Estratégia Concursos

Aula 00 (Profs. Felipe Mathias e Raphael Lacerda)

BNDES (Analista - Análise de Sistemas - Desenvolvimento) Desenvolvimento de Software - 2024 (Pós-Edital)

Autor:
Felipe Mathias, Paolla Ramos

26 de Agosto de 2024

Índice

APRESENTAÇÃO DA AULA

Olá, alunos! Bem-vindos a mais uma aula do curso de Tecnologia de Informação para concursos públicos, no Estratégia Concursos.

Me chamo Felipe Mathias e serei seu professor na aula de hoje. Sou um catarinense de 30 anos, programador *front end* (ex-programador, se preferirem haha) e atuo como professor de cursos de Tecnologia da Informação voltados a concursos há mais de um ano. Assim como você, também vivo a vida de concurseiro, aguardando minha nomeação como Auditor Fiscal da Secretaria de Fazenda de Minas Gerais (SEF-MG), onde figuro no cadastro de reserva. Atualmente, continuo, em paralelo, estudando para concursos aguardando o meu grande sonho – o cargo de Auditor Fiscal da SEF-SC, com especialidade em TI.

Minha aventura no mundo do ensino surgiu de uma vontade interna de atuar como professor – sempre amei explicar as coisas, além de ter certa facilidade em expressar conceitos mais complexos para pessoas que talvez não tenham tanta experiência na área.

Meu objetivo aqui é digerir assuntos, desde os mais simples aos mais complexos, para que qualquer aluno consiga os entender, seja um programador, operador de infraestrutura, ou simplesmente um leigo que resolveu adentrar no mundo dos concursos e se deparou com TI no seu edital.

Gostaria de pedir que **sempre** vejam as questões comentadas durante a aula. Elas trazem conteúdo essencial para o aprendizado, muitas vezes abordando alguns pontos que não foram abordados no conteúdo e são essenciais para a resolução de questões.

Caso tenha alguma dúvida, não tenha receio de entrar em contato comigo nas minhas redes sociais (especialmente no meu Instagram, que deixarei abaixo), ou no fórum de dúvidas que os responderei assim que possível.

Ah, posto bastante coisa interessante de TI direcionada para concursos lá, dá uma olhadinha que algumas coisas podem te interessar. Volta e meio acerto alguma questão de prova por lá ;)

@fe.fiscal

t.me/fefiscal

LÓGICA DE PROGRAMAÇÃO

Conceitos gerais

Você já deve ter se deparado com a palavra "programação", e provavelmente ela cause diversos tipos de receio em você. Mas não tenha medo, coruja, vamos explorar toda a base da programação e da lógica de programação nessa aula, de forma visual, e bem gradual para que você entenda todo o assunto.

Com a programação, podemos criar diversos tipos de aplicativos:

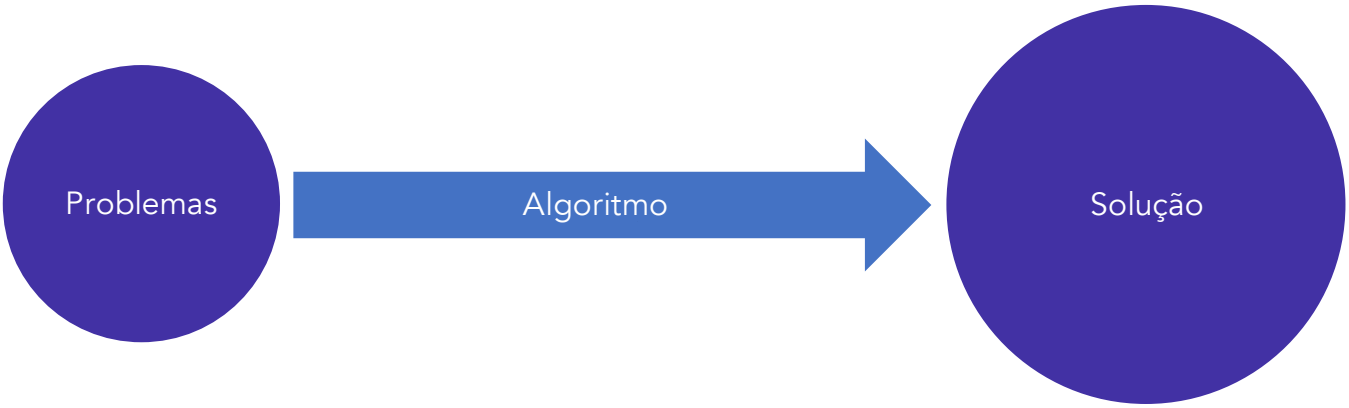
- Aplicativos web (sites interativos)
- Aplicativos de gestão empresarial (ERP)
- Jogos
- Entre muitos outros

Antes de entrarmos na prática da programação propriamente dita, precisamos entender o que é a programação. Um computador, por natureza, é "burro e preguiçoso": ele não sabe o que fazer, e só irá fazer algo a pedido de um terceiro, nunca pela sua própria vontade. A programação computacional vista justamente dizer ao computador o que fazer.

Então, de forma geral, podemos resumir a programação como a prática de escrever códigos para orientar o computador a performar determinadas ações, em determinado momento. Como uma

peça coreografada, o computador irá ler todo o *script*, e executará os comandos nele escritos. Esses comandos são instruções, definições, entre outros que veremos mais à frente.

Essas orientações ao computador são escritas a partir de algoritmos. Um algoritmo é uma sequência finita de instruções bem definidas e não ambíguas que descrevem um processo ou conjunto de operações a serem executadas para resolver um problema específico. Essa instruções são lidas e interpretadas em uma ordem específica, usualmente na ordem em que são escritas no código.



FIQUE ATENTO!

ALGORITMOS SÃO INSTRUÇÕES QUE VISAM DAR UMA SOLUÇÃO A UM PROBLEMA

(Inédita/Prof. Felipe Mathias) Julgue o item a seguir.

Na programação, um algoritmo é uma sequência lógica de passos, modeladas em torno de um problema - visando dar uma solução a ele, através de passos sequenciais.

Comentários:

Perfeito! Com um algoritmo elencamos uma sequência lógica de passos, que são executados sequencialmente, visando solucionar determinado problema através da execução de uma tarefa. (Gabarito: Certo)

Saindo do mundo computacional, imagine que você tem uma decisão a tomar - pense que sua namorada (o) está exigindo veementemente que você a (o) leve no cinema para assistir Duna 2. Diante desse cenário, você concorda, afinal é um filme excelente e muito aclamado, e decide ir ao cinema. O objetivo final é assistir ao filme, então você faz uma sequência de passos:

- Veste uma roupa cheirosa e limpa
- Passa para pegar sua companhia
- Escolhe uma sessão
- Compra os ingressos para o filme
- Assiste ao filme

Então, um algoritmo para a situação narrada acima pode ser:

Pseudocódigo

```
Início
    preparação() {
        colocar a roupa,
        pegar a companhia,
        ir ao cinema
    }

    filme() {
        escolher sessão,
        comprar ingresso,
        assistir ao filme
    }
Fim
```

Essa caixa acima será a nossa IDE - nossa plataforma de programação que iremos usar durante a aula. Ela é separada em três partes - um cabeçalho, que indica a linguagem utilizada, um corpo que apresenta um bloco de código, e, por fim, um terminal (ausente no exemplo acima) que irá mostrar o resultado da execução dos nossos comandos. Os termos "Início" e "Fim" indicam o início e o fim de um programa - irei omiti-los durante a aula para não tornarmos os códigos desnecessariamente longos.

E pronto, você cumpriu o seu objetivo - que era assistir ao filme no cinema. Porém, no meio desse caminho podem existir situações que impedem a consecução do objetivo, como o fato do filme não estar mais em exibição, ou estar com uma sessão lotada. Na computação, a programação visa, através dos algoritmos, justamente dar instruções gerais e alternativas, para lidar com os percalços, visando chegar num objetivo aceitável.

Vamos fazer mais um algoritmo, para orientar você a como aprender com essa aula?

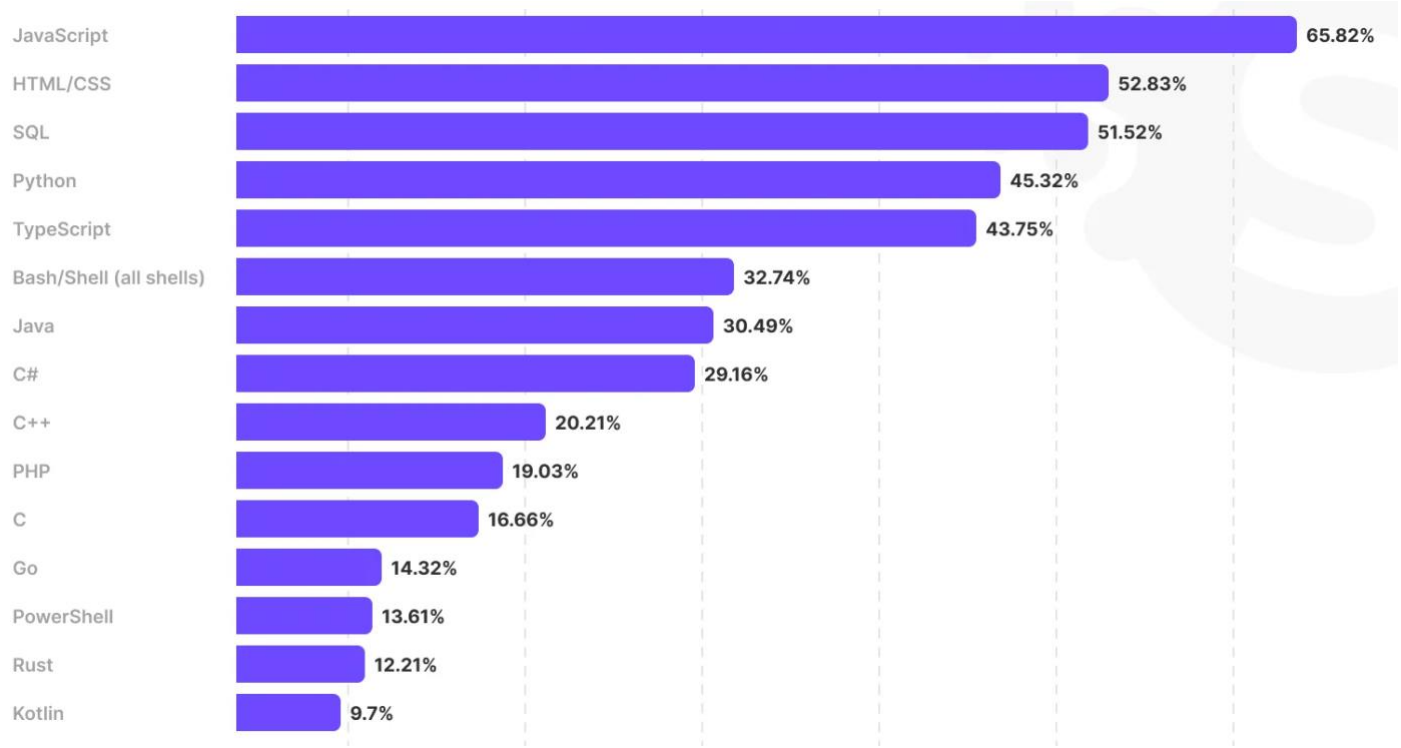
Pseudocódigo

```
Início
    estudo() {
        ler teoria,
        ler questões comentadas,
        fazer exercícios do PDF
    }
Fim
```

Linguagens de Programação

Uma linguagem de programação é um conjunto de regras, símbolos e convenções que permitem que programadores escrevam instruções que um computador pode entender e executar. Essas instruções, chamadas de código-fonte, são escritas em um formato compreensível para humanos, mas devem ser traduzidas para uma forma que o computador possa entender e executar, geralmente na forma de código de máquina.

Veja uma lista com as linguagens mais populares, em 2024:



OBS: Essa lista pode variar conforme a forma de análise dos dados

Podemos dividir as linguagens de programação conforme alguns grupos distintos, dependendo doque estamos analisando. Por exemplo, podemos classificar:

- Nível de Abstração:
  - Linguagens de baixo nível: São mais próximas da linguagem de máquina e fornecem um controle direto sobre o hardware do computador. Exemplos incluem Assembly.
  - Linguagens de alto nível: São mais abstratas e fornecem construções mais poderosas e expressivas. Exemplos incluem Python, Java, C++.
- Paradigma de Programação:
  - Imperativa: As instruções são executadas sequencialmente, alterando o estado do programa através de atribuições. Exemplos incluem C, Fortran.

  - Orientada a Objetos: Os programas são organizados em torno de objetos que podem conter dados e métodos. Exemplos incluem Java, C++.
  - Funcional: Os programas são construídos com funções puras que evitam efeitos colaterais. Exemplos incluem Haskell, Lisp.
  - Lógica: Os programas são construídos em torno de regras lógicas e inferências. Exemplos incluem Prolog.
- Propósito:
  - Geral: São usadas para uma ampla variedade de aplicações. Exemplos incluem Python, C++.
  - Específica de Domínio: São otimizadas para resolver problemas em um domínio específico. Exemplos incluem SQL para bancos de dados, MATLAB para computação numérica.
- Compilação:
  - Compiladas: São traduzidas integralmente para código de máquina antes da execução. Exemplos incluem C, C++.
  - Interpretadas: São executadas linha por linha por um interpretador. Exemplos incluem Python, JavaScript.

Para a aula de hoje, como não estamos a ponto de implementar determinada linguagem ainda, usaremos um conceito chamado de pseudolinguagem, ou português estruturado. Uma pseudolinguagem é uma forma simplificada de expressar algoritmos e lógica de programação, utilizando uma mistura de elementos de linguagens de programação reais e linguagem natural. Ela é usada principalmente para explicar conceitos de programação, projetar algoritmos e fornecer uma representação clara e compreensível de um problema ou solução, sem se preocupar com a sintaxe específica de uma linguagem de programação real.

Em uma pseudolinguagem, os algoritmos são descritos de maneira abstrata, usando termos genéricos como "se", "enquanto", "para", "variável", etc., em vez de sintaxe específica de uma linguagem de programação. Isso permite que os programadores expressem suas ideias de forma clara e concisa, sem se prenderem a uma linguagem de programação específica.

(CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

O pseudocódigo consiste em um texto estruturado com comandos escritos em linguagem humana, no qual se apoia a criação dos algoritmos computacionais.

Comentários:

Exatamente. O pseudocódigo é um texto estruturado, que não segue regras e padrões específicos, usado para descrever a lógica de um algoritmo sem se preocupar com a implementação específica de uma linguagem. (Gabarito: Certo)

Comentários

Muitas linguagens de programação trazem instruções complexas, ou tão personalizáveis que sua interpretação por outros desenvolvedores acaba sendo dificultada. Nesses casos, as linguagens em geral contam com uma ferramenta de auxílio para a interpretação: os com **comentários**.

Comentários são blocos de código não interpretados pela ferramenta que fará a leitura do código. Ele não possui um valor lógico, interpretável, somente um conjunto de instruções, observações e detalhamentos que o próprio desenvolvedor utilizará. Essa instrução pode indicar algum padrão de codificação que deve ser usado, o funcionamento de uma função ou até mesmo explicar qual variável está sendo chamada.

Para identificar um comentário, usa-se um símbolo no início do código. Esse símbolo deve ser escrito para cada linha do código - ou seja, se pularmos uma linha, temos de escrever o símbolo novamente, já que as simbologias, salvo exceções específicas de códigos, abrangem somente a linha em que o comentário se deu início.

Símbolos que são usados para os comentários diferente entre linguagens - podemos ter barras duplas `//`, uma cerquilha `#`, ou até mesmo uma estrutura especial para comentários em múltiplas linhas, como no JavaScript, que implementa comentários entre os símbolos `/*` e `*/`, ou a notação usada em CSS, que coloca os comentários entre `<!--` e `-->`.

Como pseudocódigos são literais e não exigem muita "mágica" para serem interpretados, não usaremos muitos comentários na aula de hoje - mas, quando aparecerem, usaremos a notação das barras duplas `//`. Veja um exemplo:

Pseudocódigo

```
Início
    estudo() {
        ler teoria,
        ler questões comentadas,
        fazer exercícios do PDF //esse é um comentário de código
    }
Fim
```

Variáveis e Constantes

Conceitos gerais

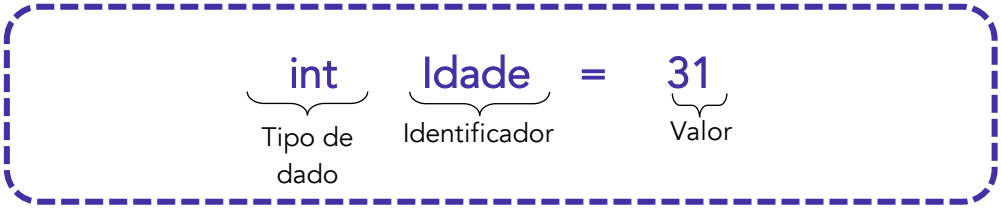
Quando estamos programando, pode ser necessário armazenar algum tipo de valor, de dado, no conjunto de instruções que estamos passando ao computador. Por exemplo, pode ser útil definir pi = 3.1415, para que não tenhamos que digitar o número toda vez que formos usá-lo, ou podemos ter dados que serão alterados durante a execução do código, mas que precisam referenciar um mesmo objeto - por exemplo, uma mudança de estado, onde a pessoa está "com fome" e depois "saciado".

