

Existem basicamente dois tipos de bibliotecas, as bibliotecas estáticas e as bibliotecas dinâmicas (ou compartilhadas). Ao compilar um programa que chama uma biblioteca estática, todo o código da biblioteca é copiado e inserido dentro do binário final. O termo "*estática*" se deve ao fato da linguagem se tornar uma parte estática do binário, não podendo ser compartilhada por outros programas.

[4] Atualmente também existem Carregadores que fazem a ligação de partes do programa em tempo de execução. Estes são chamados Carregadores-Ligadores.

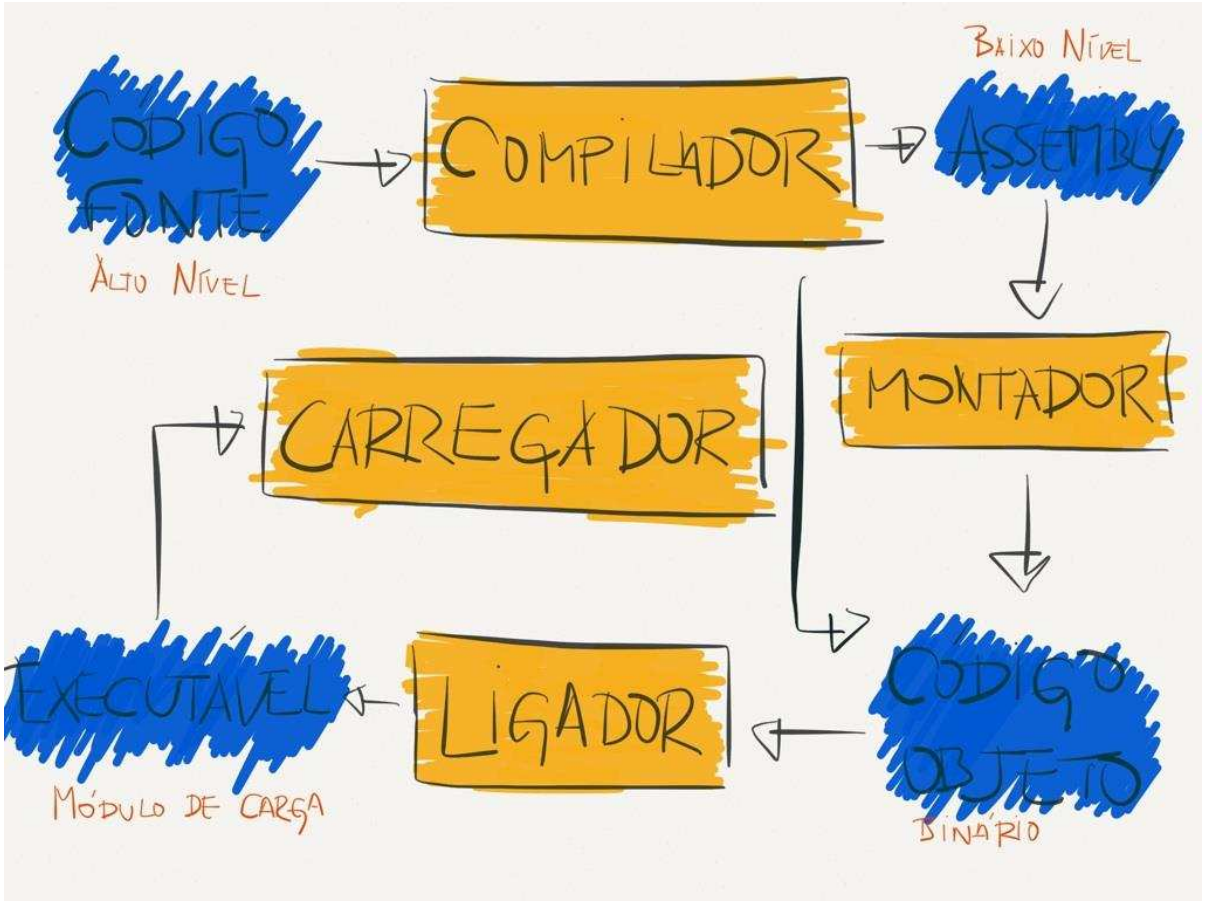
Dessa forma, mesmo quando a biblioteca presente no seu sistema mude, seja removida ou corrompida, o programa continuará funcionando. Porém além do seu binário final ocupar mais espaço em disco e memória (já que ele inclui a biblioteca estática), o seu programa não poderá usufruir de qualquer otimização que venha a ocorrer nesta biblioteca, a não ser que ele seja recompilado/reinstalado.

Ao contrário das bibliotecas estáticas, as bibliotecas dinâmicas ou compartilhadas não são inseridas em sua totalidade, elas são apenas referenciadas no binário final. O termo "*compartilhada*" expressa exatamente a característica de um ou mais programas poderem utilizar a mesma biblioteca, ocupando assim menos espaço em disco e na memória.

Além disso, a característica dinâmica da biblioteca compartilhada confere ao seu programa a possibilidade de usufruir de atualizações nas bibliotecas sem a necessidade de uma nova compilação/instalação. Entretanto pode acontecer de o programa não executar corretamente, seja por não encontrar a biblioteca ou (após uma atualização) pela biblioteca não ser mais compatível com seu programa.

O último ponto negativo da biblioteca compartilhada é que, devido a seu tempo de carregamento, é adicionada uma pequena latência nas chamadas a esta biblioteca, consequentemente seu programa será ligeiramente mais lento que o mesmo utilizando bibliotecas estáticas. No Windows, temos as .DLL (Dynamic-Link Libraries) e no Unix, temos as .LIB.

PROCESSO GENÉRICO DE COMPILAÇÃO



Um programador escreve módulos de código-fonte em uma linguagem de programação de alto nível (ex: C). Esse código-fonte passa por um pré-processador que manipula diretivas, macros, comentários, etc. Começa, então, de fato o processo de compilação por meio de uma análise léxica (tokenização e tabela de símbolos), seguida de uma análise sintática (parsing e estrutura de árvore).

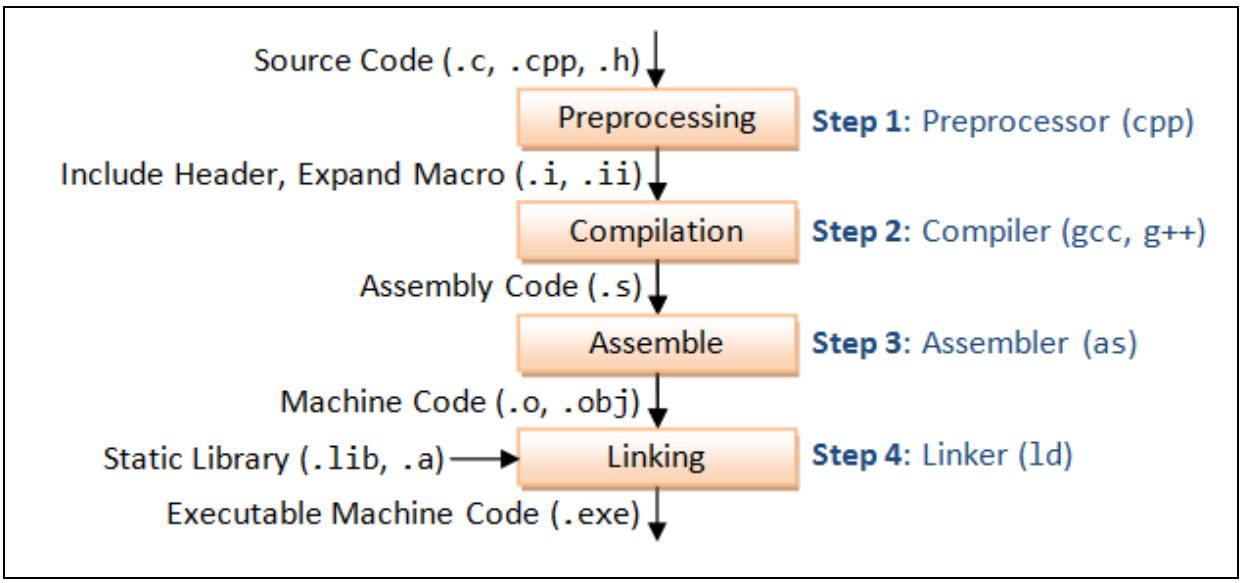
Avança-se à análise semântica (checagem de tipos e verificação de variáveis), depois à geração do código intermediário, sua otimização e, por fim, a geração do código final, que em geral é um programa em linguagem de montagem (Assembly). Porém, agora, nós temos diversos módulos em linguagem de programação de baixo nível, que os computadores ainda não conseguem entender.

Precisamos, então, do Montador[5]! Ele traduzirá cada modulo da linguagem de montagem para um código-objeto com algumas referências a resolver e que ainda não pode ser executado[6]. O Ligador combinará todos esses módulos-objeto em um

[5] Como já foi dito anteriormente, alguns compiladores fazem também o papel de montador e já entregam um código-objeto.
[6] O código-objeto é uma combinação de instruções de linguagem de máquina, dados e informações necessárias para colocar as instruções corretamente na memória.

módulo de carga, com diversas rotinas de bibliotecas para resolver as referências internas e externas.

Por fim, o carregador coloca o código de máquina nos locais apropriados da memória para ser executado pelo processador. Para agilizar o processo de tradução, algumas etapas são puladas ou combinadas. Alguns compiladores produzem módulos-objeto diretamente, e alguns sistemas utilizam Carregadores-Ligadores, que realizam as duas últimas etapas.



INTERPRETADORES

Como linguagens compiladas, linguagens interpretadas também têm um relacionamento um-para-muitos entre comandos do código fonte e instruções executáveis de máquina (i.e., nós escrevemos um comando no código-fonte, mas ele pode se transformar em diversas instruções de máquina). *E qual seria a diferença principal entre os compiladores e os interpretadores?*

Bem, diferente de compiladores, que leem todo o arquivo de código fonte antes de produzir uma sequência binária, interpretadores processam um comando de cada vez. Quem já trabalhou na prática sabe muito bem disso – a compilação ocorre toda de uma vez; na interpretação, é instrução por instrução (eu recomendo que vocês testem isso, mas só se estiverem com tempo sobrando).

Com tanto trabalho sendo feito "durante o voo", interpretadores geralmente são muito mais lentos do que compiladores. Pelo menos cinco dos seis passos requeridos por compiladores também devem ser feitos por interpretadores, e estes passos são realizados em "tempo real". Esta abordagem não dá oportunidade para otimização de código.

Além disso, a detecção de erros em interpretadores é geralmente limitada à verificação da sintaxe da linguagem e dos tipos de variáveis. Por exemplo, poucos interpretadores detectam possíveis operações aritméticas ilegais antes que ocorram ou avisam o programador antes de exceder os limites de um array. Os compiladores detectam erros bem mais variados.

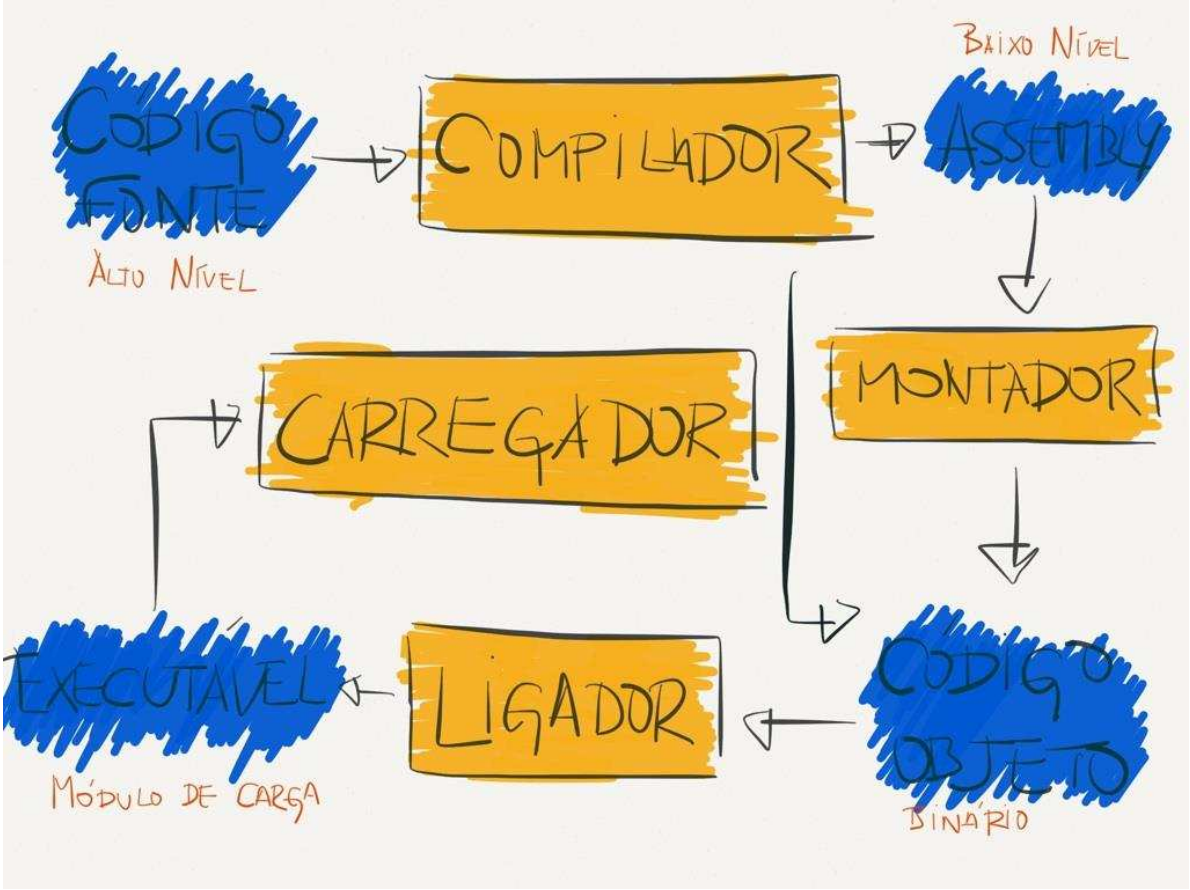
Apesar da velocidade de execução vagarosa e da verificação tardia de erros, existem boas razões para usar uma linguagem interpretada. A mais importante delas é que linguagens interpretadas permitem depuração em nível de código-fonte, tornando-as ideais para programadores iniciantes e usuários finais – podemos dizer que a detecção de erros é mais fácil para quem está começando.

CAIU na prova!

EXERCÍCIOS CESPE

1. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Uma diferença fundamental entre um compilador e um montador é que o compilador gera um arquivo executável a partir de um arquivo texto com o programa, enquanto o montador executa diretamente a descrição assembler, sem gerar arquivo na saída.

Comentários:



Conforme vimos em aula, bastava lembrar da imagem acima! Observem que há duas setinhas saindo do Compilador, i.e., ele pode gerar um código de montagem ou um código-objeto e, não, um arquivo executável. Ademais, um montador gera um arquivo de saída: o código-objeto!