A programação tem um tipo de estrutura específica para lidar com isso: são as variáveis. As variáveis representam locais de armazenamento na memória do computador onde valores podem ser guardados e manipulados durante a execução de um programa. Cada variável possui um nome único que a identifica e um tipo de dado que determina o tipo de informação que pode ser armazenada nela.

Uma variável é composta de três partes distintas:

- Identificador (ou nome)
- Tipo de dado
- Valor

Então, por exemplo, podemos ter:



O identificador será responsável por, justamente, identificar uma variável de forma inequívoca. É, basicamente, o nome da variável. Então, por exemplo, podemos criar uma variável "Fome", que

irá comportar o valor sobre o estado de fome atual, ou a variável "Cor_Cabelo", que irá receber valores para cor de cabelo.

O tipo de dado refere-se ao formato do valor que será alocado à variável. Cada linguagem específica trabalha com tipos de dados diferentes, mas, de forma geral, podemos definir alguns tipos de dados básicos - e que usaremos hoje:

- Caracteres - string → um conjunto de caracteres alfanuméricos. Exemplo: "Você será aprovado, Coruja!"
- Números inteiros - int → números do tipo inteiro. Exemplo: 4, 9, 13, 293
- Números decimais - double ou float → números de precisão decimal. Exemplo: 4.91
- Lógico - boolean → valores lógicos. Exemplo.: Verdadeiro, Falso.

Então, uma variável do tipo *string* comportará um conjunto de caracteres, usualmente formando algum tipo de texto. Especificamente nesse caso, as *strings* são delimitadas pelo uso de aspas - podendo ser aspas simples `´string´` ou aspas duplas `"string"`. Isso é um ponto muito importante, e cobrado muito pelas bancas, veja os seguintes valores:

- 5
- "5"

Se formos fazer uma comparação restrita, isso é, verificando tanto o valor quanto o tipo do dado anotado, veremos que os dois valores são diferentes. Portanto, 5 ≠ "5". Isso, pois a notação 5 indica que estamos trabalhando com um tipo numérico inteiro, e a notação "5" com uma string de caracteres.

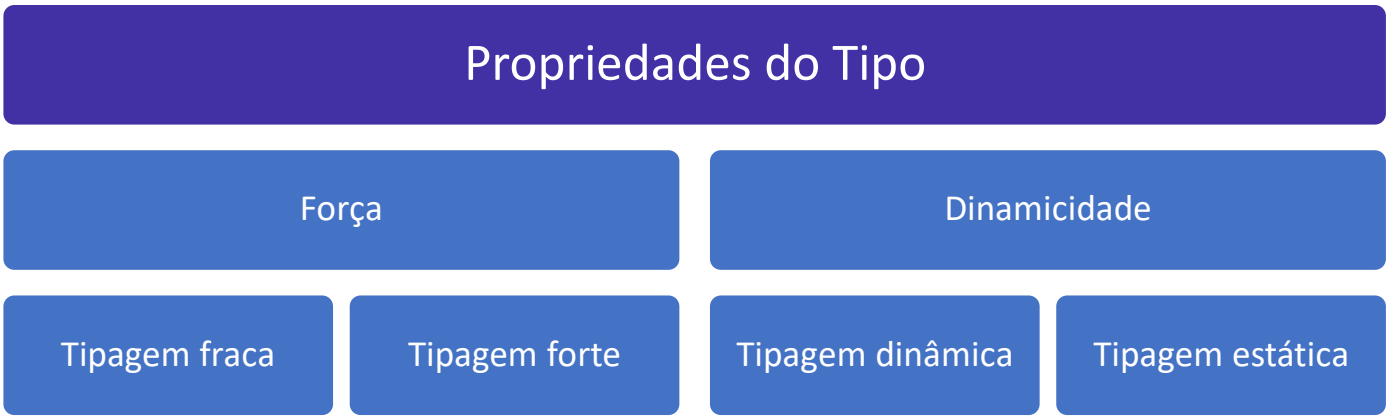
Outro ponto que é importante destacar sobre os tipos de dado é a tipagem. A tipagem diz respeita à forma como as linguagens lidam com as definições de tipos de dado. Cada linguagem de programação pode ter propriedades diferentes quanto à força da tipagem e quanto à dinamicidade da tipagem. A força da tipagem diz respeito à rigidez ou flexibilidade de uma linguagem com relação aos tipos de dados de uma variável.

No aspecto de força, uma linguagem pode ser:

- Fortemente tipada: o escopo do tipo de dado não é flexível, não permitindo, portanto, operações entre tipos de dados distintos - assim, é exigida uma transformação explícita dos dados ao mesmo tipo antes de operações, caso contrário a operação apontará erros. Exemplo de linguagens fortemente tipadas incluem Python e Jave.
- Fracamente tipada: temos um escopo flexível para cada variável, permitindo operações entre diferentes tipos sem a necessidade de uma transformação de tipo de dado explícita, já que a linguagem irá fazer a conversão implicitamente. Exemplos de linguagem incluem JavaScript e PHP.

Já a dinamicidade da tipagem diz respeito sobre a capacidade de uma linguagem determinar o tipo de variável durante a execução de um programa. Uma linguagem pode ser também de dois tipos, quanto à dinamicidade:

- Tipagem dinâmica: o tipo é definido com base no valor que está sendo atribuído, sem necessidade de indicação expressa do tipo de dado. Exemplo: JavaScript e Python.
- Tipagem estática: a linguagem é incapaz de definir, por ela mesma, o tipo de dado apenas com base no valor - nesse caso, é necessária uma declaração explícita do tipo de dado junto da variável. Exemplo: Java, C++.



(CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

Os dados de um algoritmo devem ser definidos por tipos para que seus conteúdos possam ser submetidos a operações corretas, inerentes a cada tipo de dado.

Comentários:

Apesar da definição se dar ora de forma expressa, ora de forma implícita (na tipagem dinâmica), ele **sempre** deve ser definido, de forma que o interpretador do código consiga fazer as operações de forma correta íntegra. Correta a afirmativa. (Gabarito: Correto)

Para o pseudocódigo usado na aula de hoje, trabalharemos com tipagem fraca e tipagem dinâmica. Deixando claro que os tipos não são excludentes, então podemos ter uma linguagem fraca e dinâmica, fraca e estática, assim por diante.

E, por fim, temos o valor da variável. É nele que declaramos o que a variável comportará, seu conteúdo. Esse valor, por estarmos tratando de uma variável, é, justamente, variável rs. Ele poderá ser modificado ao longo da execução de um código sem nenhum problema maior. Em

contraponto a isso, temos as constantes. Valores armazenados em uma constante não podem ser alterados depois de declarados e atribuídos.

VARIÁVEL → VALOR PODE SER ALTERADO

CONSTANTE → VALOR NÃO PODE SER ALTERADO

Declarando variáveis

Agora que você sabe o que é uma variável, precisamos aprender a declará-la. Já fizemos isso lá em cima, quando dei o exemplo de uma variável idade a vocês. Cada linguagem tem uma forma diferente, veja como funciona em diferentes linguagens:

JavaScript
```
let melhorCurso = "Estratégia"
```

Python
```
melhorCurso = "Estratégia"
```

R
```
melhorCurso <- "Estratégia"
```

Java
```
String melhorCurso = "Estratégia"
```

Veja que é um processo simples - definimos o nome, o valor e, quando necessário, atribuímos o tipo de dado à relação.

Agora, uma pergunta: essas variáveis podem ser usadas em todo o programa?

A resposta é: depende. Isso pois as variáveis têm uma propriedade chamada de escopo. O escopo define onde poderemos usar nossa variável dentro do programa, do código da aplicação. De forma geral, temos dois tipos de escopo:

- Escopo global: as variáveis podem ser acessadas (usadas) em qualquer ponto do código, seja fora ou dentro um outro bloco de código interno, como funções, métodos, classes etc.
- Escopo local: são acessíveis somente no contexto em que foram criadas. Por exemplo, uma variável criada dentro de uma classe, só é utilizada dentro dessa classe.

Escopo de uma variável

Global

Local

Nessa aula, para declararmos variáveis, iremos usar um padrão utilizado por linguagens de tipagem estáticas, similar ao usado em Java. O padrão será:

```
[Tipo de dado] identificação = valor
```

Vamos declarar umas variáveis?

Pseudocódigo

```
String nomeCurso = "Desenvolvimento de sistemas";
int numeroInteiro = 9;
double numeroDecimal = 9.31;
boolean condicao = true;
```

Atribuição de valores

Na seção anterior, fizemos declarações de variáveis - que consiste em definir a variável e seu tipo de dado suportado. Porém, eu "omiti" uma informação de você: a declaração de uma variável envolve apenas a sua criação, não a alocação de um valor a esse espaço de memória. Para alocarmos um valor na variável, estamos fazendo o processo chamado de atribuição de valores.

Apesar dessa diferenciação entre declaração e alocação de valores, é prática comum que ambas sejam feitas de uma só vez, já que isso otimiza o espaço do código, reduzindo as linhas e otimizando o programa. Ainda assim, após determinada declaração de uma variável, é possível fazer novas atribuições de valor a ela, já que ela é, justamente, variável.

Essa atribuição pode ocorrer de duas formas distintas: através de um valor, ou através de uma referência. Quando uma variável é atribuída por valor, o que é atribuído é uma cópia do valor. Isso significa que uma cópia do valor é feita e atribuída a outra variável. Modificar a variável original não afetará a cópia e vice-versa. Tipos primitivos, como inteiros e caracteres, são geralmente atribuídos por valor.

Já quando uma variável é atribuída por referência, o que é copiado é a referência (ou endereço de memória) para o valor, não o valor em si. Isso significa que ambas as variáveis agora apontam

para o mesmo objeto na memória. Modificar o objeto através de uma variável afetará o objeto acessado pela outra variável, pois ambas apontam para a mesma área de memória.

Formas de atribuição

- Por valor
- Por referência

Para ficar mais fácil de visualizar a diferença, vou trazer uma tabela com comparações.

Aspecto	Atribuição por Valor	Atribuição por Referência
O que é copiado	O próprio valor	A referência para o valor original
Impacto de modificação	A modificação da variável original não afeta a variável	A modificação da variável original modifica também a variável atribuída por referência
Memória	Exige mais espaço de memória, já que os dados são copiados	Exige menos espaço de memória, já que é feita somente uma referência
Exmplo	a = 3; b = 5;	a = b; c = d

(VUNESP/EPC/S2023) Considere que, ao se chamar uma função: a) foram passados os valores de variáveis para ela; b) o valor de cada variável na função chamadora é copiado nas variáveis fictícias correspondentes da função chamada; c) as alterações feitas nas variáveis fictícias na função chamada não têm efeito nos valores das variáveis reais na função chamadora.

Esse método é conhecido como chamada

a) cruzada.
b) exclusiva.
c) reversa.
d) por valor.
e) por referência.

Comentários:

O método que (b) copia os valores, e (c) cujas alterações não impactam a variável real, é chamado de método de atribuição por valor. (Gabarito: Letra D)

Ordem de Leitura

Temos três tipos de estruturas num código, que delimitam a ordem de leitura das instruções:

- Estrutura sequencial: é a estrutura geral do código, que exige uma leitura sequencial do código, na ordem em que ele aparece.
- Estrutura de seleção: são estabelecidas por sintaxes condicionais, e delimitam que apenas uma das opções deve ser escolhida. Exemplo dessa estrutura são os blocos `Se...então`.
- Estruturas de iteração: define um bloco de código que deve ser repetido enquanto determinada condição for obedecida. Exemplo dessa estrutura são os blocos `Enquanto`.

Quando estamos encarando a estrutura sequencial, determinada linha de código não tem conhecimento de nada que está escrito após ela - por isso, não podemos acessar o valor de variáveis antes de declará-las. Exceções a isso são linguagens que permitem *hoisting*.

O *hoisting* é uma ferramenta que empurra, automaticamente, toda variável para o topo do código, é como se uma leitura prévia fosse feita varrendo o código e, encontrando uma variável, ela é jogada para o topo do código para ser lida antes de tudo.

Apenas nesses casos é que as variáveis poderão ser utilizadas e lidas antes de serem declaradas. Esse comportamento é específico de uma linguagem - o JavaScript (e suas variantes, como o TypeScript). Na aula de hoje, usaremos a abordagem tradicional, sem *hoisting*, que é o mais recomendado para evitar erros de compilação.

> Compilação é o processo pelo qual um programa escrito em uma linguagem de programação de alto nível é traduzido para um formato executável ou código de máquina entendido pelo computador.

Ah, mais um comentário! Você deve ter percebido o uso do ponto e vírgula `;`. Essa é uma simbologia utilizada para delimitar o fim de uma instrução. O interpretador do código iniciará a ler uma linha e considerará tudo de forma conjunta, até a delimitação do fim da linha. Essa simbologia, a depender da linguagem de implementação, não é obrigatória.

Outro ponto importante na leitura e interpretação de códigos é a endentação - esses pequenos espaços deixados antes de começarmos um código, à esquerda dele. Ela permite uma leitura mais limpa do código, garantindo a correta interpretação da hierarquia entre os diversos objetos de um código - inclusive, em alguns formatos de arquivos de marcação, como YAML, e linguagens de programação, como o Python, a endentação é obrigatória e serve como parâmetro de interpretação do arquivo.

Lembre-se: sempre endente seu código.

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

(Inédita/Prof. Felipe Mathias) Julgue o item abaixo, com base nos conceitos da lógica de programação.

A endentação é um processo essencial na construção de códigos. Além de garantir uma melhor leitura do código, algumas linguagens, como Python, utilizam a endentação como forma de garantir a hierarquia dentro do código.

Comentários:

Perfeito! O processo de endentação ajuda a visualizar melhor a hierarquia dos blocos de comandos, além de ser obrigatória para implementar algumas linguagens, como o Python. (Gabarito: Correto)

Operadores

Você já deve estar acostumado a usar operadores, sejam eles os matemáticos, como soma `+` e diferença `-`, ou até mesmo os comparativos, como o "maior ou igual" `≥`. Na programação, esses operadores também estão presentes e ocupam um papel de destaque - muitas vezes, eles que movem nossas funções.

Temos 3 grupos diferentes de operadores: os operadores matemáticos, os operadores relacionais e os operadores lógicos. Vamos conhecê-los!

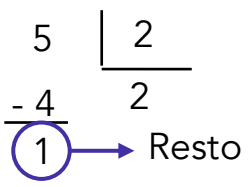
Operadores matemáticos

Os operadores matemáticos são operadores que visam trazer - *adivinhem* - operações matemáticas para o contexto da programação. Com ele, podemos criar expressões matemáticas, equações, e outras operações. Em geral, os operadores refletem a simbologia usada no "mundo real" - então não teremos novidades aqui. Vamos a uma tabela-resumo.

Operador	Símbolo	Exemplo
Adição	+	`4 + 4 == 8`
Subtração	-	`4 - 3 == 1`
Multiplicação	*	`4 * 3 == 12`
Divisão	/	`6 / 4 == 1.5`
Divisão inteira	\	`6 \ 4 == 1`
Exponenciação	^	`3^2 == 9`
Módulo (resto)	%	`3%2 == 1`
Incremento	++	`4++ = 5`
Decremento	--	`4-- = 3`

Um ponto inicial importante: os números decimais devem obedecer à notação com pontuação. Então, escrevemos 1.99 para designar a "famosa" loja que existe em toda cidade, não 1,99. Um outro ponto que pode ter lhe chamado atenção é justamente a operação de módulo - ela retorna o resto de uma divisão.

Por exemplo, na divisão de 5 por 2, o resto dessa operação será 1.