Gabarito: E

2. (CESPE - 2006 – DATAPREV - Analista de Sistemas) O compilador é parte integrante do kernel de sistemas operacionais.

Comentários:

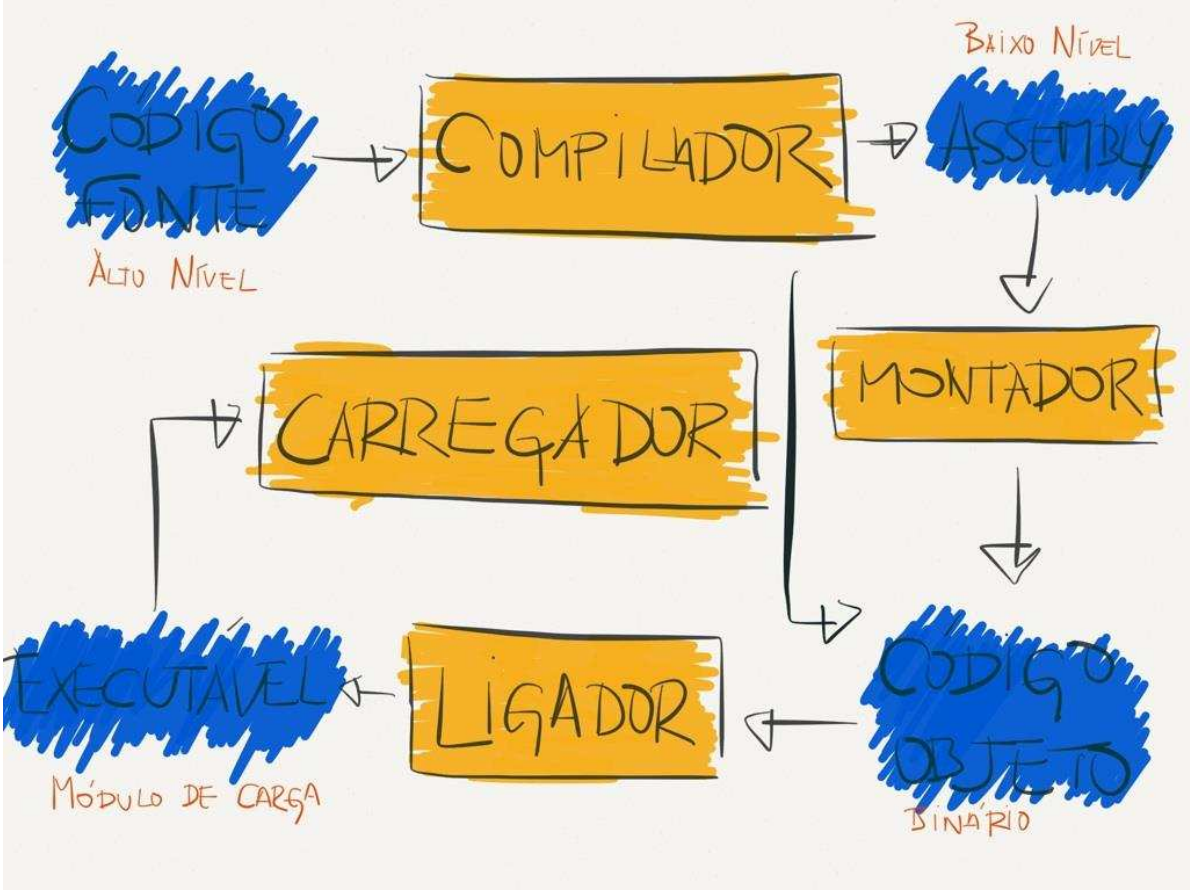
Não, isso não faz sentido! Eu imagino que essa questão foi formulada porque o GCC faz parte do SO Linux – mas não é regra haver compiladores como parte do kernel de sistemas operacionais.

Gabarito: E

---

3. (CESPE - 2014 – ANATEL - Analista de Sistemas) A compilação é o processo de análise de um programa escrito em linguagem de alto nível, dominando programa-fonte, e sua conversão em um programa equivalente, escrito em linguagem binária de máquina, denominado programa-objeto.

Comentários:



Conforme vimos em aula, está perfeito! O código-fonte entra no compilador e sai um código de montagem ou o próprio código (binário) de máquina.

Gabarito: C

---

4. (CESPE - 2011 – ECT - Analista de Sistemas) Instruções em linguagem de máquina são apresentadas na forma de padrões de bits utilizados para representar as operações internas ao computador. A linguagem de montagem constitui uma versão da linguagem de máquina; cada instrução é representada por uma cadeia de texto que descreve o que a instrução faz. Nesse processo, o montador é o elemento que converte instruções em linguagem de montagem para linguagem de máquina.

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, o código de montagem transforma a linguagem de montagem em código-objeto ou linguagem (binária) de máquina.

Gabarito: C

---

5. (CESPE - 2007 – TCU - Analista de Sistemas) Uma das principais tarefas do ligador, programa responsável por unir módulos-objeto em um único módulo, denominado módulo de carga, é resolver referências internas e externas.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Essa questão foi anulada! O item sugere que o arquivo de saída do Ligador seja um módulo pronto a ser executado, e, em algumas situações, isso não ocorre.

Gabarito: X

6. (CESPE - 2007 – TCU - Analista de Sistemas) Um interpretador pode ser considerado como um programa que lê um conjunto de instruções e as executa passo a passo. Programas interpretados são, em geral, menores e mais facilmente mantidos, embora sejam mais lentos que os programas compilados.

Comentários:

Com tanto trabalho sendo feito "durante o voo", interpretadores geralmente são muito mais lentos do que compiladores. Pelo menos cinco dos seis passos requeridos por compiladores também devem ser feitos por interpretadores, e estes passos são realizados em "tempo real". Esta abordagem não dá oportunidade para otimização de código.

Conforme vimos em aula, ele é realmente mais lento.

Gabarito: C

7. (CESPE - 2007 – TCU - Analista de Sistemas) A análise semântica — responsável por verificar se a estrutura gramatical do programa está correta, ou seja, se essa estrutura foi formada de acordo com as regras gramaticais da linguagem — é uma das tarefas realizadas pelo compilador.

Comentários:

Análise Sintática: também conhecida como Análise Gramatical ou Parsing, analisa uma sequência de entrada para determinar sua estrutura gramatical, segundo uma determinada gramática formal. Ela pega os tokens resultantes do processo de análise léxica e joga em uma estrutura hierárquica, como uma árvore. Assim como no português, verifica-se a estrutura do texto.

Conforme vimos em aula, quem verifica estrutura gramatical é a análise sintática e, não, análise semântica.

Gabarito: E

8. (CESPE - 2007 – TSE - Analista de Sistemas – A) Um programa pode ser composto por partes independentemente carregadas e realocadas. Um ligador

pode ser usado para resolver as referências aos símbolos externos às partes e para produzir um código executável.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o Ligador tem a função de conectar esses diversos elementos.

Gabarito: C

---

9. **(CESPE - 2007 – TSE - Analista de Sistemas – B)** Um carregador transfere para a memória códigos a serem executados. Se for transferido um código objeto, tem que ser armazenado nos endereços definidos quando foi gerado, pois um código objeto não pode ser realocado.

Comentários:

Não, um código-objeto pode ser realocado por meio da utilização de carregadores realocáveis, i.e., pode alocar o programa em partes não-fixas da memória.

Gabarito: E

---

10. **(CESPE - 2007 – TSE - Analista de Sistemas – C)** Para gerar um código objeto, um compilador precisa fazer a análise sintática e semântica de um programa. Para isso ser possível, a semântica da linguagem, mas não a sintaxe, é descrita na notação Backus-Naur Form (BNF).

Comentários:

Não, a Notação BNF é utilizada para descrever a Sintaxe!

Gabarito: E

---

11. (CESPE - 2007 – TSE - Analista de Sistemas – D) Os interpretadores não analisam sintaticamente os códigos fonte uma vez que os traduzem para um formato interno. Por isso, um interpretador traduz um código em menos tempo que um compilador.

Comentários:

Os interpretadores realizam tradução mais rápida que compiladores; no entanto, programas compilados são executados mais rapidamente que programas interpretados. O item está incorreto porque interpretadores analisam sintaticamente os códigos fontes (eles realizam toda a fase de análise, mas não de síntese).

Gabarito: E

---

12. (CESPE - 2008 – FUB - Analista de Sistemas) O compilador pode ser visto como um tradutor, uma vez que o mesmo transforma um programa de linguagem de alto nível para linguagem de baixo nível.

Comentários:

> *Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?*

Conforme vimos em aula, essa é uma definição mais ampla de compilação.

Gabarito: C

---

13. (CESPE - 2008 – TRT/5 - Analista de Sistemas) As definições de variáveis no código de um programa não interferem na geração de um código-objeto porque as variáveis não possuem relação direta com a entrada e a saída de dados em um programa.

Comentários:

Como não? Claro que interferem! Entradas e Saídas de dados geralmente são armazenadas em uma variável.

Gabarito: E

14. (CESPE - 2008 – ANAC - Analista de Sistemas) O compilador, em contraste com o montador, opera sobre uma linguagem de alto nível, enquanto o montador opera sobre uma linguagem de montagem.

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, é exatamente isso! O Compilador opera sobre uma linguagem de alto nível (entre outras) para traduzir para uma linguagem de máquina, já o Montador opera sobre uma linguagem de montagem (Assembly) para traduzir também para uma linguagem de máquina.

Gabarito: C

15. (CESPE - 2008 – ANAC - Analista de Sistemas) A criação da tabela de símbolos constitui tarefa realizada pelo ligador.

Comentários:

Análise Léxica: nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

Conforme vimos em aula, a Tabela de Símbolos é criada por Compiladores durante a Análise Léxica.

Gabarito: E

16. (CESPE - 2009 – FINEP - Analista de Sistemas – A) Um código escrito em linguagem de máquina (Assembly) deve ser compilado e ligado antes de ser executado.

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, Assembly é uma linguagem de montagem e, não, de máquina. Além disso, ela não é compilada, mas montada!

Gabarito: E

---

17. (CESPE - 2009 – FINEP - Analista de Sistemas – B) A análise semântica é uma tarefa normalmente realizada pelo link-editor.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o Link-Editor (Ligador) realiza a ligação. Os Compiladores e Interpretadores realizam a análise semântica.

Gabarito: E

---

18. (CESPE - 2009 – FINEP - Analista de Sistemas – C) A otimização de código, feita durante a fase de análise, é uma das tarefas do compilador.

Comentários:

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: Análise ou Front-End (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e Síntese ou Back-end (Otimização de Código Intermediário e Geração de Código Final).

Conforme vimos em aula. de fato, a otimização de código é uma das tarefas do compilador. No entanto, ela é realizada na fase de síntese e, não, de análise.

Gabarito: E

---

19. (CESPE - 2009 – FINEP - Analista de Sistemas – D) Um interpretador é classificado como um tradutor, uma vez que analisa e executa o código. O compilador, por realizar análise e síntese do código, não é considerado um tradutor.

Comentários:

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?

Conforme vimos em aula, compilador também é um tradutor! Compiladores, Montadores e Interpretadores são tradutores.

Gabarito: E

---

20. (CESPE - 2009 – FINEP - Analista de Sistemas – E) A construção da tabela de símbolos é atividade que pode ser iniciada durante a análise léxica.

Comentários:

Análise Léxica: nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.

Conforme vimos em aula, é realmente na análise léxica.

Gabarito: C

---

21. (CESPE - 2010 – INMETRO - Analista de Sistemas – B) Todo compilador de linguagem de programação de alto nível tem a responsabilidade de analisar o código fonte até a geração de código executável.

Comentários:

O compilador é um programa de computador que lê um código-fonte escrito em uma linguagem de alto nível e o traduz para uma linguagem de baixo nível. O processo de compilação é dividido em duas fases: Análise ou Front-End (Análise Léxica, Sintática, Semântica e Geração de Código Intermediário) e Síntese ou Back-end (Otimização de Código Intermediário e Geração de Código Final).

Conforme vimos em aula, a geração do código executável é feita na síntese, portanto ele tem a responsabilidade de analisar o código-fonte até antes da geração do código intermediário.

Gabarito: E

---

22. (CESPE - 2010 – INMETRO - Analista de Sistemas – C) Carregadores são programas usados exclusivamente por linguagens de programação de alto nível, com o objetivo de transferir um módulo de carga para a memória.

Comentários:

São programas que transferem o código de máquina de um módulo objeto para a memória e encaminham o início de sua execução. Eles recebem o módulo de carga como entrada, transferem seu código para a memória e realizam apenas os ajustes de realocação de acordo com o endereço base de memória. Um programa pode ser composto por partes independentemente carregadas e realocadas.

Não! Carregadores não são usados exclusivamente por linguagens de programação de alto nível.

Gabarito: E

---

23. (CESPE - 2010 – INMETRO - Analista de Sistemas – D) A declaração de variável "int 7g;" em um programa escrito na linguagem Java, leva a um erro de compilação detectado durante a análise sintática.

Comentários:

Lembrem-se que primeiro é feita a Análise Léxica, depois Análise Sintática e, por último, Análise Semântica. Nesse caso, já dará erro na Análise Léxica! Lembre-se que é nessa fase que se lê o código-fonte, caractere por caractere, e se divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos.

Quando começar a compilação, a análise léxica vai separar "int" e "7g". O Analisador Léxico lerá "7g", verificará se é uma palavra-chave, um literal, um separador, um operador, etc e, como não é, ele o classificará como um identificador. No entanto, identificadores não podem começar com número em Java, logo o compilador indicará um erro logo na Análise Léxica.

Lembrando que alguns números permitem, sim, sufixos – por exemplo: **long** (39832L), **float** (2.4f) e **double** (-7.832d). No entanto, a letra "g" não é um sufixo permitido em Java.

Gabarito: E

---

24. (CESPE - 2010 – INMETRO - Analista de Sistemas – E) Em programas que usam funções disponíveis em bibliotecas, as referências a estas funções serão resolvidas pelo ligador. No caso de bibliotecas estáticas, o código objeto das funções é integrado ao módulo executável durante o processo de ligação.

Comentários:

Perfeito! Executáveis e Bibliotecas fazem referências mútuas conhecidas como ligações, tarefa tipicamente realizada por um ligador. Lembrando que Bibliotecas Dinâmicas são aquelas que podem ser compartilhadas com vários programas e Bibliotecas Estáticas são aquelas que podem ser ligadas somente a um programa individualmente.

Gabarito: C

---

25. (CESPE - 2010 – INMETRO - Analista de Sistemas) A análise léxica/sintática de uma linguagem que tem palavras reservadas tende a ser mais complexa que a de linguagens que têm apenas palavras-chave usadas também como identificadores.

Comentários:

Não, isso não interfere na complexidade da análise léxica/sintática.

Gabarito: E

---

26. (CESPE - 2010 – TRE/MT - Analista de Sistemas) Durante a compilação de um código-fonte, a fase do compilador que é responsável por produzir uma sequência de tokens é a:

a) análise léxica.
b) análise semântica.
c) análise sintática.
d) geração de código executável.
e) verificação de tipos.