(CEBRASPE/EMPREL/2023)

calc = 5 % 2
imprime(calc)

Assinale a opção que apresenta o resultado da execução do pseudocódigo precedente.

a) 1
b) 2
c) 10
d) 20
e) 25

Comentários:

Questão tranquila, agora que você sabe como o módulo funciona. O cálculo é exatamente o que fizemos acima - portanto, o resto do módulo de 5 por 2 (5%2) é 1. (Gabarito: Letra A)

Agora, eu tenho um desafio a você, qual o resultado da operação abaixo?

```
X = 4 * 3 + 2 % 2
```

Bom, para conseguir responder, você deve saber a ordem de execução dos operadores - e aqui é simples, seguimos a lógica dos operadores matemáticos tradicionais. A ordem se dá pela precedência de operadores - veja a ordem, da maior preferência à menor:

1) Parênteses `( )`
2) Incremento/decremento `++`, `--`
3) Multiplicação, divisão e módulo `*`, `/`, `%`
4) Adição e subtração `+`, `-`

Então, nossa equação pode ser representada da seguinte forma, agrupando as execuções:

```
X = (4 * 3) + (2 % 2)
```

A multiplicação 4*3 é igual a 12, já o módulo, o resto da divisão de 2 por 2 é 0. Portanto, teremos X = 12 + 0 = 0.

"Ok Felipe, para números esse assunto é tranquilo. Mas existem somas de strings?"

Bom, se você não tinha essa dúvida, espero tê-la plantado na sua cabeça. E sim, meu caro aluno, é possível somarmos *strings*. Esse tipo de operação, na computação, recebe um nome especial: concatenação. A concatenação de textos é uma operação que faz, basicamente, a junção de dois blocos de texto. Veja, em código, como ela funciona:

Pseudocódigo

```
String nome = "Felipe";
String sobrenome = "Mathias";
String nomeCompleto = nome + sobrenome;
Escrever(nomeCompleto) //função para mostrar o valor no terminal
```

```
FelipeMathias
```

Acima, usamos uma função chamada `Escrever()`. Irei explicar mais à frente o que é uma função e como elas funcionam, mas já adianto que o propósito dessa função específica é escrever determinado item no console (a parte preta abaixo do bloco de código). No nosso caso, ela "imprime" o valor da variável nomeCompleto.

(CEBRASPE/Pref. Fortaleza/2023) Julgue o item que se segue, relativo a conceitos de avaliação de expressões.

Após o algoritmo a seguir ser executado, os valores das variáveis c e d serão iguais.

real d
real c
d = 6 / -2 + -3 * -2 – 3
escreva (d)
c = 6 / -2 + 3 * 2 – 3
escreva(c)

Comentários:

Para resolvermos as operações, precisamos saber a ordem de execução dos operadores. Separarei por parênteses para que você possa observar melhor, vamos lá.

d = (6 / -2) + (-3 * -2) - 3
d = (-3) + (6) - 3; d = 0

c = (6 / -2) + (3*2) -3
c = (-3) + (6) - 3; c = 0

Como tanto *d* quanto *c* possuem o mesmo valor (0), a afirmativa está correta. (Gabarito: Certo)

Operadores Relacionais

Os operadores relacionais são direcionados a criar relações, comparações entre diferentes elementos. Então, temos operadores como "igual a", "maior que", entre outros. É importante ressaltar que o retorno de um operador relacional sempre será um valor booleano. Ou seja, o valor será verdadeiro, ou falso.

Os operadores são:

Operador	Símbolo	Exemplo
Igualdade	==	4 == 4 ⇒ verdadeiro
Diferença	!=	4 != 4 ⇒ false
Maior que	>	4 > 3 ⇒ verdadeiro
Menor que	<	4 < 3 ⇒ falso
Maior ou igual a	>=	4 >= 4 ⇒ verdadeiro
Menor ou igual a	<=	4 <= 4 ⇒ verdadeiro

Um detalhe que você deve ter atenção é exatamente o primeiro operador da lista, o operador de igualdade. Quando estamos comparando a igualdade de um valor, estamos verificando se ambos expressam o mesmo valor - diferentemente da operação "=" usada na matemática tradicional, como em "2 + 2 = 4". Tenha em mente que estamos fazendo comparações com o `==`, enquanto com o `=` estamos fazendo atribuições, usualmente a variáveis. Vamos fazer umas operações de relação e vermos os resultados no console.

Pseudocódigo

```
boolean comparacao1 = 4 == 4;
boolean comparacao2 = 4 > 4;
boolean comparacao3 = 4 >= 4;
boolean comparacao4 = 4 != 4;
Escrever(comparacao1);
Escrever(comparacao2);
Escrever(comparacao3);
Escrever(comparacao4);
```

```
Verdadeiro
Falso
Verdadeiro
Falso
```

Podemos também, assim como nas operações matemáticas, fazer comparações de caracteres. Aqui, a comparação será lexicográfica. Nela, as letras são tratadas conforme números na tabela de códigos ASCII - basicamente, começamos pela letra A com o maior número, e terminamos com o Z, como maior letra. Além disso, as minúsculas são "maiores" que as maiúsculas.

A comparação será feita caractere a caractere. Se a palavra contiver caracteres iguais, a comparação seguirá até encontrar um caractere de valor diferente, para poder realizar a comparação. Veja alguns exemplos, com os caracteres sendo comparados destacados:

- B > A
- a > A
- Mario > Maria
- Carlos > Carla

Operadores lógicos

Os operadores lógicos são operadores que introduzem as operações de lógica booleana, aquela que vemos junto de raciocínio lógico matemático, dentro do panorama de programações. Na lógica booleana, temos 2 valores: valores verdadeiros, e valores falsos. Dados esses valores, podemos fazer comparações lógicas entre eles, através de 3 operadores:

- E
- OU
- NÃO (ou NEGAÇÃO)

Nos operadores relacionais, a saída será um valor booleano a partir da comparação de dois valores quaisquer. Aqui nos operadores lógicos, a saída também será um valor booleano - porém, estamos comparando outros dois valores booleanos. Para saber a saída, preciso que você tenha um conhecimento básico da lógica matemática. Vou fazer uma breve revisão com você!

E lógico (conjunção)

O comparador E lógico é usado para comparar duas expressões booleanas e resulta em verdadeiro apenas se ambas as expressões forem verdadeiras. Se uma das expressões for falsa, o resultado da operação será falso. Então, podemos ter os seguintes casos, considerando dois valores booleanos distintos:

- Verdadeiro E Verdadeiro → Verdadeiro
- Verdadeiro E Falso → Falso
- Falso E Verdadeiro → Falso
- Falso E Falso → Falso

OU lógico (disjunção)

Com o comparador OU lógico, comparamos duas expressões lógicas e temos como resultado o valor verdadeiro se algum dos dois valores comparados, ou ambos, forem verdadeiros. Ou seja, só retornaremos falso se todos os elementos forem falsos. Então, teremos os seguintes casos:

- Verdadeiro OU Verdadeiro → Verdadeiro
- Verdadeiro OU Falso → Verdadeiro
- Falso OU Verdadeiro → Verdadeiro
- Falso OU Falso → Falso

NÃO lógico (negação)

O último comparador não é bem uma comparação, já que, diferentemente doas anteriores, ele não compara dois valores, ele age em um único valor. O NÃO é responsável por "inverter" o valor de uma variável - ou seja, se ela é verdadeira, a o NÃO irá a transformar em falsa; se for falsa, o NÃO irá a transformar em verdadeira. Então, temos os seguintes casos:

- NÃO Verdadeiro → Falso
- NÃO Falso → Verdadeiro

Essas comparações podem ser analisadas a partir de uma tabela - chamada de tabela verdade. Considere que teremos duas variáveis, p e q, que podem assumir valores verdadeiros (V) ou falsos (F). Suas relações se darão da seguinte forma:

p	q	p E q	p OU q	NÃO p	NÃO q
V	V	V	V	F	F
V	F	F	V	F	V
F	V	F	V	V	F
F	F	F	F	V	V

Vou passar algumas operações e quero que você responda mentalmente o valor *booleano* resultante dessa operação, antes de verificar a resposta. Antes, vamos listar algumas variáveis de forma agrupada - ou seja, declaramos o tipo e uma série de variáveis, e todas as variáveis receberão o mesmo tipo. Vamos lá!

Int a = 3, b = 4, c = 1, d = 5, e = 9, f = 5

1) a > b OU d == f

Resposta: Verdadeiro. A primeira sintaxe é false, já que a (3) não é maior que b (4), mas a segunda sintaxe é verdadeira e, como estamos com o operador OU lógico, precisamos de apenas um dos lados com valor Verdadeiro para termos retorno Verdadeiro.

2) a > b E d == f

Resposta: Falso. Aqui temos as mesmas comparações - mas como o operador é o E lógico, precisamos que ambos os lados sejam verdadeiros para ter o retorno verdadeiro.

3) d >= f E c < e

Resposta: Verdadeiro. Agora temos duas condições verdadeiras - d é igual a f (5 = 5), e c (1) é, de fato, menor que e (9).

4) b%a == c

Resposta: Verdadeiro. O resultado de b%a, ou 4%3 é 1. Como c também é 1, ambos os valores são iguais.

(CEBRASPE/Pref. Fortaleza/2023)

```
real c, b, d;
real x, y;
c = 5;
b = 8;
d = 3;
x = (c < b) ou (b < d) e (c < d);
y = ((c < b) ou (b < d)) e (c < d);
escreva(x);
escreva(y);
```

Com base no algoritmo precedente, julgue o item a seguir, relativo a operadores e expressões.

Após a execução desse algoritmo, os valores das variáveis x e y serão diferentes.

Comentários:

Questão interessante. Temos 5 variáveis (c, b, d, x, y), todas do tipo real (números positivos). As variáveis *c, b* e *d* são declaradas de forma direta, com os valores 5, 8 e 3, e as variáveis X e Y vão receber o valor resultante da operação lógica contida nelas. Um detalhe importante, como

estamos trabalhando com as variáveis *x* e *y* do tipo real, apesar de termos uma comparação lógica, o valor de saída será um número - 0 se Falso, 1 se Verdadeiro. Vamos realizar as duas operações.

x = (c < b) ou (b < d) e (c < d)

Basicamente, estamos fazendo o seguinte: x = (5 < 8) ou (8 < 3) e (5 < 3); x = (V) ou (F) e (F)

Para resolvermos, precisamos saber a ordem de execução das operações lógicas. As linguagens podem variar, mas, de forma geral, temos NÃO → E → OU.

Então, faríamos:

x = (V) ou ((F) e (F)) = (V) ou (F); x = 1 (Verdadeiro).

y = ((c < b) ou (b < d)) e (c < d)

Agora temos um parênteses - e ele sempre terá precedência. Executaremos o lado do "OU" antes e, depois, compararemos com o outro lado da equação.

y = ((V) ou (F)) e (F) = (V) e (F); y = 0 (Falso).

Como x (1) e y (0) são diferentes, a afirmativa está incorreta. (Gabarito: Falso)

Operações Lógicas com Números (Bitwise)

As operações lógicas não se restringem aos valores numéricos - elas podem ser aplicadas também a valores numéricos. Para isso, você deve ter um conhecimento básico de números binários. Esse será um assunto que você verá em mais profundidade em um momento específico, então vou trazer uma breve explicação para que você possa entender as operações.

INTRODUÇÃO AOS NÚMEROS BINÁRIOS

Os números binários são um sistema numérico que utiliza apenas dois dígitos: 0 e 1. Enquanto o sistema decimal, que é o sistema numérico mais comum, utiliza 10 dígitos (de 0 a 9), o sistema binário tem uma abordagem mais direta.

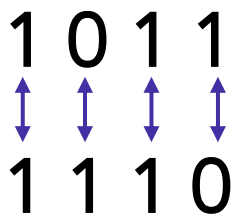
Num sistema binário, o 1 indica que o número está presente, e o 0 indica que não está. O número mais à direita sempre equivale a 2^0 e, a partir dali, subimos uma casa exponencial. Veja como fica, em base decimal, os 8 primeiros bits:

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
128 64 32 16 8 4 2 1

Então, se quisermos escrever o número 3, por exemplo, podemos escrever 11 – que equivale a $2^1 + 2^0 = 2 + 1 = 3$. Outro exemplo – o número 99. Para chegar nele, precisamos "ativar" o 64, 32, 2 e 1 (64 + 32 + 2 + 1 = 99). Então, precisamos do 2^0, 2^1, 2^5 e do 2^6. O número, em binários, ficaria assim:

1100011

Quando comparamos logicamente dois números, fazemos uso de operadores **bitwise**, que comparam nossas variáveis *bit a bit*. Ou seja, se temos um número 1011, e outro 1110 (em bits, obviamente), iremos comparar 1 com 1, 0 com 1, 1 com 1 e 1 com 0.

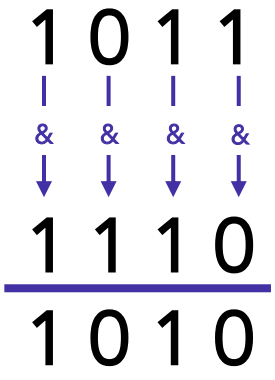


Vamos ver os principais operadores – que derivam de operadores lógicos (E, OU e afins). Encare o 1 como verdadeiro, e 0 como falso, que ficará bem mais tranquilo de entender.

Bitwise AND (&)

As operações ***bitwise and*** (E lógico) usamos o ***e*** lógico do JavaScript: **&** . As comparações apontarão valores 1 apenas quando ambos os binários comparados forem 1. Vamos seguir com o nosso exemplo 1011 e 1110 – que equivalem a 11 e 14, respectivamente. A operação 11 & 14 ocorre da seguinte forma:

- `1 & 1 → 1`
- `0 & 1 → 0`
- `1 & 1 → 1`
- `1 & 0 → 0`

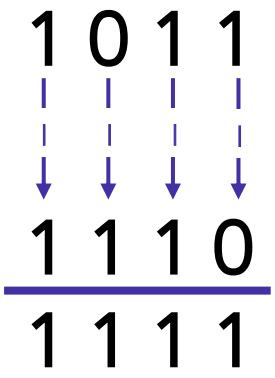


Como resposta, temos 1010 em binários, ou 10, em base numérica.

Bitwiser OR (|)

Para operações ***bitwise or*** (OU lógico) usamos o *ou* lógico do JavaScript: `|`. Aqui, teremos o valor 1 apontado quanto pelo menos um, e qualquer um, dos dois elementos for 1. A operação 11 | 14 ocorre da seguinte forma:

- `1 | 1 → 1`
- `0 | 1 → 1`
- `1 | 1 → 1`
- `1 | 0 → 1`



Como resposta, temos 1111, ou 15.

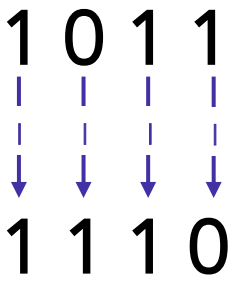
XOr (^)

Para operações ***bitwise xor*** (OU EXCLUSIVO lógico) usamos o *xor* lógico do JavaScript: `^`. O *XOR* corresponde, em lógica, ao *ou exclusivo*, ou ao *ou A ou B*, mas não ambos. Dessa forma, retornaremos 1 quando somente um dos dois números for 1, e 0 quando ambos forem 1 ou 0. Então:

- `1 ^ 1 = 0`
- `1 ^ 0 = 1`
- `0 ^ 0 = 0`

A operação 14 ^ 11 se desenrola da seguinte forma:

- `1 ^ 1 → 0`
- `0 ^ 1 → 1`
- `1 ^ 1 → 0`
- `1 ^ 0 → 1`



Como resposta, temos 0101, ou 5.

Bitwise Not

O operador ***bitwise not*** (negação lógica) usa o símbolo `~` para indicar uma negação do que estamos comparando. Aqui, não estamos mais comparando dois números, e sim negando um número. Mas aqui temos um detalhe: estamos trabalhando com números binários. A negação 5 não é, simplesmente, -5.

O que ocorre com o operador é a inversão total de bits. Usualmente, trabalhamos com espaços de 32 bits. O número 5 seria representado da seguinte forma:

```
00000000000000000000000000000101
```

Porém, é comum que façamos a omissão dos 0s para simplificar o entendimento – já que, em conversões, eles não contribuem em nada. Mas, para negarmos, iremos fazer uma complementar e transformar todos os bits 0 em 1, e 1 em 0:

```
11111111111111111111111111111010
```

Acontece que a negação transforma um número positivo em negativo, nesse caso os binários funcionam um pouco diferente. Aqui, o bit mais significativo (o bit mais à esquerda) é usado como bit de sinal. O bit mais significativo igual a `1` indica um número negativo, e, se for igual a `0`, indica um número positivo.

Quando os números forem indicados como negativos, temos uma mudança – os 0s representam complementos de 2. E, ao contrário de números positivos, não começamos com 2^0, e sim com 2^1.

$$11111111111111111111111111110\mathbf{1}0\mathbf{1}0 = -((\mathbf{2^1} + \mathbf{2^2}) = -6$$

Ok, parece complexo né? Mas tenho uma forma muito mais simples de resolver isso, seguindo uma fórmula infalível:

$$\mathbf{\sim X = -(X + 1)}$$

A negação de qualquer número vai ser igual ao negativo do seu número subsequente. Então:

- ~5 = -(5+1) = -6

- ~0 = -(0+1) = -1
- ~12 = -(12+1) = -13
- ...
- ~14.324 = -(14.324+1) = -14.325

Deslocamento de Bits

A última operação que veremos, que na verdade são 3, representam operações de deslocamento – chamadas de ***bitwise left shift*** (deslocamento de bits para a esquerda), e ***bitwise right shift*** (deslocamento de bits para a direita). Sua simbologia é:

- Bitwise Right Shift = **>>** ;
- Bitwise Left Shift = **<<** ;

Aqui, deslocamos todos os bits para a direita, ou esquerda, preenchendo o bit restante com o bit de sinal – 0 se positivo, 1 se negativo. Como a sintaxe do operador é `X >> n`, podemos, em *X*, definir o número que queremos operar, e em *n* definir a quantidade de casas que iremos deslocar.

Vamos deslocar alguns números:

5 >> 1

- Estamos deslocando 5 (0101) 1 bit para a direita. Iremos excluir o bit mais à direita (que destaquei em vermelho), e preencher o restante com 0s.
- 0101 >> 1 → 0010
- Convertendo para base decimal, temos que 5 >> 1 = 2

5 << 1

- Estamos deslocando 5 (0101) 1 bit para a esquerda. Aqui é mais simples – basta adicionar mais um 0 no final da cadeia de bits.
- 0101 << 1 → 01010
- Convertendo para base decimal, temos que 5 >> 1 = 10

-5 >> 1:

- Em *bitwise right shift* com números negativos, por estarmos trabalhando com números negativos (bit mais significativo igual a 1), o preenchimento é feito com 1s, e não com 0s.
- Lembrem que a negação de um número é igual a ele, somado a um, negativo (~X = -(X+1). Então, para acharmos -5, temos que fazer ~4.
- 4 em binário: 000...0100 (vou omitir os 0s, mas saibam que existem 32 números aqui)
- Negando todos os bits, para achar -5: 111...1011

- Agora temos um número para trabalhar: 11111111111111111111111111111011
- Vamos excluir o elemento mais à direita, e adicionar o 1 no começo.
- 11111111111111111111111111111011 >> 1 → 11111111111111111111111111111101
- Agora, voltando para a base decima, podemos simplesmente aplicar a fórmula. Na base de 2, consideramos os 0s como bits ativos, então teríamos 2 no binário acima. Retornando, basta fazer VALOR = -(X+1) = -(2+1) = -3

Bom, lembrem que eu falei que são 3 operadores, mas só vimos 2 né? O outro operador é uma variação, chamado de zero-fill. O *zero-fill*, como o nome aponta, preencherá *n* casas, à direita ou à esquerda, apenas com 0s.

Nesse caso, temos o zero-fill right shift. Ele possui uma diferença significativa: ele muda o bit mais significativo. não preencheremos mais o número com os bits significativos, e sim sempre com 0. Então, aqui, números negativos se tornarão positivos. Sua notação é dada ao adicionarmos mais um símbolo ao shift, >>> .

5 >>> 1:

- Em números positivos, também temos o mesmo efeito. 5, em binários, é 0101. Iremos deslocar o número à direita, excluindo o último 1, e preenchendo o restante com 0s.
- 0101 >>> 1 → 0010
- Veja que o resultado foi o mesmo, 2, sem diferença.

-5 >>> 1:

- Agora o buraco é mais embaixo. Já temos a notação em 32 bits de -5: 111...1011
- Iremos preencher o campo à esquerda com um 0 – só que agora esse campo que mudaremos é o bit mais significativo. Isso irá transformar nosso número negativo em um número positivo.
- 11111111111111111111111111111011 >> 1 → 01111111111111111111111111111101
- Agora, retornando à base decimal, temos um número gigante rs – 2.147.483.645

Vamos fazer uma questão?

(FGV/CÂMARA DOS DEPUTADOS/2023) Analise as operações bitwise do JavaScript, exibidas a seguir.

4 & 1

7 | 2
~ -5
9 >> 2
9 >>> 1

Os valores de cada uma dessas expressões, na ordem, são:

a) 0; 7; 5; 1; 2.
b) 1; 9; 5; 1; 2.
c) 0; 7; 4; 2; 4.
d) 0; 9; 4; 0; 4.
e) 1; 1; -5; 4; 2.

Comentários:

Apesar da questão ser implementada em linguagem (JavaScript), você não deve ter dificuldades para resolvê-la, já que esse tipo de operação independe de linguagem. Vamos lá!

1. 4 & 1

Primeiro, convertemos 4 e 1 para bits, ficando com 0100 & 0001. Agora, comparamos:

- 0 & 0 = 0
- 1 & 0 = 0
- 0 & 0 = 0
- 0 & 1 = 0

Resultado 0000 = 0

2. 7 | 2

Convertendo-os para binários, ficamos com 0111 e 0010.

- 0 | 0 = 0
- 1 | 0 = 1
- 1 | 1 = 1
- 1 | 0 = 1

Resultado 0111 = 7

3. ~ -5

Vamos aplicar a fórmula, já que é a forma mais prática de resolvermos a questão. Lembrem:

- ~X = - (X+1)
- ~ (-5) = - (-5+1)
- ~ (-5) = - (-4)
- ~ (-5) = 4

Resultado = 4

4. 9 >> 2

Essa envolve um deslocamento bitwise de 2 casas à direita. 9, em binário, é 1001. Deslocando:

- 1001 >> 2 = 0010
- Portanto, temos 0010 = 2

5. 9 >>> 1

Como falei para vocês, operações zero-fill com números positivos não mudam nada. Assim, vamos deslocar 1001 uma casa para a direita, preenchendo com 0s.

- 1001 >> 2 = 0100

Portanto, temos 0100 = 4

Sendo assim, ficamos com 0, 7, 4, 2 e 4 – correta a letra C. (Gabarito: Letra C)

Estruturas condicionais

Lembram do exemplo que lhe trouxe lá no começo da aula, do problema que poderia surgir ao chegar no cinema? Vamos recapitular.

> *"Porém, no meio desse caminho podem existir situações que impedem a consecução do objetivo, como o fato do filme não estar mais em exibição, ou estar com uma sessão lotada."*

Agora, pense logicamente, poderíamos ter uma variável chamada de `filme_disponível`, que recebe um valor booleano `Verdadeiro`, se o filme estiver disponível, ou `Falso`, se ele não estiver disponível. O objetivo do "programa" era assistir ao filme, porém se `filme_disponível` for `Falso`, não poderemos concluir esse objetivo.

Imagine que essa variável seja de fato falso, ou seja, o filme, por algum motivo, não está disponível para ser assistido. E agora, quais as opções? Simplesmente cancelar o "programa"? Fazer outra coisa, ir a um bar, um restaurante? Todas essas situações precisam ser pensadas enquanto estamos elaborando os planos da noite - caso contrário, sua companhia irá ficar irritada, né?

Na computação não é diferente, precisamos pensar em diversas situações diferentes para que um programa lide com elas. Para esses casos, surge uma estrutura basilar em toda a programação - as estruturas condicionais. Temos duas estruturas principais que você precisa saber e entender: o Se...então e o Escolha...caso.

Essas estruturas são chamadas de estruturas de seleção, pois deve-se selecionar um caminho para ser seguido. Vamos entender cada uma das estruturas.

CEBRASPE/CAU BR/2024) Com relação à lógica de programação, julgue o próximo item.

A estrutura de controle IF, que pode ser classificada como do tipo iteração, determina o caminho que o algoritmo deve seguir, de acordo com determinada condição.

Comentários:

Apesar do IF (Se) definir um caminho para o algoritmo seguir, ela não é uma estrutura de iteração, e sim uma estrutura de seleção. (Gabarito: Errado)

Se... então

O Se...então (ou *If... Then*, em inglês) é a estrutura básica das condicionais em programação. Nela, analisamos determinada situação, se ela for verificada, ou seja, se a comparação feita for de valor `Verdadeiro`, executaremos determinado bloco de código. Veja que aqui iremos usar os operadores lógicos e relacionais, então é importante que você tenha os entendido.

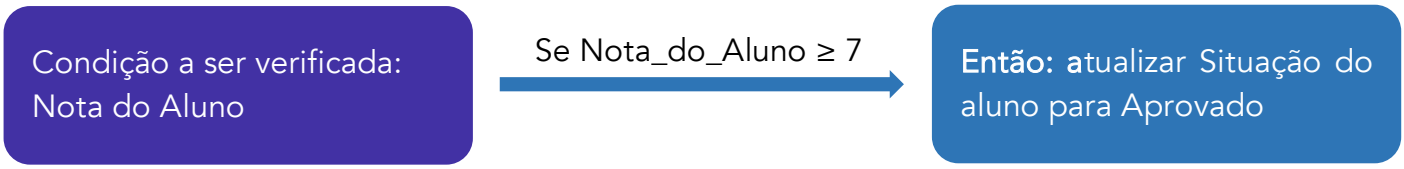
Essa condicional será composta de duas partes:

- **`Se (comparação)`** → é a comparação que será verificada, e só será aceita se o valor em (comparação) for verdadeiro
- **`Então: <bloco de código>`** → é o bloco de código executado se for a

Para que você possa entender melhor, vamos analisar um "sistema de aprovação", que irá verificar se determinado aluno atingiu a nota desejada para ser aprovado. Aqui, vamos trabalhar inicialmente com uma situação - apenas a mudança para "aprovado" da variável situação do aluno, se ele tiver uma nota igual ou superior a 7.

Então, vamos trabalhar com as seguintes variáveis:

- Nota_do_aluno → irá receber um valor decimal (double) com a nota do aluno
- Situação_do_aluno → será iniciada como uma string vazia, e será atualizada para "aprovado", se Nota_do_aluno ≥ 7



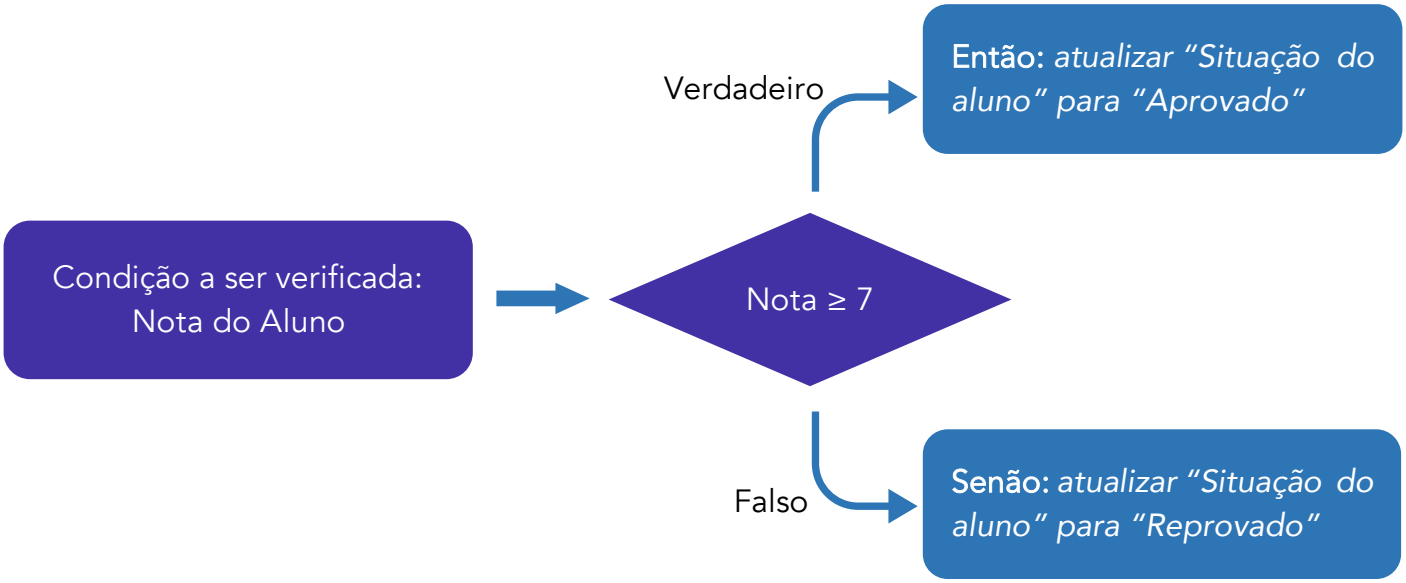
Vamos ver como fica isso em um código:

Pseudocódigo

```
String Nota_do_aluno = 8;
String Situação_do_aluno = "";

Se (Nota_do_aluno >= 7) Então:
    Situação_do_aluno = "Aprovado"; ⟼ Bloco de Código
Fim Se

Escrever(Situação_do_aluno)
```

```
Aprovado
```

Mas você consegue ver que esse programa está falho, né? Apenas indicaremos se o aluno está aprovado. Não conseguimos indicar se o aluno está reprovado, se a nota for menor que 7. Precisamos ajeitar nosso programa. Para isso, complementamos o se...Então com mais um elemento: **senão**. Ele incluirá um bloco de código "residual", que será executado se a condição de verificação for falsa.



Em código, representamos essa estrutura da seguinte forma:

Pseudocódigo

```
String Nota_do_aluno = 6;
String Situação_do_aluno = "";

Se (Nota_do_aluno >= 7) Então:
    Situação_do_aluno = "Aprovado";
Senão:
    Situação_do_aluno = "Reprovado";
Fim Se

Escrever(Situação_do_aluno)
```

```
Reprovado
```

Podemos aprimorar ainda mais o nosso programa, colocando uma série de `Senão`. A interpretação do código será feita a partir do primeiro `Se...Então`, caso o resultado seja `Falso`, pularemos ao próximo `Senão`, até percorrermos todo o código - caso não seja encontrada nenhuma resposta de valor `Verdadeiro`, o bloco de código utilizado corresponderá ao último `Senão`.

Vamos adicionar mais uma camada ao nosso exemplo: teremos os seguintes casos:

- Nota ≥ 7 = Aprovado
- Nota < 7 e ≥ 5 = Recuperação
- Nota < 5 = Reprovado

O nosso código ficará da seguinte forma:

Pseudocódigo

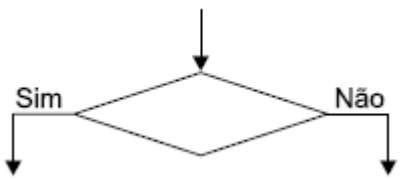
```
String Nota_do_aluno = 6;
String Situação_do_aluno = "";

Se (Nota_do_aluno >= 7) Então:
    Situação_do_aluno = "Aprovado";
Senão (Nota_do_aluno < 7 && Nota_do_aluno >= 5):
    Situação_do_aluno = "Recuperação";
Senão:
    Situação_do_aluno = "Reprovado";
Fim Se

Escrever(Situação_do_aluno)
```

```
Recuperação
```

Veja que temos duas comparações - Nota_do_aluno < 7 e Nota_do_aluno >= 5, como estamos usando o E lógico, para atender a esse Senão precisamos ter um retorno do tipo Verdadeiro nas duas comparações. Podemos criar diversas situações de uso do Senão no meio, tendo um número ilimitado de comparações - porém, como o código varre todas as opções verificando sua implementação, essa pode não ser a melhor alternativa em termos de otimização.

Uma outra possibilidade de uso do Se...Então é o uso de condicionais aninhadas. Teríamos uma condicional dentro de outra condicional - o que também pode prejudicar o desempenho do programa, apesar de muitas vezes ser necessário. Vamos para a próxima estrutura de condicionais que acaba resolvendo alguns dos problemas que lhe falei.

(VUNESP/UNICAMP/2023) Diagramas de blocos podem ser utilizados para expressar lógicas de programação. Considere o seguinte bloco.



Esse bloco expressa, de uma forma geral, a lógica de programação do tipo

a) DO – UNTIL
b) DO – WHILE
c) IF – THEN - ELSE
d) REPEAT – UNTIL
e) SELECT – CASE

Comentários:

O bloco expressa uma escolha, se determinada condição for implementada, vamos para o caminho do Sim, caso contrário, para o caminho do Não - isso indica a estrutura do SE-ENTÃO-SENÃO, ou IF-THEN-ELSE, em inglês. (Gabarito: Letra C)

(CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Se os valores a = 3, b = 4 e c = 8 forem entradas do algoritmo a seguir e o sistema no qual o algoritmo for executado utilizar números decimais com quatro casas de precisão, então a execução do referido algoritmo apresentará em tela o resultado 10.3923.

```
sp = (a + b + c)/2;
ar = sp*(sp - a)*(sp - b)*(sp - c);

se (ar < 0)
   escreva (“Não é possível obter resultado.”);

senão
   escreva (“Resultado: “);
   escreva(raiz_quadrada(ar));
fimse
```

Comentários:

Vamos desenvolver os cálculos.