Comentários:

> ***Análise Léxica:*** *nesta fase, um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.*

Conforme vimos em aula, a tokenização ocorre na Análise Léxica!

Gabarito: A

---

27. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A) Compiladores são projetados para um tipo específico de hardware e de sistema operacional.

Comentários:

Não, compiladores não dependem do hardware.

Gabarito: E

---

28. (CESPE - 2010 – TRE/MT - Analista de Sistemas – B) A interpretação nada mais é do que uma compilação cruzada.

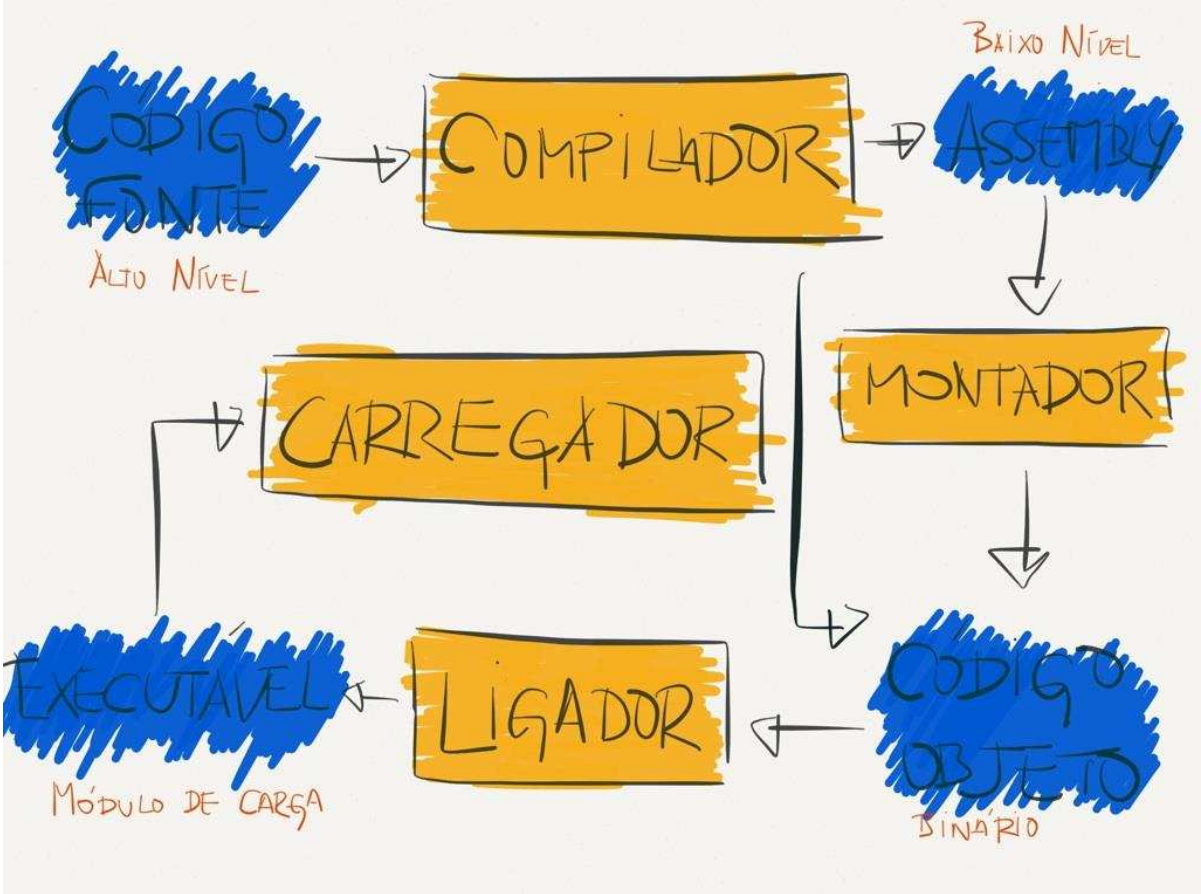
Comentários:

A compilação cruzada é aquela capaz de produzir código executável em uma plataforma diferente da qual o compilador está sendo executado. Interpretadores não podem ser definidos como uma compilação cruzada.

Gabarito: E

29. (CESPE - 2011 – FUB - Analista de Sistemas) Na programação empregando uma linguagem de alto nível, a utilização de um compilador implica o uso de um ligador e de um carregador para a correta execução do programa; por outro lado, a utilização de um interpretador, que simula a existência de um processador cujas instruções são aquelas da linguagem de alto nível empregada, torna desnecessárias as etapas de ligação e carga.

Comentários:



Conforme vimos em aula, a questão está correta. A interpretação torna desnecessárias as duas últimas etapas (Ligação e Carga).

Gabarito: C

30. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Em programa escrito em linguagem de alto nível e traduzido por compilador, alguns comandos que fazem parte desse código são instruções da linguagem de programação, enquanto outros comandos são instruções típicas do compilador denominadas diretivas.

Comentários:

O pré-processador é um programa que recebe um texto e efetua conversões léxicas, tais como substituição de macros, inclusão condicional, inclusão de ficheiros e exclusão de comentários. É baseado em Diretivas de Compilação (Ex: #define, #include, etc), que são comandos que não são compilados, dirigidos ao pré-processador e executado pelo compilador antes do processo de compilação em si.

Conforme vimos em aula, perfeito! *Lembram-se das diretivas de compilação?* Elas não são compiladas, são pré-processadas antes do processo de compilação em si.

Gabarito: C

---

31. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Durante a execução de um programa que utiliza funções em uma biblioteca dinâmica, esta deve ser carregada em memória e ligada ao processo em tempo de execução.

Comentários:

Existem basicamente dois tipos de bibliotecas, as bibliotecas estáticas e as bibliotecas dinâmicas (ou compartilhadas). Ao compilar um programa que chama uma biblioteca estática, todo o código da biblioteca é copiado e inserido dentro do binário final. O termo "estática" se deve ao fato da linguagem se tornar uma parte estática do binário, não podendo ser compartilhada por outros programas.

Conforme vimos em aula, bibliotecas dinâmicas são ligadas em tempo de execução.

Gabarito: C

---

32. (CESPE - 2011 – TRE/ES - Analista de Sistemas) O link-editor tem a função de vincular os dados de um programa aos programas de sistema e a outros programas de usuário.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o ligador tem a função de vincular diversos componentes.

Gabarito: C

---

33. (CESPE - 2010 – TRF/4 - Analista de Sistemas - A) Interpretadores são programas que convertem códigos escritos em linguagem de alto nível para programas em linguagem de máquina.

Comentários:

Como linguagens compiladas, linguagens interpretadas também têm um relacionamento um-para-muitos entre comandos do código fonte e instruções executáveis de máquina (i.e., nós escrevemos um comando no código-fonte, mas ele pode se transformar em diversas instruções de máquina). E qual seria a diferença principal entre os compiladores e os interpretadores?

Conforme vimos em aula, a questão trata do compilador – ele realmente converte códigos escritos em uma linguagem de alto nível para programas em linguagem de máquina (ou código-objeto)! Já os interpretadores convertem um programa escrito em linguagem de programação de alto nível em código executável.

Gabarito: E

---

34. (CESPE - 2010 – TRF/4 - Analista de Sistemas - B) Linguagens de alto nível cumprem tarefas mais substanciais com um número menor de comandos, mas exigem programas tradutores denominados compiladores para converter programas em linguagem de alto nível para linguagem de máquina.

Comentários:

Perfeito! Linguagens de alto nível, em geral, necessitam de menos comandos para realizar uma mesma tarefa que uma linguagem de baixo nível. Além disso, para converter programas em linguagens de alto nível para linguagem de máquina, é necessário um compilador.

Gabarito: C

35. (CESPE - 2010 – TRF/4 - Analista de Sistemas - C) Um computador pode entender qualquer linguagem de máquina, pois a linguagem de máquina não é definida pelo projeto de hardware do computador.

Comentários:

Não! A Linguagem de Montagem é traduzida para uma Linguagem de Máquina (0 e 1) específica de cada processador. Apesar de ser um código binário, há diversas maneiras de interpretá-lo de acordo com o processador (hardware) para o qual ele foi escrito.

Gabarito: E

36. (CESPE - 2010 – TRF/4 - Analista de Sistemas - D) Programadores podem escrever instruções em várias linguagens de programação e todas são entendidas diretamente pelos computadores sem a necessidade de tradução.

Comentários:

Claro que não! Elas precisam ser traduzidas, porque computadores só entendem 0 e 1 (0V e 5V).

Gabarito: E

37. (CESPE - 2010 – TRF/4 - Analista de Sistemas - E) Softwares escritos em linguagens de máquina são portáveis.

Comentários:

Elas não são portáveis, porque eles dependem da arquitetura do processador.

Gabarito: E

38. (CESPE - 2010 – BASA - Analista de Sistemas) Um interpretador é um programa que lê um código escrito em uma linguagem e o converte em um código equivalente em outra linguagem. No processo de conversão, o interpretador relata a presença de erros no código original.

Comentários:

Eu sinceramente não consigo encontrar erros nessa questão. A primeira parte está correta e vale tanto para compiladores quanto para interpretadores. A segunda parte também está correta e também vale para compiladores e interpretadores. Acredito que o examinador deve ter retirado essa frase originalmente falando de um compilador, mudou para interpretador e achou que ia ser errado, mas – para mim – ela continua correta.

Vamos por partes! Ambos leem um código escrito em uma linguagem e o convertem para outra linguagem equivalente. A diferença é que o compilador transforma uma linguagem em uma outra linguagem equivalente que posteriormente deve ser executada; já o interpretador transforma em uma linguagem em uma outra linguagem equivalente que é imediatamente executada. Logo, não há erro nessa primeira parte da questão.

Na segunda parte, ele diz que, no processo de conversão, o interpretador relata a presença de erros no código original. Ora, uma vez que interpretadores leem declarações singulares, eles exibem um erro de cada vez e você terá que consertar esse erro para interpretar a próxima declaração. Já os compiladores exibem todos os erros e alertas de uma vez e, sem consertar todos os erros, o programa não poderá ser executado. Para mim, não há contradição com o que foi dito!

A grande diferença é que o interpretador, em geral, vai se limitar a detectar erros somente de sintaxe da linguagem e dos tipos de variáveis, enquanto os compiladores detectam uma variedade maior de erros.

Gabarito: E

---

39. (CESPE - 2010 – INMETRO - Analista de Sistemas) Um interpretador traduz um programa descrito no nível da linguagem para o nível da máquina, enquanto o compilador eleva a máquina ao nível da linguagem, para que o programa execute a partir da fonte.

Comentários:

Não, a questão claramente inverteu compiladores e interpretadores.

Gabarito: E

---

40. (CESPE - 2011 – ECT - Analista de Sistemas) No programa em linguagem de alto nível, os interpretadores executam os passos definidos para cada instrução e produzem o mesmo resultado que o do programa compilado. Entretanto, a execução de um programa em linguagem de alto nível com o uso de interpretadores é mais lenta que a execução de um programa compilado, uma vez que precisa examinar cada instrução no programa-fonte, à medida que ela ocorre, e desviar para a rotina que executa a instrução.

Comentários:

Perfeito, essa é uma questão para decorar! Muito bem escrita e boa para sedimentar alguns conceitos!

Gabarito: C

---

41. (CESPE - 2009 – TCE/AC - Analista de Sistemas - A) Os programas escritos em linguagem de programação de alto nível precisam ser convertidos em programas de máquina, sendo o linker um tipo de software básico que efetua essa tradução.

Comentários:

Vamos ser bem objetivos? Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (podendo ser uma linguagem de montagem ou código-objeto). Vejamos algumas definições para ajudar a esclarecer! Bacana?

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o Linker (Ligador) não efetua tradução alguma! Quem faz isso é o Compilador!

Gabarito: E

---

42. (CESPE - 2007 – TCU - Analista de Sistemas) Um montador é considerado um software de sistema, responsável pela tradução de uma linguagem de alto nível para uma linguagem de baixo nível (linguagem simbólica).

Comentários:

Existe uma maneira de efetuar tradução de forma mais rápida e simples, por meio de um Montador. O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex: Assembly) em um equivalente em linguagem de máquina (com algumas referências). Galera, não confundam Assembler, que é um montador, com Assembly, que é uma linguagem de programação de baixo nível.

Conforme vimos em aula, montadores não são softwares de sistema e não realizam a tradução de linguagens de alto nível para uma linguagem de baixo nível. Na verdade, ele traduz linguagens de baixo nível para linguagem de máquina.

Gabarito: E

---

43. (CESPE - 2008 – FUB - Analista de Sistemas) O montador realiza a tarefa de combinar módulos objetos em um módulo de carga.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, é o Ligador que é responsável por receber como entrada os diversos módulos e conectá-los, gerando como saída um único módulo de carga.

Gabarito: E

---

44. (CESPE - 2008 – FUB - Analista de Sistemas – C) Para que um programa possa ser executado, seu código de máquina deve estar presente na memória. O ligador é o programa do sistema responsável por transferir o código de máquina de um módulo objeto para a memória e encaminhar o início de sua execução.

Comentários:

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo). Ele é responsável por resolver referências internas e externas.

Conforme vimos em aula, o ligador é o responsável por interligar os diversos módulos de um programa para gerar o programa que será posteriormente carregado para a memória. O carregador é responsável por transferir o código de máquina de um módulo objeto para a memória e encaminhar o início de sua execução.

Gabarito: E

---

45. (CESPE - 2006 – DETRAN - Analista de Sistemas - D) O termo linguagem de máquina é usado para denominar linguagens do tipo assembler ou C++.

Comentários:

Não, Linguagem de Máquina é código binário (0 e 1). Assembler não é linguagem, é um programa chamado Montador. Assembly é uma linguagem de montagem e C++ é uma linguagem de alto nível orientada a objetos.

Gabarito: E

ACERTEI	ERREI

LISTA DE EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)
PARADIGMA ESTRUTURADO

1. (CESPE - 2004 – SESPA/PA - Analista de Sistemas) A programação estruturada é uma filosofia de projeto procedimental que restringe o número e o tipo de construções lógicas usadas para representar o detalhe do algoritmo.

2. (CESPE - 2004 – STJ - Analista de Sistemas) Existem três construções fundamentais para a programação estruturada: seqüência, condição e repetição. Qualquer programa, independentemente da área de aplicação ou complexidade técnica, pode ser projetado e implementado usando apenas essas três construções lógicas. No entanto, o uso dogmático de apenas essas três construções pode algumas vezes causar dificuldades práticas.

3. (CESPE - 2009 – ANAC - Analista de Sistemas) Tipos abstratos de dados só podem ser definidos em linguagens que implementam o paradigma de programação estruturada.

4. (CESPE - 2010 – BASA - Analista de Sistemas) Na programação estruturada, existem estruturas de sequência, de decisão e de iteração. No primeiro tipo, uma tarefa é executada após a outra, linearmente. No segundo, a partir de um teste lógico, determinado trecho de código é executado, ou não. No terceiro, a partir de um teste lógico, determinado trecho de código é repetido por um número finito de vezes.