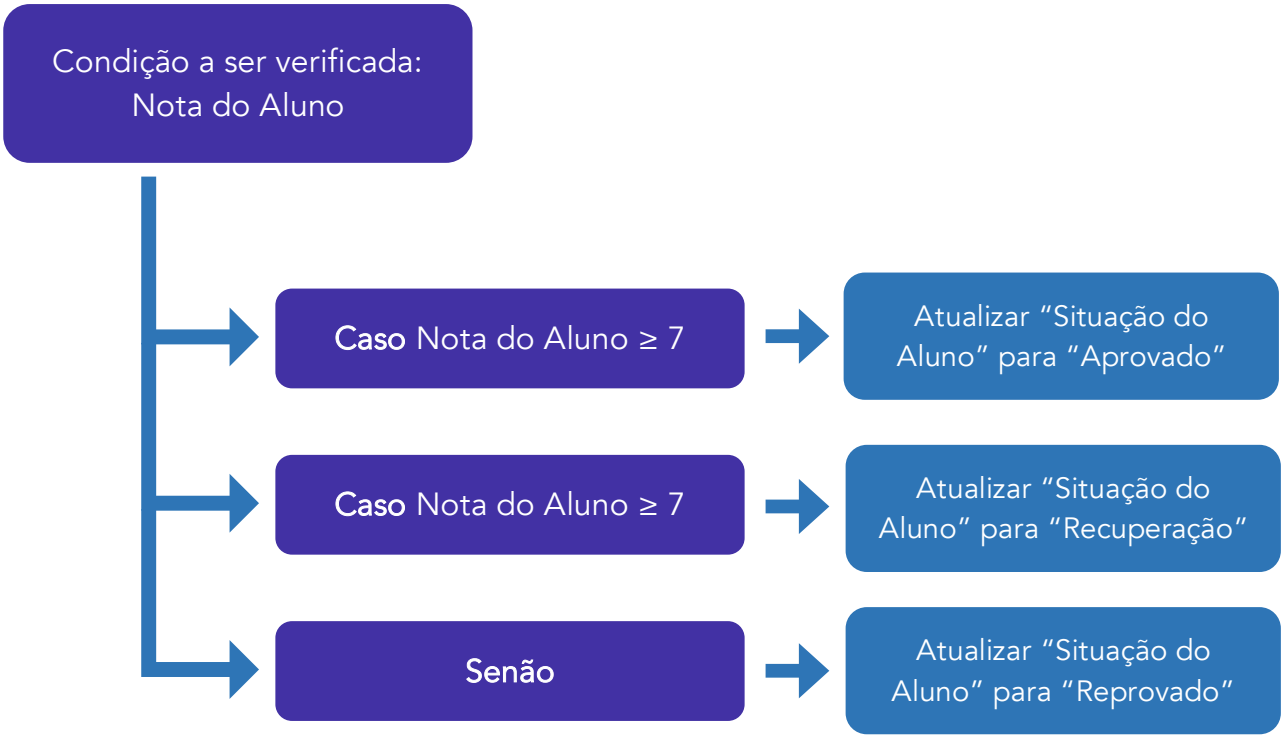
sp = (3 + 4 + 8)/2 = 15/2 = 7.5
ar = 7.5 * (7.5 - 3) * (7.5 - 4) * (7.5 - 8) = -59.0625

Como temos um valor de ar menor que 0 (ar < 0), executaremos o bloco Se, tendo como retorno o texto “Não é possível obter resultado”, e não o valor 10.3923. (Gabarito: Errado)

Escolha...caso

O **`Escolha...caso`** (também chamado de `Switch...case`) é um formato de condicional para seleções múltiplas, adequado quando temos um conjunto de condições serem verificadas. Cada caso irá analisar uma condição diferente, e, se for verificada a condição desse caso específico, ela será implementada. Terminamos o bloco de Escolha...caso com um senão, que será uma opção residual: se nenhuma outra condição tiver a sua veracidade verificada, ativaremos esse bloco.

No mesmo exemplo que terminamos no capítulo anterior, podemos escrever o código com o `Escolha...caso`, veja:



Em código, teremos a seguinte estruturação:

Pseudocódigo

```
String Nota_do_aluno = 6;
String Situação_do_aluno = "";

Escolha
    Caso (Nota_do_Aluno >= 7) Faça:
        Situação_do_aluno = "Aprovado";
    Caso (Nota_do_aluno < 7 && >= 5) Faça:
        Situação_do_aluno = "Recuperação";
    Senão
        Situação_do_aluno = "Reprovado";
Fim Escolha
```

Estruturas de Repetição

Imagine que você precise criar um código que irá Escrever, no terminal, os números de 1 a 1.000.000. De uma forma rudimentar, baseada na "força bruta", você poderia escrever o código da seguinte forma:

Pseudocódigo

```
Escrever(1)
Escrever(2)
Escrever(3)
...
Escrever(999999)
Escrever(1000000)
```

Já pensou o trabalho que seria escrever cada uma dessas linhas? 1 milhão de linhas para um programa relativamente simples. Para resolver esses problemas, assim como implementar mais dinamicidade nos códigos, entram em voga as estruturas de repetição (também chamadas de estruturas de iteração) . Nela, repetiremos determinado bloco de código conforme a condição de verificação. Esse tipo de estrutura pode ser chamada de loop, em inglês, e cada repetição do bloco é chamada de iteração, ou laço.

Temos 4 formas de repetição que iremos abordar:

- Enquanto
- Repita...até
- Faça...enquanto
- Para

Muito foco aqui pois essa é uma parte essencial, já que esse conteúdo é cobrado tanto na parte de lógica de programação, quanto nas próprias linguagens em si, dada a sua importância. Vamos lá!

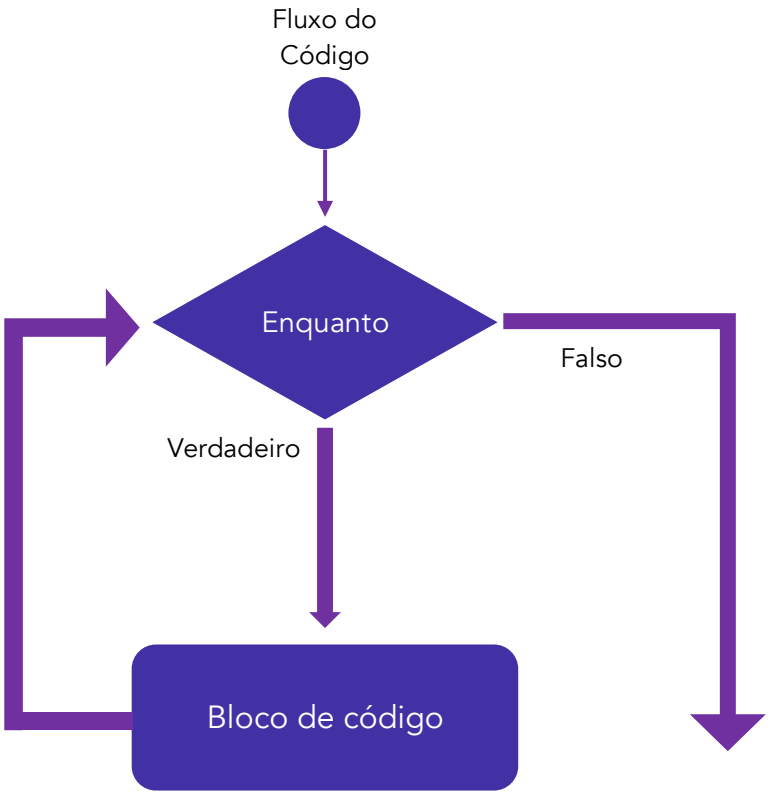
Enquanto

O **`Enquanto`** (ou *While*) é uma estrutura de repetição que irá repetir determinado bloco de código enquanto a condição de verificação tiver valor **`Verdadeiro`**. Aqui, a verificação é feita antes de iniciarmos a execução, se o valor for verdadeiro, seguimos em frente na execução; caso contrário, a repetição é terminada.

A estrutura geral da repetição `Enquanto` é a seguinte:

```
Enquanto (condição a ser verificada) Faça:
     Bloco de código a ser executado
Fim Enquanto
```

Graficamente, podemos interpretar o bloco `Enquanto` da seguinte forma:



Vamos refazer a nossa contagem de 1 a 1.000.000 com a estrutura do `Enquanto`. Para isso, criaremos uma variável para comportar os números, chamada de `contagem`, e, a cada laço do `Enquanto`, iremos fazer um incremento nessa variável, usando o operador `++`. Lembrando que esse código será executado enquanto a variável `contagem` for menor ou igual a 1.000.000. O código ficará da seguinte forma:

Pseudocódigo

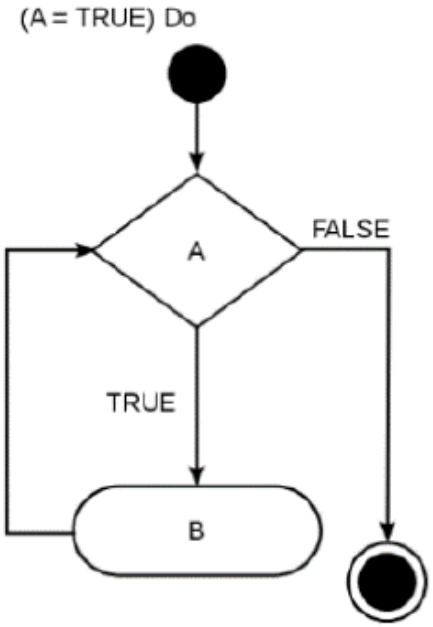
```
int contagem = 1

Enquanto (contagem <= 1000000) Faça:
    Escrever(contagem)
    contagem++
Fim Enquanto
```

Explicando o passo a passo do que acontecerá na execução desse programa:

- Será verificada a condição contagem ≤ 1.000.000
- Se a condição for verdadeira, iremos executar o bloco de código:
  - Escrever o valor atual da variável contagem no terminal
  - Incrementar a contagem (ou seja, somar 1 a ela)
- Termina o laço atual, e repete a execução do comando até que a condição analisada seja considerada falsa

QUESTÃO DE PROVA

(FUNCERN/PREF. SÃO TOMÉ/2023) Dentro da lógica de programação é uma estrutura que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador. A estrutura de repetição básica abaixo está se referindo:



a) do while.
b) for.
c) while.
d) if.

Comentários:

Questão bem interessante. Veja que, inicialmente (em A), estamos verificando se determinada condição é TRUE (Verdadeira) ou FALSE (Falsa). Se for TRUE, executaremos o bloco B, e essa execução se repetirá enquanto a verificação de A continuar verdadeira. Isso caracteriza o bloco condicional Enquanto, o *While*. (Gabarito: Letra C)

(CEBRASPE/PREF. FORTALEZA/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Ao final da execução do algoritmo a seguir, o valor 0 será apresentado.

```
constante A = 50
enquanto (A > 0)
A -= 5;
fim enquanto
escreva (A);
```

Comentários:

Essa questão tem algumas camadas de conhecimento sendo exploradas. Para começar, podemos afirmar errado pois a variável A é do tipo constante, ou seja, não poderá ser alterada. Porém, vamos ignorar esse fato por enquanto e vamos analisar a estrutura do Enquanto.

Temos uma condição A > 0, ou seja, enquanto A for maior que 0, iremos executar um bloco de código. O bloco de código é a operação `A -= 5`, que é uma operação de atribuição: cada vez que ela for chamada, retiraremos 5 unidades do valor de A. Portanto, iríamos reduzir o valor de 5 em 5 - teríamos 50, 45, 40... 0.

Veja que, apesar da condição ser A > 0, quando tivermos o valor de A = 5, a condição ainda será verdadeira e executaremos a operação `A -= 5`, resultando em A = 0. Nesse momento, como A possui o valor 0, a condição não será mais estabelecida e não executaremos mais o código - mas o valor do A que seria impresso realmente seria 0. Note a importância da ordem de leitura do código.

Porém, novamente, como estamos tratando de uma constante, não podemos alterar seu valor de forma externa - então o valor da operação escreva (A) (que é similar ao Escrever(A), que estamos fazendo até agora) será 50. (Gabarito: Errado)

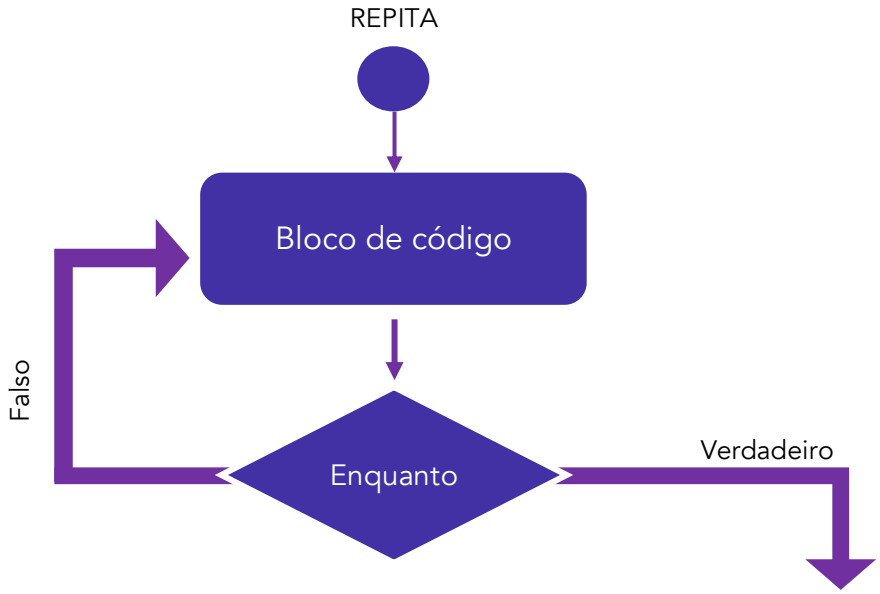
Repita...até

A segunda forma de estrutura de repetição, o **`Repita...até`**, que também pode aparecer como **`Faça...enquanto`**, é uma estrutura muito similar ao enquanto - porém, a verificação da condição é feita após a execução do bloco de código. Então, aqui, mesmo que a condição não seja satisfeita, iremos executar o bloco de código ao menos uma vez. Além disso, aqui o bloco de código é executado enquanto a condição for falsa. Se a condição for implementada, verdadeira, sairemos do bloco.

A estrutura básica dessa estrutura de repetição é:

```
Repita:
     Bloco de Código
Até (Condição a ser verificada)
Fim Repita
```

Graficamente, podemos fazer a seguinte representação:



Vamos pegar o mesmo código que usamos anteriormente, no Enquanto, e aplicá-lo aqui.

Pseudocódigo

```
int contagem = 1
Repita:
    Escrever(contagem)
    contagem++
Até (contagem = 1000000)
Fim Repita
```

Agora eu lhe pergunto, qual será o valor que será impresso no terminal na execução do último laço desse *loop*?

Se você respondeu 1.000.001, parabéns! Você entendeu a ordem de interpretação de um código. Veja bem, quando tivermos o valor 1.000.000, a verificação `contagem ≤ 1.000.000` ainda será verdadeira - o que quer dizer que iremos executar mais uma vez o código. Dessa vez, o valor de contagem será levado a 1.000.001, e, na verificação da condição, não teremos mais o valor `Verdadeiro` e iremos romper o *loop*.

(FGV/TCE SP/2023) Marta está definindo um algoritmo para descrever um menu de funções do sistema, apresentando as opções baseadas em números, seguido da leitura da opção, com a saída ocorrendo após a digitação do número zero.

Para gerenciar o fluxo de execução, que envolve a exibição do menu e leitura da opção, repetindo-se até que seja digitada a opção zero, Marta deve utilizar a estrutura de controle:

a) enquanto - faça;
b) se - então;
c) repita - até;
d) para - faça;
e) se - então - senão.

Comentários:

Questão interessante, que cobra uma parte conceitual da lógica de programação.

O fluxo de execução de um código é a ordem com que o interpretador lê esse código e executa operações. Estamos seguindo um fluxo de execução de bloco de código → verificação de condição, até que determinada condição seja verificada como verdadeira. Essa estrutura se caracteriza como uma repetição do tipo Repita...Até. (Gabarito: Letra C)

Faça...enquanto

O **`Faça...enquanto`**, ou `Do...while` é uma estrutura de repetição muito semelhante ao Repita...até, mas com algumas diferenças "grandes". Ele também executará, obrigatoriamente, um bloco de código - e irá executá-lo repetidas vezes, desde que a condição de verificação seja verdadeira.