5. (CESPE - 2011 – EBC - Analista de Sistemas) Um programa que possui somente um ponto de entrada e somente um ponto de saída pode ser considerado estruturado.

6. (CESPE - 2011 – EBC - Analista de Sistemas) Em programação estruturada, por meio do mecanismo de seleção, é possível testar determinada condição e estabelecer ações a serem realizadas.

7. (CESPE - 2011 – EBC - Analista de Sistemas) O mecanismo de iteração pode ser utilizado para sequenciar comandos, controlando a execução do programa.

8. (CESPE - 2013 – MPU - Analista de Sistemas) Mediante a utilização da técnica de programação estruturada, é possível obter programas mais legíveis e,

consequentemente, menos suscetíveis a erros, e também definir e melhorar o grau de coesão entre as funções de um programa.

9. (CESPE - 2013 – TRT/17 - Analista de Sistemas) Os módulos, também denominados de funções, rotinas ou procedimentos, são empregados para dividir um programa grande em partes menores, o que permite a realização, de forma individual, do desenvolvimento, do teste e da revisão, sem alterar o funcionamento do programa.

10. (FCC - 2004 – TRF/4 - Analista de Sistemas) A técnica de programação estruturada contém uma estrutura básica adicional, originada pela estrutura Seleção, que é denominada:

a) Dountil.
b) Dowhile.
c) Case.
d) Sequence.
e) If-then-else.

11. (FCC - 2005 – TRT/11 - Analista de Sistemas) Na técnica de programação estruturada, a construção lógica denominada seleção é aplicada com as estruturas:

a) Do-while e repeat-until.
b) do-while e if-then-else.
c) case e do-while.
d) case e if-then-else.
e) if-then-else e repeat-until.

12. (FCC - 2006 – AR/CE - Analista de Sistemas) Em programação estruturada, uma lógica de seleção é implementada apenas pela estrutura:

a) case.
b) do-while.
c) case e do-while.
d) case e if-then-else.
e) Case e do-while.

13. (FCC - 2009 – TJ/PA - Analista de Sistemas) Na programação estruturada é adequado e fundamental:

a) o conhecimento e uso de construções de sequência, repetição e seleção.
b) o uso da mais baixa coesão possível entre tarefas de um módulo.
c) o uso do mais alto acoplamento possível entre módulos.
d) construir módulos cujo escopo de efeito não esteja no seu alcance de controle.
e) aumentar a redundância das interfaces modulares sempre que possível.

14. (ESAF - 2008 – STN - Analista de Sistemas) Para resolver um determinado problema, um programador tem em mente como deve ser o programa principal que, por sua vez, controlará todas as outras tarefas distribuídas em sub-rotinas, para as quais deverá desenvolver os respectivos algoritmos. Este cenário exemplifica o conceito de programação:

a) Estruturada.
b) Orientada a Objetos.
c) Funcional.
d) Numérica.
e) Orientada a Aspectos.

15. (Instituto Cidades - 2012 - TCM-GO - Auditor de Controle Externo – Informática – I) A programação estruturada é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

16. (CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação) Considere que, em um sistema, seja necessário montar uma lista de opções e criar uma rotina para calcular a média das notas dos alunos. Nessa situação, é correto escolher um procedimento para a primeira ação e uma função para a segunda.

17. (FEPESE - 2010 - SEFAZ-SC - Auditor Fiscal da Receita Estadual - Parte III - Tecnologia da Informação) A programação estruturada é caracterizada por quais conceitos?

a) exceções, entrada e saída.
b) objeto, função e interação.
c) procedimento, função e argumento.
d) sequência, polimorfismo e hierarquia.
e) sequência, seleção e iteração.

18. (ESAF - 2009 - ANA - Analista Administrativo - Tecnologia da Informação - Desenvolvimento) Na programação estruturada, são necessários apenas três blocos de formas de controle para implementar algoritmos. São eles:

a) seleção, repetição e aninhamento.
b) empilhamento, aninhamento e operação.
c) sequência, aninhamento e seleção.
d) sequência, seleção e repetição.
e) função, operação e programa.

LISTA DE EXERCÍCIOS COMENTADOS (CESPE)
COMPILADORES E INTERPRETADORES

1. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Uma diferença fundamental entre um compilador e um montador é que o compilador gera um arquivo executável a partir de um arquivo texto com o programa, enquanto o montador executa diretamente a descrição assembler, sem gerar arquivo na saída.

2. (CESPE - 2006 – DATAPREV - Analista de Sistemas) O compilador é parte integrante do kernel de sistemas operacionais.

3. (CESPE - 2014 – ANATEL - Analista de Sistemas) A compilação é o processo de análise de um programa escrito em linguagem de alto nível, dominando programa-fonte, e sua conversão em um programa equivalente, escrito em linguagem binária de máquina, denominado programa-objeto.

4. (CESPE - 2011 – ECT - Analista de Sistemas) Instruções em linguagem de máquina são apresentadas na forma de padrões de bits utilizados para representar as operações internas ao computador. A linguagem de montagem constitui uma versão da linguagem de máquina; cada instrução é representada por uma cadeia de texto que descreve o que a instrução faz. Nesse processo, o montador é o elemento que converte instruções em linguagem de montagem para linguagem de máquina.

5. (CESPE - 2007 – TCU - Analista de Sistemas) Uma das principais tarefas do ligador, programa responsável por unir módulos-objeto em um único módulo, denominado módulo de carga, é resolver referências internas e externas.

6. (CESPE - 2007 – TCU - Analista de Sistemas) Um interpretador pode ser considerado como um programa que lê um conjunto de instruções e as executa passo a passo. Programas interpretados são, em geral, menores e mais facilmente mantidos, embora sejam mais lentos que os programas compilados.

7. (CESPE - 2007 – TCU - Analista de Sistemas) A análise semântica — responsável por verificar se a estrutura gramatical do programa está correta, ou seja, se essa estrutura foi formada de acordo com as regras gramaticais da linguagem — é uma das tarefas realizadas pelo compilador.

8. (CESPE - 2007 – TSE - Analista de Sistemas – A) Um programa pode ser composto por partes independentemente carregadas e realocadas. Um ligador pode ser usado para resolver as referências aos símbolos externos às partes e para produzir um código executável.

9. (CESPE - 2007 – TSE - Analista de Sistemas – B) Um carregador transfere para a memória códigos a serem executados. Se for transferido um código objeto, tem que ser armazenado nos endereços definidos quando foi gerado, pois um código objeto não pode ser realocado.

10. (CESPE - 2007 – TSE - Analista de Sistemas – C) Para gerar um código objeto, um compilador precisa fazer a análise sintática e semântica de um programa. Para isso ser possível, a semântica da linguagem, mas não a sintaxe, é descrita na notação Backus-Naur Form (BNF).

11. (CESPE - 2007 – TSE - Analista de Sistemas – D) Os interpretadores não analisam sintaticamente os códigos fonte uma vez que os traduzem para um formato interno. Por isso, um interpretador traduz um código em menos tempo que um compilador.

12. (CESPE - 2008 – FUB - Analista de Sistemas) O compilador pode ser visto como um tradutor, uma vez que o mesmo transforma um programa de linguagem de alto nível para linguagem de baixo nível.

13. (CESPE - 2008 – TRT/5 - Analista de Sistemas) As definições de variáveis no código de um programa não interferem na geração de um código-objeto porque as variáveis não possuem relação direta com a entrada e a saída de dados em um programa.

14. (CESPE - 2008 – ANAC - Analista de Sistemas) O compilador, em contraste com o montador, opera sobre uma linguagem de alto nível, enquanto o montador opera sobre uma linguagem de montagem.

15. (CESPE - 2008 – ANAC - Analista de Sistemas) A criação da tabela de símbolos constitui tarefa realizada pelo ligador.

16. (CESPE - 2009 – FINEP - Analista de Sistemas – A) Um código escrito em linguagem de máquina (Assembly) deve ser compilado e ligado antes de ser executado.

17. (CESPE - 2009 – FINEP - Analista de Sistemas – B) A análise semântica é uma tarefa normalmente realizada pelo link-editor.

18. (CESPE - 2009 – FINEP - Analista de Sistemas – C) A otimização de código, feita durante a fase de análise, é uma das tarefas do compilador.

19. (CESPE - 2009 – FINEP - Analista de Sistemas – D) Um interpretador é classificado como um tradutor, uma vez que analisa e executa o código. O compilador, por realizar análise e síntese do código, não é considerado um tradutor.

20. (CESPE - 2009 – FINEP - Analista de Sistemas – E) A construção da tabela de símbolos é atividade que pode ser iniciada durante a análise léxica.

21. (CESPE - 2010 – INMETRO - Analista de Sistemas – B) Todo compilador de linguagem de programação de alto nível tem a responsabilidade de analisar o código fonte até a geração de código executável.

22. (CESPE - 2010 – INMETRO - Analista de Sistemas – C) Carregadores são programas usados exclusivamente por linguagens de programação de alto nível, com o objetivo de transferir um módulo de carga para a memória.

23. (CESPE - 2010 – INMETRO - Analista de Sistemas – D) A declaração de variável "int 7g;" em um programa escrito na linguagem Java, leva a um erro de compilação detectado durante a análise sintática.

24. (CESPE - 2010 – INMETRO - Analista de Sistemas – E) Em programas que usam funções disponíveis em bibliotecas, as referências a estas funções serão resolvidas pelo ligador. No caso de bibliotecas estáticas, o código objeto das funções é integrado ao módulo executável durante o processo de ligação.

25. (CESPE - 2010 – INMETRO - Analista de Sistemas) A análise léxica/sintática de uma linguagem que tem palavras reservadas tende a ser mais complexa que a de linguagens que têm apenas palavras-chave usadas também como identificadores.

26. (CESPE - 2010 – TRE/MT - Analista de Sistemas) Durante a compilação de um código-fonte, a fase do compilador que é responsável por produzir uma sequência de tokens é a:

    a) análise léxica.

b) análise semântica.
c) análise sintática.
d) geração de código executável.
e) verificação de tipos.

27. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A) Compiladores são projetados para um tipo específico de hardware e de sistema operacional.

28. (CESPE - 2010 – TRE/MT - Analista de Sistemas – B) A interpretação nada mais é do que uma compilação cruzada.

29. (CESPE - 2011 – FUB - Analista de Sistemas) Na programação empregando uma linguagem de alto nível, a utilização de um compilador implica o uso de um ligador e de um carregador para a correta execução do programa; por outro lado, a utilização de um interpretador, que simula a existência de um processador cujas instruções são aquelas da linguagem de alto nível empregada, torna desnecessárias as etapas de ligação e carga.

30. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Em programa escrito em linguagem de alto nível e traduzido por compilador, alguns comandos que fazem parte desse código são instruções da linguagem de programação, enquanto outros comandos são instruções típicas do compilador denominadas diretivas.

31. (CESPE - 2011 – TJ/ES - Analista de Sistemas) Durante a execução de um programa que utiliza funções em uma biblioteca dinâmica, esta deve ser carregada em memória e ligada ao processo em tempo de execução.

32. (CESPE - 2011 – TRE/ES - Analista de Sistemas) O link-editor tem a função de vincular os dados de um programa aos programas de sistema e a outros programas de usuário.

33. (CESPE - 2010 – TRF/4 - Analista de Sistemas - A) Interpretadores são programas que convertem códigos escritos em linguagem de alto nível para programas em linguagem de máquina.

34. (CESPE - 2010 – TRF/4 - Analista de Sistemas - B) Linguagens de alto nível cumprem tarefas mais substanciais com um número menor de comandos, mas exigem programas tradutores denominados compiladores para converter programas em linguagem de alto nível para linguagem de máquina.

35. (CESPE - 2010 – TRF/4 - Analista de Sistemas - C) Um computador pode entender qualquer linguagem de máquina, pois a linguagem de máquina não é definida pelo projeto de hardware do computador.

36. (CESPE - 2010 – TRF/4 - Analista de Sistemas - D) Programadores podem escrever instruções em várias linguagens de programação e todas são entendidas diretamente pelos computadores sem a necessidade de tradução.

37. (CESPE - 2010 – TRF/4 - Analista de Sistemas - E) Softwares escritos em linguagens de máquina são portáveis.

38. (CESPE - 2010 – BASA - Analista de Sistemas) Um interpretador é um programa que lê um código escrito em uma linguagem e o converte em um código equivalente em outra linguagem. No processo de conversão, o interpretador relata a presença de erros no código original.

39. (CESPE - 2010 – INMETRO - Analista de Sistemas) Um interpretador traduz um programa descrito no nível da linguagem para o nível da máquina, enquanto o compilador eleva a máquina ao nível da linguagem, para que o programa execute a partir da fonte.

40. (CESPE - 2011 – ECT - Analista de Sistemas) No programa em linguagem de alto nível, os interpretadores executam os passos definidos para cada instrução e produzem o mesmo resultado que o do programa compilado. Entretanto, a execução de um programa em linguagem de alto nível com o uso de interpretadores é mais lenta que a execução de um programa compilado, uma vez que precisa examinar cada instrução no programa-fonte, à medida que ela ocorre, e desviar para a rotina que executa a instrução.

41. (CESPE - 2009 – TCE/AC - Analista de Sistemas - A) Os programas escritos em linguagem de programação de alto nível precisam ser convertidos em programas de máquina, sendo o linker um tipo de software básico que efetua essa tradução.

42. (CESPE - 2007 – TCU - Analista de Sistemas) Um montador é considerado um software de sistema, responsável pela tradução de uma linguagem de alto nível para uma linguagem de baixo nível (linguagem simbólica).

43. (CESPE - 2008 – FUB - Analista de Sistemas) O montador realiza a tarefa de combinar módulos objetos em um módulo de carga.

44. (CESPE - 2008 – FUB - Analista de Sistemas – C) Para que um programa possa ser executado, seu código de máquina deve estar presente na memória. O ligador é o programa do sistema responsável por transferir o código de máquina de um módulo objeto para a memória e encaminhar o início de sua execução.

45. (CESPE - 2006 – DETRAN - Analista de Sistemas - D) O termo linguagem de máquina é usado para denominar linguagens do tipo assembler ou C++.

GABARITO DOS EXERCÍCIOS COMENTADOS (DIVERSAS BANCAS)

PARADIGMA ESTRUTURADO

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

GABARITO DOS EXERCÍCIOS COMENTADOS (CESPE)

COMPILADORES E INTERPRETADORES

1	2	3	4	5	6	7	8	9	10
E	E	C	C	X	C	E	C	E	E
11	12	13	14	15	16	17	18	19	20
E	C	E	C	E	E	E	E	E	C
21	22	23	24	25	26	27	28	29	30
E	E	E	C	E	A	E	E	C	C
31	32	33	34	35	36	37	38	39	40
C	C	E	C	E	E	E	E	E	C
41	42	43	44	45	46	47	48	49	50
E	E	E	E	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.