Veja que já temos uma primeira diferença - o Faça...enquanto irá executar as instruções enquanto a condição for verdadeira. Uma segunda diferença é o momento da verificação da condição: apesar de executar um bloco independentemente do que ocorra, o Faça...enquanto verifica a condição antes de executar o bloco uma vez, enquanto o Repita...até verifica depois.
Então, resumindo:

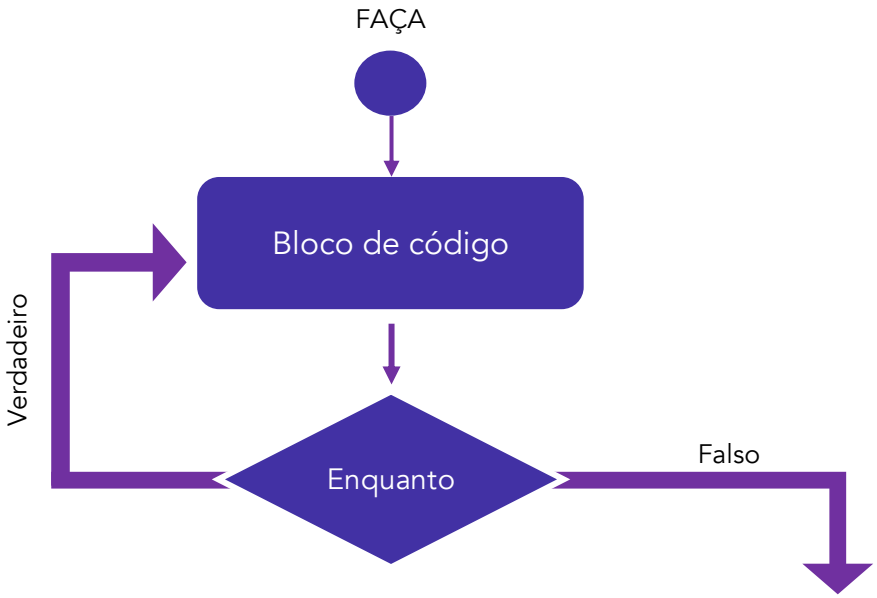
Aspecto	Repita...até	Faça...enquanto
Tipo de Estrutura	Repetição (loop)	Repetição (loop)
Executa o bloco ao menos uma vez	Sim	Sim
Verificação da condição	Depois de executar o primeiro bloco	Antes de executar o primeiro bloco
Saída do Loop	Quando a condição for FALSA	Quando a condição for VERDADEIRA

A estrutura para o Faça...enquanto é a seguinte:

```
Repita:
     Bloco de Código
Até (Condição a ser verificada)
```

```
Fim Repita
```

Graficamente, podemos representar a estrutura da seguinte forma:



Para

A estrutura de repetição Para é uma estrutura destinada a uso quando você já sabe quantas iterações, quantos laços do *loop* você quer que sejam realizados. No Para, definimos um valor inicial, um valor final e quanto esse valor será incrementado (ou decrementado) a cada laço - servindo como forma de controle para quantas iterações teremos.

A estrutura geral do comando é:

```
Para (a; b; c) Faça
     Bloco de Código
Fim Para
```

Na sintaxe, os valores *a*, *b* e *c* correspondem a:

- a → valor de início
- b → valor final
- c → incremento ou decremento

Alternativamente, esse bloco pode aparecer como `Para i de x até z`, onde *i* indica a variável da contagem, *x* o valor inicial e *z* o valor final - nesse caso, sempre subiremos uma unidade a cada laço. Vamos a alguns exemplos, para que você entenda bem a sistemática do Para.

Exemplo 1 - Para (y = 0; y < 5; y++)

- a → y = 0
- b → y < 5
- c → y++

Veja que começamos o valor inicial declarando uma variável y, com valor 0. É prática comum declararmos uma variável qualquer para isso, e essa variável terá escopo local - ou seja, só será aplicável ao contexto dessa estrutura. Em seguida, definimos o valor limite, y < 5, e definimos a forma de incremento - y++ (ou seja, uma unidade por laço).

Então, a cada laço, a variável y terá seu valor alterado. A execução se dará da seguinte forma:

- Laço 1 → y = 0;
- Laço 2 → y = 1;
- Laço 3 → y = 2;
- Laço 4 → y = 3;
- Laço 5 → y = 4;

Então, com essa sintaxe Para (y = 0; y < 5; y++) estamos passando para o código que queremos executar o bloco de código definido por 5 vezes.

Exemplo 2 - Para (y = 5; y > 0; y--)

- a → y = 5
- b → y > 0
- c → y--

Praticamente a mesma sintaxe, mas estamos fazendo um decremento da variável Y. Teremos, também, a execução de 5 laços, veja:

- Laço 1 → y = 5;
- Laço 2 → y = 4;
- Laço 3 → y = 3;
- Laço 4 → y = 2;
- Laço 5 → y = 1;

O uso do Para passa a ser interessante quando o valor da variável local declarada é útil para o contexto do *loop*. Por exemplo, na nossa contagem de 1 a 1.000.000, podemos fazer a execução do nosso programa numa forma bem reduzida. Aqui, utilizarei a variável local *i*, que iniciará em 0 e irá até 1.000.000, incrementando em 1 unidade a cada laço. Utilizarei essa mesma variável local

no comando Escrever(), dentro do bloco de código, para que esse valor seja impresso. Veja como diminuímos o tamanho do código:

Pseudocódigo

```
Para (i = 1; i <= 1000000; i++) Faça
    Escrever(i)
Fim Para
```

(CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

No algoritmo a seguir, o comando x = x + 10 é executado quatro vezes.

```
x = 10;
para (y = 40; y < 100; y = y + 16)
    x = x + 10;
fim para
```

Comentários:

O Para inicia com a variável em 40, com o limite superior menor que 100, e incrementa o valor em 16 unidades a cada laço. Então, teremos execuções para y igual a 40, 56, 72 e 84. A próxima iteração seria com y = 100, que passa a não obedecer mais à condição de verificação (y < 100). Portanto, de fato, executamos o comando 4 vezes. (Gabarito: Certo)

Agora, vamos resolver uma questão de um nível avançado, que combina as estruturas condicionais e de repetição - o que é muito comum em questões e na prática do dia a dia. Vamos lá!

(VUNESP/TJM SP/2023) Analise o programa a seguir, apresentado na forma de uma pseudolinguagem (Português Estruturado).

```
Início
  Inteiro: x, y, z, i;
  x ← 3;
  y ← 3;
  z ← 3;
  Para i de 1 até 6 faça
  [      Se (z = 3)
        Então
```

```
        [
            x ← x+1;
            y ← y+2;
            z ← z-1;
        ]
        Senão
        [
            z ← z+1;
        ]
    ]
    z ← x + y + z;
Fim.
```

O valor da variável z ao final da execução desse programa será:

a) 15
b) 16
c) 17
d) 18
e) 19

Comentários:

Não deixe o tamanho da sintaxe te assustar. Encare com calma, linha por linha, para entender o que está acontecendo. Iniciamos o código declarando 3 variáveis do tipo inteiro, x, y e z - todas com o valor igual a 3. Em seguida, iniciamos um loop `Para` numa notação um pouco diferente do que vimos, mas basicamente estamos iniciando em 1, indo até 6 com incrementos de uma unidade - ou seja, seria como Para (i = 1, i <= 6; i++) - ou seja, 6 laços.

O bloco de código executado a cada laço é um `Se...então...senão`. A condição `Se` é z = 3 e, como iniciamos o programa com z = 3, iniciaremos o bloco fazendo as operações elencadas no bloco `Se`. Vamos lá!

Laço 1 - i = 1; z = 3 (bloco Se);
x = x + 1 = 3 + 1; x = 4
y = y + 2 = 3 + 2; y = 5
z = z - 1 = 3 - 1; z = 2 <- como caímos em z diferente de 3, o próximo bloco será no senão.

Laço 2 - i = 2; z = 2 (bloco Senão);
z = z + 1 = 2 + 1; z = 3

Laço 3 - i = 3; z = 3 (Bloco Se)

x = x + 1 = 4 +1; x = 5
y = y + 2 = 5 + 2; y = 7
z = z - 1 = 3 - 1; z = 2

Laço 4 - i = 4; z = 2 (bloco Senão)
z = z + 1 = 2 + 1; z = 3

Laço 5 - i = 5; z = 3 (bloco Se)
x = x + 1 = 5 + 1; x = 6
y = y + 2 = 7 + 2; y = 9
z = z - 1 = 3 - 1; z = 2

Laço 6 - i = 6; z = 2 (bloco Senão)
z = z + 1 = 2 + 1; z = 3

Bom, fizemos os laços, e terminamos com os seguintes valores:

x = 6
y = 9
z = 3

Por fim, temos uma nova atribuição de valor à variável z antes de finalizar o programa:

z = x + y + z
z = 6 + 9 + 3 = 18

Portanto, o valor final de Z será igual a 18. (Gabarito: Letra D)

Estruturas de Desvio

Pode ser que determinadas linhas de código só precisem ser executadas se determinada condição for (ou não for) satisfeita. Por exemplo, se determinada condição for falsa, pode ser que tenhamos que executar uma série de salvaguardas para proteger a aplicação, como numa tentativa de invasão de *hackers*. Porém, se a condição for verdadeira, esse determinado bloco não seria executado.

Embora muitas estruturas que vimos podem agir implementando esse "salto" de linhas, como, por exemplo, uma estrutura condicional, temos uma estrutura específica para esses casos: é a estrutura de desvio incondicional - representada nos algoritmos por **`Vá para`**, ou, em inglês, `Go to`. Apesar de não termos essa limitação em pseudocódigo, essa estrutura de desvio não é suportada nativamente por diversas linguagens, necessitando de uma adaptação através de funções.

Veja um exemplo:

Pseudocódigo

```
Inicio

int nota
String situaçao

Se (nota >= 7) Então
    Situaçao = "Aprovado"
    Vá_para Fim
Fim Se

Função Prova_de_Recuperação() {
    //bloco de código
}

Se (nota < 5) Então
    Situaçao = "Reprovado"
Fim Se

Fim
```

No exemplo acima, pularemos toda a parte do código necessário para gerar uma prova de recuperação, caso a nota do aluno seja maior ou igual a 7. Outros casos comuns de uso do Vá para envolvem *loops* aninhados - onde, se determinada condição no *loop* interno for satisfeita, usamos um comando de desvio para encerrar o *loop* total, tanto interno quanto externo.

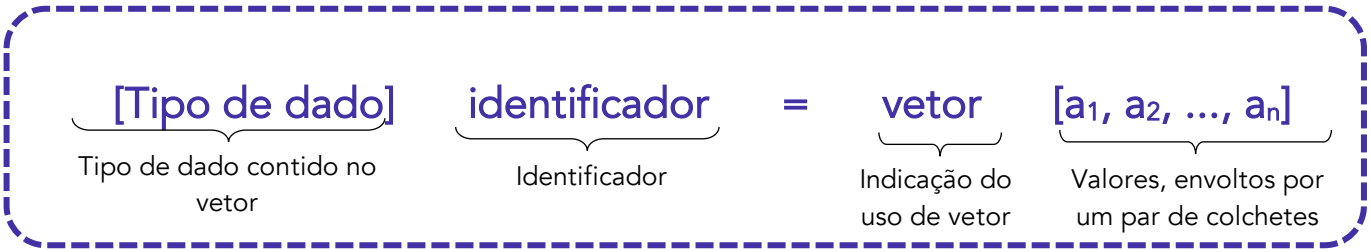
Estruturas de Dados

As Estruturas de Dados são formas de agruparmos diversos valores em uma única variável. Esses assuntos são vistos com mais profundidade em aula específica, mas, como muitas questões de lógica de programação exigem conhecimentos acerca desses assuntos, vou passar um panorama geral para você do que é cada estrutura de dados. Veremos 5 estruturas na aula de hoje:

- Vetores
- Listas
- Filas
- Pilhas
- Matrizes
- Registros

Vetores

Os vetores, também referenciados pelo seu nome em inglês, *array*, são estruturas de dados homogêneas, isso é, recebem somente um tipo de dado, que comportam uma coleção de dados ordenados. Além disso, eles são um tipo de estrutura unidimensional, ou seja, os dados são agrupados em uma única direção, formando uma lista de dados. Para declararmos vetores, vamos usar o seguinte padrão:

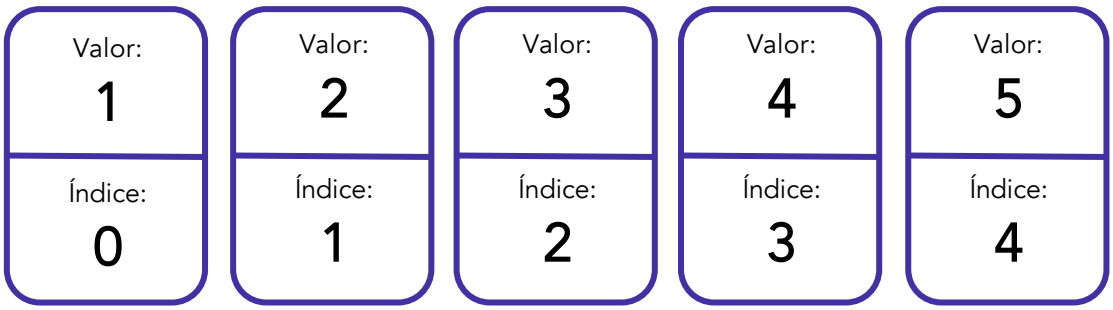


Então, por exemplo, imagine a seguinte declaração:

Pseudocódigo

```
int vetorNumeros = vetor [1, 2, 3, 4, 5]
```

Cada elemento, desses 5 atribuídos ao vetor, é delimitado por uma posição - chamada de índice. Esse índice, ou *index* em inglês, é útil para acessarmos determinado valor dentro do vetor. Usualmente, começa-se a contagem de posições a partir do 0. Portanto, o primeiro elemento terá a posição 0, e incrementamos a posição de 1 em 1. Veja:



Para acessarmos determinado elemento do vetor, basta delimitarmos a posição desse elemento dentro de um par de colchetes - veja:

Pseudocódigo

```
int vetorNumeros = vetor [1, 2, 3, 4, 5]

Escrever(vetorNumeros[2])
```

```
3
```

Um vetor possui algumas características intrínsecas que o definem:

- É uma estrutura de dados estática, isso é, possui um tamanho fixo determinado na hora da sua criação
- São armazenados de forma contígua na memória, possibilitando uma pesquisa mais veloz de elementos no seu corpo.

Listas

As listas são uma estrutura de dados que comportam uma multiplicidade de valores de forma ordenada, assim como os vetores - mas elas tem algumas diferenças basilares entre si. Enquanto os vetores são estáticos, as listas são dinâmicas, isso é, permitem a inserção e remoção de elementos livremente, já que não temos um tamanho fixo.

Veja uma comparação:

Aspecto	Vetores	Listas
Tamanho	Fixo	Variável
Acesso	Diretamente por índice	Acesso sequencial
Inserção/Remoção	Envolve operações complexas, como cópias e deslocamentos	Inserção e remoção livres
Flexibilidade	Baixa	Alta
Memória	Ocupa espaços contíguos	Não há uma delimitação, podendo resultar em fragmentação

Imagine uma lista enorme, com 1.000 números. A complexidade de escrever os valores manualmente, como fizemos no vetor, passa a ser maior, não é? Para resolver isso, podemos começar com uma lista vazia, e alimentá-la através de operações de atribuição (`+=`) dentro de um loop. Veja como ficaria.

Pseudocódigo

```
int lisaLonga = lista []

Para (i = 1; i <= 1000; i++) Faça:
    listaLonga += i
Fim Para
```

Espero que tenha percebido como o conhecimento na programação é granular - na sintaxe acima, apesar de curta, exploramos boa parte do que vimos durante a aula. Se você não a entendeu, é sinal que pulou alguma das etapas anteriores. Por isso recomendo que só prossiga a aula se entendeu todos os passos feitos acima.

(CEBRASPE/Pref. Fortaleza/2023) Julgue o item que se segue, relativo a conceitos de avaliação de expressões.

Após o algoritmo a seguir ser executado, o valor da variável soma1 será maior que o da variável soma2.

```
vetor a[7];
real soma1, soma2;
inteiro i;

a = [1,3,9,27,81,243,729];}
soma1 = 0;
i = 0;

enquanto (i < 7) faça
soma1 = soma1 + a[i]
i = i + 1
fim enquanto

soma2 = 1 * (1-3^7)/(1-3)
```

escreva(soma1)
escreva(soma2)

Comentários:

A questão traz algumas variáveis, dentre elas um vetor *a* com 7 elementos. Vamos achar o valor de soma1 e soma2, para verificarmos o apontamento da afirmativa. Vamos começar com o soma1.

O loop Enquanto irá somando o valor atual de soma1 com a respectiva posição no vetor. Então, como i = 0, e o aumento a cada laço é incremental, faremos 7 loops (de i = 0 até i = 6). Vamos ver o primeiro laço:

1) Laço 1 - i = 0:
soma1 = soma1 + a[0]
soma1 = 0 + 1 = 1

2) Laço 2 - i = 1:
soma1 = soma1 + a[1]
soma1 = 1 + 3 = 4

3) Laço 3 - i = 2:
soma1 = soma1 + a[2]
soma1 = 4 + 9 = 13

4) Laço 4 - i = 3:
soma1 = soma1 + a[3]
soma1 = 13 + 27 = 40

5) Laço 5 - i = 4:
soma1= soma1 + a[4]
soma1 = 40 + 81 = 121

6) Laço 6 - i = 5
soma1 = soma1 + a[5]
soma1 = 121 + 243 = 364

7) Laço 7 - i = 6
soma1 = soma1 + a[6]

soma1 = 364 + 769 = 1039

Portanto, o valor final da variável soma1 é 1.039. Na hora da prova, se você tiver entendido a lógica - que estamos somando todos os valores do vetor - a resolução se torna muito mais rápido. Só iríamos somar os valores de a, e encontraríamos o mesmo valor.

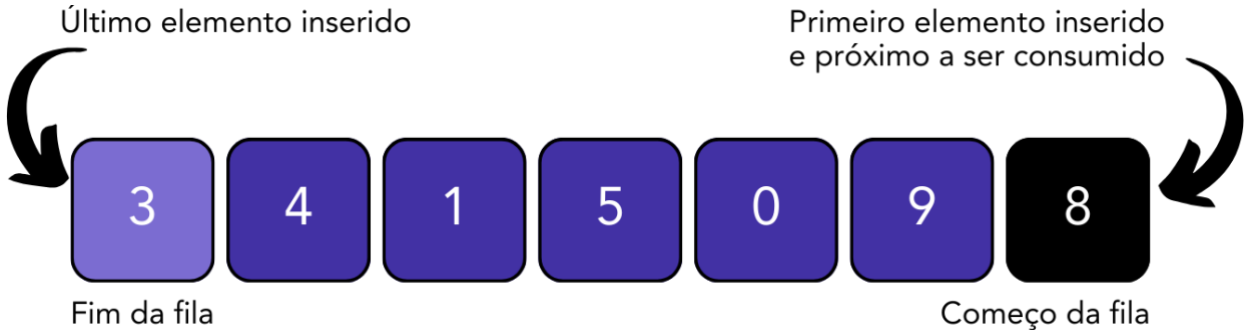
Agora, vamos para a soma2.

soma2 = 1 * (1 - (3^7)) / (1-3)
soma2 = 1 * (1 - 2187) / (-2)
soma2 = 1 * (-2186) / (-2)
soma2 = 1 * 1.093
soma2 = 1.093

Então, o valor de soma2 (1.093) é maior que o valor de soma1 (1.039). Sendo assim, afirmativa incorreta. (Gabarito: Errado)

Filas

As filas, ou ***queues,*** são estruturas de dados homogêneas e unidimensionais, assim como as listas - mas possuem uma propriedade especial: as inserções e remoções da sua estrutura utilizam a regra "primeiro a entrar, primeiro a sair" - ou FIFO ("First In, First Out"). Ou seja, sempre que formos inserir um elemento na estrutura, ele será inserido no fim da fila, e quando retirarmos algum dado, o primeiro dado da fila será retirado.



As operações com uma fila incluem:

- Enfileirar (enqueue): adicionar um elemento ao final da fila
- Desenfileirar (dequeue): retirar um elemento do começo da fila
- Frente (front): obter o valor do elemento que está na frente da fila

As filas, assim como a próxima estrutura de dados, pilhas, não são implementadas de forma direta por algumas linguagens - então, é comum o uso de extensões ou a implementação através de um

tipo especializado de lista. Aqui na lógica de programação, como não trabalhamos com uma linguagem específica, não temos problemas quanto a isso.

(CEBRASPE/SEPLAN RR/2023) Julgue o item a seguir acerca dos conceitos de estrutura de dados.

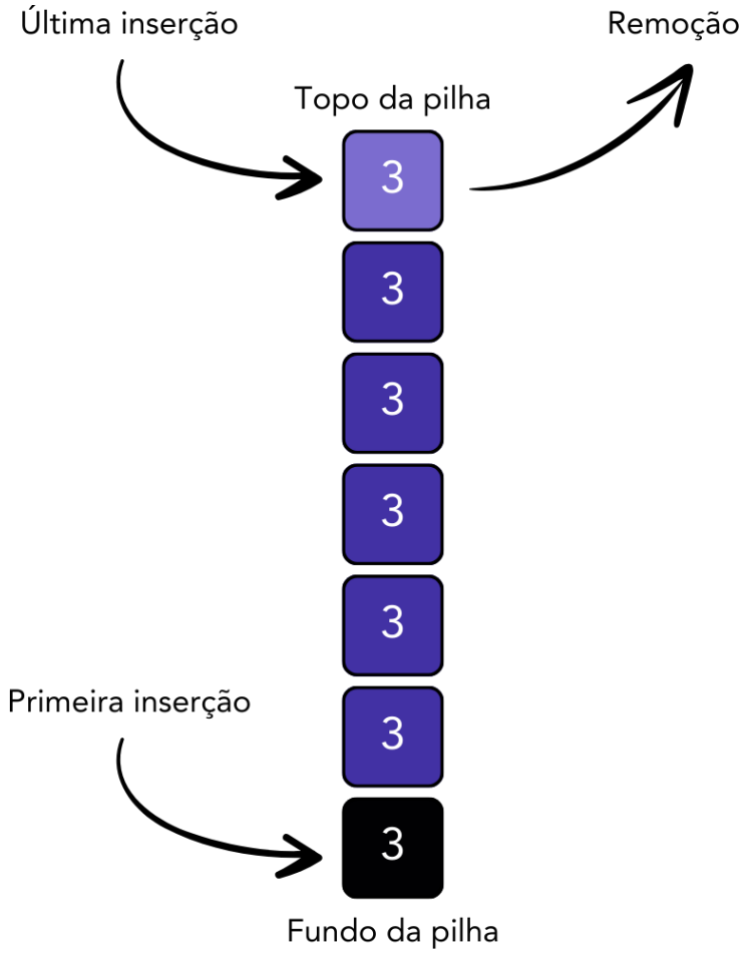
Sempre que houver uma remoção na estrutura de dados denominada fila, o elemento removido será aquele que está na estrutura há mais tempo.

Comentários:

A remoção de elemento na estrutura de fila é direcionada ao elemento da "frente", do começo da fila. Como esse elemento é, de fato, o que está há mais tempo na fila, que teve sua inserção feita primeiro, a alternativa está correta. (Gabarito: Correto)

Pilhas

As pilhas, ou stacks, assim como as filas, são estruturas de dados homogêneas e unidimensionais. Aqui, novamente, temos uma estrutura diferenciada, utilizando o padrão "Último a Entrar, Primeiro a Sair" - ou LIFO ("Last In, First Out"). Pense num pacote de Pringles - a última batata adicionada ao pacote será aquela no topo do tubo, e essa também será a primeira a ser retirada. A estrutura em pilha funciona da mesma forma.



As operações relacionadas à pilha são:

- Empilhar ou empurrar (push): adicionar um elemento ao topo da pilha
- Desempilhar ou puxar (pop): remove um elemento do topo da pilha
- Topo (top): retorna o valor do elemento no topo da pilha

(IBFC/TRF 5/2024) Estruturas de dados são constantemente utilizadas em algoritmos para resolução de problemas, desde os mais simples aos mais complexos, desta forma, estrutura de dados utiliza o princípio "Último a entrar, primeiro a sair"(LIFO):

a) Fila
b) Lista Encadeada
c) Pilha
d) Árvore

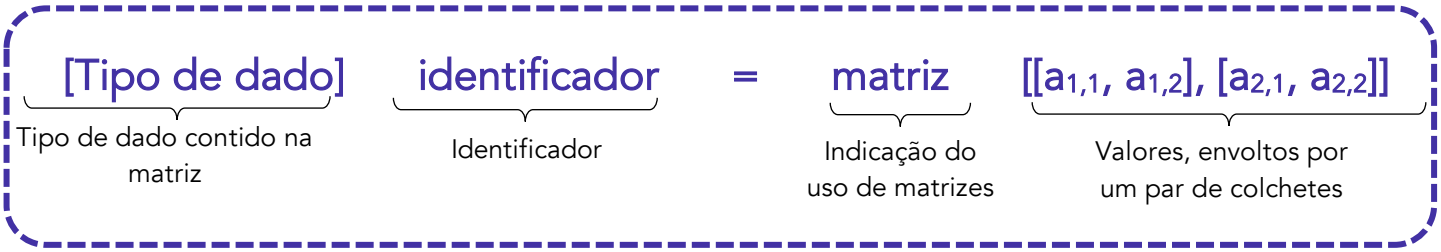
Comentários:

A estrutura que usa o padrão LIFO são as pilhas. (Gabarito: Letra C)

Matrizes

As matrizes são estruturas de dados homogêneas, assim como os vetores - mas a principal diferença aqui é que passamos a trabalhar com duas dimensões. Assim, passamos a trabalhar com índices em duas dimensões - a dimensão horizontal e vertical. Pense em uma estrutura similar a uma matriz que você estudou em matemática, no ensino médio.

Para declararmos uma matriz é simples, temos um processo similar ao da lista - porém, ao delimitarmos os colchetes, cada linha será delimitada por um outro par de colchetes internos. Veja uma estrutura geral:



Assim como nos vetores, podemos acessar os elementos da matriz através de seu índice. Porém, agora temos 2 eixos para verificar a posição, assim sendo necessário dois argumentos ao especificarmos o dado que queremos recuperar. Por exemplo, matriz[0, 0] irá retornar o primeiro elemento, da primeira linha.

Veja um exemplo em uma tabela - nela, iremos selecionar o elemento matriz[2, 3], onde 2 representa a linha, e 3 a coluna.

Índice	0	1	2	3	4
0	Brasil	EUA	Canada	México	Bolívia
1	Japão	Egito	Gana	China	Bélgica
2	Argentina	Itália	Inglaterra	Peru	Austrália
3	Rússia	Nepal	Holanda	Cuba	Líbia

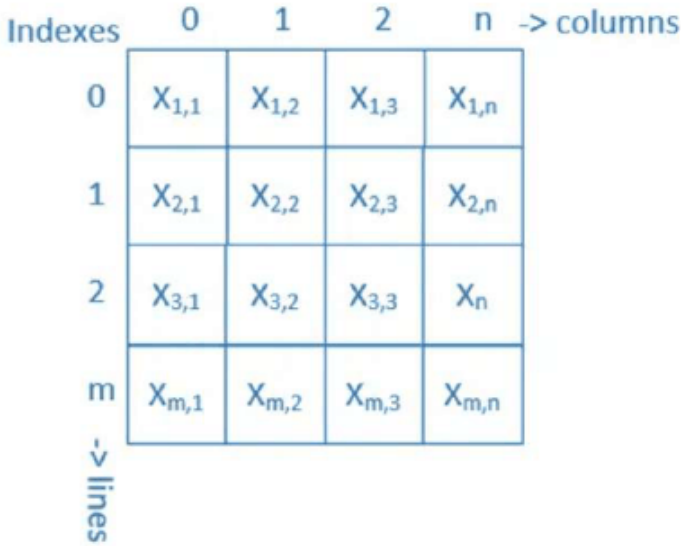
Em termos de sintaxe, poderíamos montar essa tabela e selecionar o mesmo elemento da seguinte forma:

Pseudocódigo

```
int matrizPaises = matriz [
    ["Brasil", "Eua", "Canada", "México", "Bolívia"],
    ["Japão", "Egito", "Gana", "China", "Bélgica"],
    ["Argentina", "Itália", "Inglaterra", "Peru", "Austrália"],
    ["Rússia", "Nepal", "Holanda", "Cuba", "Líbia"]]
Escrever(matrizPaises[2, 3])
```

```
Peru
```

(IDCAP/CREA BR/2023) É correto afirmar que a imagem abaixo corresponde a um exemplo de:



a) Lista.
b) Árvore.
c) Conjunto.
d) Matriz.
e) Tabela espelhada.

Comentários:

A estrutura apresenta um conjunto de dados elencados em linhas em colunas, cada qual com seu respectivo índice. Essa estrutura caracteriza uma matriz. (Gabarito: Letra D)

(AOCP/IFNMG/2022) "Uma variável composta homogênea multidimensional é formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome), e alocadas sequencialmente na memória. Uma vez que as variáveis têm o mesmo nome, o que distingue são índices que referenciam sua localização dentro da estrutura. Uma variável desse tipo

precisa de um índice para cada uma de suas dimensões". Essa descrição se refere a qual estrutura de dados?

a) Registro.
b) Vetor.
c) Matriz.
d) Pilha.
e) Fila.

Comentários:

A estrutura multidimensional descrita pela questão são as matrizes. (Gabarito: Letra C)

Registros

O registro é uma estrutura de dados heterogênea, ou seja, agrupa diferentes tipos de dado numa mesma variável. Ele é usado para representar entidades ou objetos complexos, nos quais cada campo dentro do registro armazena um tipo específico de informação.

Em um registro, cada campo possui um nome único que o identifica e um tipo de dado associado que define o tipo de informação que pode ser armazenada nele. Por exemplo, em um registro de Pessoa, poderíamos ter campos como "nome" (String), "idade" (inteiro), "altura" (real), etc.

Quando definimos um registro, definimos um "esqueleto" de estrutura, sem passar nenhum valor para a variável. Posteriormente, podemos criar variáveis derivadas desse esqueleto, com toda a sua estrutura implementada. Vamos para um exemplo!

Pseudocódigo

```
registro Pessoa
    Nome: String
    Idade: Int
    Altura: Float
fim Registro
```

Com isso, criamos uma estrutura de registro. Agora, podemos declarar variáveis conforme essa estrutura, que vai funcionar como um "tipo de dado" para a variável declara. Assim, podemos passar valores para os diferentes identificadores internos da entidade. Vamos fazer duas atribuições, para que você possa entender melhor.

EXEMPLIFICANDO

Pseudocódigo

```
Pessoa pessoa1
pessoa1.nome = "Felipe"
pessoa1.idade = 30
pessoa1.altura = 1.88

Pessoa pessoa2
pessoa2.nome = "Milena"
pessoa2.idade = 24
pessoa2.altura = 1.62
```

Veja que "navegamos" através das diferentes variáveis dentro do registro através de um ponto - por exemplo, em pessoa1.nome, estou atribuindo um valor para a propriedade nome, da variável pessoa1. Isso permite também que retornemos esse valor através de uma seleção, assim como fizemos com os vetores e as matrizes.

Rotinas

Funções e Procedimentos

À medida que encontramos problemas cada vez mais complexos, é necessário quebrarmos um programa geral em diversos outros programas menores, "subprogramas", cada qual com uma funcionalidade definida. Essas pequenas partes que formam o programa são chamadas de **rotinas**. As rotinas facilitam a execução e legibilidade do código, além de adicionar a modularidade ao programa - já que definimos determinada funcionalidade, ela pode ser replicada em outros pontos do código, quando for necessário.

Existem dois tipos de rotinas na programação: as **funções** e os **procedimentos**. A principal diferença entre ambas está no retorno. **Funções trazem algum retorno ao código, enquanto procedimentos não**. Esse retorno é algum valor que afeta o código, que pode ser usado por outras variáveis - usualmente caracterizado pelo uso da palavra `Retorne`.

Vou dar um exemplo: quando usamos o código `Escrever(texto)` para que determinado texto seja mostrado no terminal, não temos nenhum retorno no código, apenas os dados sendo mostrados no terminal. Esse é um procedimento. Para funções, o exemplo mais clássico é o uso de uma função para fazermos a exponenciação de um número. Podemos fazer:

```
Função exp(a, b)
     retorne a^b
fim Função
```

Assim, se rodarmos exp(3, 2), seria o mesmo que escrevermos 3^2.

PROCEDIMENTO → NÃO RETORNA UM VALOR

FUNÇÃO → RETORNA UM VALOR

Feita essa explicação, peço uma licença poética a você: de forma geral, iremos nos referir tanto a procedimentos quanto a funções apenas por "funções".

(CEBRASPE/Pref. Fortaleza/2023) No que concerne a conceitos de algoritmos e blocos de comandos, julgue o item seguinte.

As funções são um bloco de código ou comandos constituindo um conjunto específico de instruções repetíveis, que recebem uma ou mais entradas e produzem alguma saída.

Comentários:

Perfeito! Funções são blocos de código que executam alguma instrução repetível, isso é, pode ser usada repetidamente sem "esgotamento". Para ser caracterizada como uma função, deve haver um retorno. (Gabarito: Correto)

Para declararmos uma função, usaremos a seguinte estrutura:

```
Função (parâmetros):
    Bloco de código
Fim Função
```

Para entendermos melhor como implementar essa estrutura, precisamos entender os parâmetros.

Parâmetros

Os parâmetros (também chamados de argumentos) em funções são variáveis locais que são especificadas como parte da definição de uma função e que recebem valores quando a função é chamada ou invocada. Eles servem como mecanismo para passar informações para dentro de uma função, permitindo que ela trabalhe com valores específicos.

Quando uma função é definida, os parâmetros são listados entre parênteses após o nome da função. Por exemplo, em uma função que calcula a soma de dois números, os parâmetros seriam os dois números que estão sendo somados. Quando a função é chamada, os valores passados como argumentos para os parâmetros são atribuídos às variáveis de parâmetro dentro da função, permitindo que ela os utilize em suas operações internas.

Você pode não ter percebido, mas quando lhe expliquei a função exponencial, usamos os parâmetros. Vamos relembrar:

```
Função exp(a, b)
     retorne a^b
Fim Função
```

Aqui, temos os parâmetros a e b. Como é uma função de exponenciação, o parâmetro *a* será nossa base, e o parâmetro *b* o expoente. Assim, toda vez que chamarmos uma função, passaremos os parâmetros que queremos calcular. Por exemplo podemos achar 3^2 passando a função exp(3,2), ou 3^3 passando a função exp(3,3).

(CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

Na instrução A(i) = FUNC(i), a saída da função FUNC( ) é passada corretamente como parâmetro de entrada (i) para a função A.

Comentários:

Na instrução A(i) = FUNC(i), o valor de retorno da função FUNC(i) está sendo atribuído à variável A na posição i, mas não está sendo passado como parâmetro para a função A. A atribuição acontece após a chamada da função FUNC(i), portanto, não está sendo passado como parâmetro de entrada para a função A. (Gabarito: Errado)

Chamar uma função é o ato de solicitar a execução do bloco de código associado a essa função. Quando uma função é definida, ela cria uma espécie de "caixa preta" que contém um conjunto de instruções que podem ser executadas quando a função é chamada. Para chamar uma função, utilizamos seu nome seguido por parênteses que contêm os parâmetros necessários para a execução da função, se houver.

Veja mais um exemplo de uma função simples.

Pseudocódigo

```
Função soma (a, b)
    retorne a + b
Fim Função

Escrever(soma(3, 4))
Escrever(soma(2, 2))
Escrever(soma(3, 9))
```

```
7
4
12
```

Na sintaxe acima, temos dois parâmetros: **a** e **b**. Ao chamarmos a função, passamos os valores substituindo os parâmetros - esses valores podem ser declarações diretas de números, como fizemos ou referências a outras variáveis. A função irá somar ambos os números e retornar o resultado.

Podemos também fazer funções mais avançadas, com blocos de códigos extensos - incluindo outras funções locais, declaração de variáveis locais, entre outros. Vamos fazer um exemplo, uma função que irá identificar se determinado valor é par ou não.

Pseudocódigo

```
Função par(i)
    Se (i%2 == 0) Então
        Retorne "É par"
    Senão
        Retorne "É impar"
Fim Função

Escrever (par(2))
Escrever (par(0))
Escrever (par(3))
Escrever (par(9))
```

```
É par
É par
É impar
É impar
```

A função apresentada irá passar o parâmetro apresentado para um bloco condicional. Se o resto da divisão do parâmetro por 2 for 0, teremos um número par - e retornaremos "É par". Caso contrário, executaremos o bloco Senão e teremos um retorno "É ímpar".

(Inédita/Prof. Felipe Mathias) Considere a seguinte função em pseudocódigo:

```
Função soma_fatorial(n)
   Soma = 0
   Para i de 1 até n faça
      Fatorial = 1
      j = 1
      Enquanto j <= i faça
         Fatorial = Fatorial * j
         j = j + 1
      Fim Enquanto
      Soma = Soma + Fatorial
   Fim Para
   Retorne Soma
Fim Função
```

Qual seria o valor retornado pela função soma_fatorial(3)?

A) 3
B) 6
C) 9
D) 12

Comentários:

Estamos rodando a função com n = 3, e dentro dela temos um bloco Para começando com i = 1, indo até i = 3 , com somas unitárias (ou seja, teremos 3 laços). Ademais, dentro do bloco Para, temos um outro loop Enquanto - que será executado enquanto a variável local j for menor ou igual à variável i. Vamos fazer os laços.

Laço 1 - Soma = 0; i = 1; n = 3; j = 1; Fatorial = 1

Devemos começar pelo bloco mais interno - o enquanto. Nele, calcularemos:

- Fatorial = Fatorial * j
- Fatorial = 1 * 1
- Fatorial = 1

- j = j + 1
- j = 1 + 1 = 2
- j = 2

Agora, podemos fazer a operação atribuída para soma:

- Soma = Soma + Fatorial
- Soma = 0 + 1
- Soma = 1

Laço 2 - Soma = 1; i = 2; n = 3; j = 2; Fatorial = 1

- Fatorial = Fatorial * j
- Fatorial = 1 * 2
- Fatorial = 2

- j = j + 1
- j = 2 + 1
- j = 3

- Soma = Soma + Fatorial
- Soma = 1 + 2
- Soma = 3

Laço 3 - Soma = 3; i = 3; n = 3; j = 3; Fatorial = 2

- Fatorial = Fatorial * j
- Fatorial = 2 * 3
- Fatorial = 6

- Como estamos no último laço, o valor de j não é mais importante

- Soma = Soma + Fatorial
- Soma = 3 + 6
- Soma = 9

Portanto, concluímos que o valor de Soma para soma_fatorial(3) é 9. Esse programa é responsável por calcular a soma de todos os fatoriais até de 1 até 3 - portanto, somamos o fatorial de 1 (1), de 2 (2) e de 3 (6). (Gabarito: Letra C)

Essa passagem de parâmetros que vimos até agora é conhecida como passagem de valor. Podemos ter uma outra forma de passagem de parâmetros chamada de referência. Aqui, ao invés de passar uma cópia do valor de uma variável para uma função, é passada uma referência (ou endereço de memória) para a variável original. Isso significa que a função pode modificar o valor da variável original diretamente, já que ela tem acesso direto à sua localização na memória.

Guarde a diferença!

PARÂMETRO POR VALOR → PASSA UMA CÓPIA DO VALOR ORIGINAL

PARÂMETRO POR REFERÊNCIA → PASSA UMA REFERÊNCIA AO LOCAL DE ARMAZENAMENTO

(CONSULPLAN/Pref. Campos dos Goytacazes/2024) Em programação, sobre a passagem de parâmetros por referência em comparação com a passagem de parâmetros por valor, assinale a afirmativa correta.

a) A passagem por referência é uma técnica que permite que variáveis sejam executadas em paralelo; ao contrário da passagem por valor.
b) A passagem de uma referência à memória onde o valor está armazenado; a passagem por valor envolve a cópia do valor real do argumento para a função.
c) A passagem por referência permite modificar o valor da variável passada como argumento, não necessariamente uma variável global; ao contrário da passagem por valor.
d) A passagem por valor é uma técnica que permite modificar o valor de uma variável global a partir de uma função; a passagem por referência mantém a integridade do valor original.

Comentários:

Lembre-se, a passagem de parâmetros por referência passa um ponteiro, que aponta para o local na memória onde o dado está armazenado, permitindo a modificação do parâmetro diretamente pela função. Já a passagem por valor traz apenas uma cópia literal do parâmetro. (Gabarito: Letra B)

Temos alguns tipos especiais de funções. As principais, e que você precisa saber, são:

- Funções recursivas
- Funções anônimas
- Funções de ordem superior
- Funções nominadas
- Funções puras

Tipos de Funções

Funções recursivas

Pode ser que uma função chame a si mesma durante o a sua execução. Esses casos são chamados de funções recursivas. Esses casos são úteis para cálculos matemáticos complexos, como Fibonacci, navegações de estruturas de dados mais avançadas, algoritmos mais avançados, entre outros.

Para estruturarmos uma função recursiva, não basta simplesmente chamar a própria função dentro do seu corpo - afinal, se fizermos isso, teremos um *loop* infinito que "quebrará" o código. Devemos seguir uma estrutura composta de:

- Um caso base, literal, que determina quando a função recursiva deve parar de chamar a si mesma e começar a retornar valores . Ela é aparte essencial para que não entremos em um loop infinito.
- Uma chamada recursiva, que irá chamar a função principal com um argumento diferente a cada laço. Cada chamada recursiva normalmente reduz o problema em direção ao caso base, garantindo que, eventualmente, o caso base seja alcançado e a recursão pare.

Função Recursiva

Caso base

Chamada recursiva

(CEBRASPE/PETROBRAS/2022) Julgue o item subsequente, a respeito de algoritmos para ordenação e pesquisa e de programação recursiva.

Uma função é dita recursiva quando, dentro dela, é feita uma ou mais chamada a ela mesma.

Comentários:

Perfeito! Uma função recursiva é aquela que chama a si mesmo um ou mais vezes dentro de seu próprio bloco de código. (Gabarito: Certo)

(FGV/SES MT/2024) A sequência de Fibonacci tem aplicação, dentre outras, em algoritmos de busca, classificação e criptografia. Trata-se de uma lista infinita de números, em que cada um de seus valores é o resultado da soma dos dois anteriores. Matematicamente, esta relação de recorrência é representada por:

f(0) = 0
f(1) = 1
f(n) = f(n-1) + f(n+2), para qualquer natural n > 1

Considere que foram elaboradas duas implementações algorítmicas definidas em linguagem Python (CODIGO-01 e CODIGO-02).

CODIGO-01

```
def f(n: int):
    if n == 1:
        return 1
    if n == 2:
        return 1
    else:
        return f(n-1) + f(n-2)
```

CODIGO-02

```
def f(n: int):
    ultimo = 1
    penultimo = 1
    if n == 1:
        return penultimo
    elif n == 2:
        return ultimo
    else:
        atual = 0
        for i in range(2, n):

            atual = ultimo + penultimo
            penultimo = ultimo
            ultimo = atual
        return atual
```

Quanto às implementações, assinale a afirmativa correta.

a) Somente o primeiro algoritmo emprega função recursiva.
b) Os dois algoritmos empregam função recursiva.
c) Somente o primeiro algoritmo emprega função iterativa.
d) Os dois algoritmos empregam função iterativa.

Comentários:

Trouxe essa questão mais como um desafio a você - apesar de ela estar expressa numa linguagem real, em Python, você deve ter sido capaz de identificar a recursividade apenas no código 01, nesse seguinte trecho:

```
    else:
        return f(n-1) + f(n-2)
```

Estamos chamando a própria função f, com parâmetros diferentes. Já no código 02, não conseguimos encontrar nenhum caso de recursividade, já que a função f só é citada na sua própria definição, não dentro do seu bloco de código. Portanto, podemos afirmar que somente o primeiro algoritmo emprega função recursiva. (Gabarito: Letra A)

Vou trazer um exemplo para você entender melhor o funcionamento, vamos lá.

EXEMPLIFICANDO

```
Função fatorial(n)
    Se n == 0 Então          → Bloco base
        Retorne 1

    Senão                    → Chamada recursiva
        Retorne n * fatorial (n - 1)

Fim Função
```

Vamos chamar essa função e ver como dá seu funcionamento? Vamos começar com `fatorial(3)`.

Laço 1 - n = 3

Como n ≠ 0, vamos cair no bloco Senão. Nele, vamos fazer o seguinte:

- n * fatorial (n - 1) = 3 * fatorial(2)

Terminamos com um bloco que não sabemos o resultado - a função fatorial(2). Precisamos ir para o próximo laço da recursividade, para encontrarmos esse valor.

Laço 2 - n = 2

Novamente, temos n ≠ 0, caímos no bloco Senão. Nele, faremos:

- n * fatorial (n - 1) = 2 * fatorial(1)

Mesmo problema, Temos achar o resultado de fatorial(1).

Laço 3 - n = 1

Continuamos na mesma situação, vamos executar o bloco Senão:

- n * fatorial (n - 1) = 1 * fatorial(0)

Continuamos tendo que achar o valor de fatorial(0).

Laço 4 - n = 0

Finalmente chegamos no caso em que vamos executar o bloco base. Como temos n == 0, iremos simplesmente retornar 1. Então, podemos afirmar que fatorial(0) = 1.

Perceba que chegamos a o que eu gosto de chamar de "fundo do poço" da recursividade. Chegamos ao caso em que ativamos o bloco base, e, como estamos no fundo do poço, só nos resta subir. Iremos fazer agora o caminho oposto - substituindo o valor das funções que ficaram sem atribuição.

- Laço 4 → fatorial(0) = 1
- Laço 3 → fatorial(1) = 1 * fatorial(0) = 1 * 1 = 1
- Laço 2 → fatorial(2) = 2 * fatorial(1) = 2 * 1 = 2
- Laço 1 → fatorial(3) = 3 * fatorial(2) = 3 * 2 = 6

Portanto, encontramos que a execução de fatorial(3) é igual a 6. Perceba que vamos agregando os valores encontrados, até voltarmos ao topo da recursão e encontrarmos o valor que pretendíamos originalmente.

Funções puras

Uma função é chamada de função pura se, e somente se, dada uma mesma entrada, sempre retornará uma mesma saída, sem efeitos colaterais observáveis. Isso quer dizer que o resultado da função depende apenas de seus argumentos, e não de estados externos, como variáveis globais ou estados do sistema.

Portanto, uma função pura tem 3 características principais:

- Determinismo: Dadas as mesmas entradas, uma função pura sempre produzirá o mesmo resultado.
- Ausência de efeitos colaterais: Funções puras não modificam variáveis globais, arquivos, bancos de dados, etc. Seu único efeito é retornar um valor.

- Transparência referencial: Pode-se substituir a chamada de uma função pura pelo seu resultado, sem alterar o comportamento do programa.

(CESGRANRIO/IPEA/2024) Na programação funcional, que é um paradigma suportado pela Linguagem Scala, uma das práticas fundamentais é o uso de funções puras.

A principal propriedade que caracteriza uma função pura é a(o)

a) baixa coesão
b) ausência de efeitos colaterais
c) incapacidade de chamar outra função
d) alto acoplamento
e) encapsulamento aberto

Comentários:

A questão exige conhecimento sobre a linguagem Scala, mas isso é indiferente, já que a definição de uma função pura independe da linguagem. Das alternativas apresentadas, a única que aponta uma característica da função é a letra B, com a ausência de efeitos colaterais. (Gabarito: Letra B)

Funções nominadas

As funções nominadas, sem surpresa, são funções com nomes específicos e podem ser chamadas por esse nome, em qualquer parte do programa. Ela é declarada com um nome específico, de forma a permitir sua reutilização ao longo do programa. Até agora, todas as funções que trabalhamos são funções nominadas.

Existem algumas funções nominadas "especiais", que já foram pré-concebidas e têm seu nome restrito ao uso por outras funções. Elas trazem funcionalidades pré-programadas, como o `print()` em Python, que faz uma impressão do código no terminal de comando, ou o `length()` em JavaScript, que registra o comprimento de um objeto.

Funções anônimas

Funções anônimas são chamadas assim pois não possuem nome - são um contraponto às funções nominadas. Elas só podem ser utilizadas localmente, já que não possuem uma referência para serem chamadas externamente. O exemplo mais comum de funções anônimas são as funções *lambda*, em Python.

Veja um exemplo em pseudocódigo:

Pseudocódigo

```
int Quadrado = Função(x) retornar x * x
int Resultado = Quadrado(5)

5
```

Veja que estamos chamando a variável Quadrado, que especifica uma função local, e não a própria função - até porque ela não tem nome, não temos como chamá-la.

Funções de ordem superior

Uma função de ordem superior é uma função que recebe outra função como argumento e/ou retorna uma função como resultado. Em outras palavras, uma função de ordem superior trata funções como cidadãos de primeira classe, permitindo que elas sejam passadas e retornadas como qualquer outro tipo de dado. Veja um exemplo:

Pseudocódigo

```
Função operacaoMatematica(x, y, operacao)
    retornar operacao(x, y)
fim Função
```

A princípio, a função parece uma função simples, apenas com três parâmetros - mas, no corpo dela, dentro do bloco de código, estamos chamando outra função, a função `operacao()`. Essa função pode ser definida externamente, e passada como parâmetro. Veja como podemos desenvolver esse caso.

Pseudocódigo

```
Função aplicarOperacaoMatematica(x, y, operacao)
    retornar operacao(x, y)
fim Função

Função soma(a, b)
    retornar a + b
fim Função

Função multiplicacao(a, b)
    retornar a * b
fim Função

Int resultado1 = operacaoMatematica(5, 3, soma)
Int resultado2 = operacaoMatematica(5, 3, multiplicacao)

Escrever(resultado1)
Escrever(resultado2)
```

```
8
15
```

Vamos resolver um exercício complexo? Nele iremos misturar as funções de ordem superior com recursividade.

(CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Após executado, o algoritmo a seguir apresentará 720 como resultado final.

```
função f(x)
  se (x == 0 ou x == 1)
    retorna 1;
  fimse
  retorna f(x - 1)*x
fimfunção

função func(a)
  se (a == 0)
    retorna 2
  fimse
  retorna a + f(a - 1)
fimfunção
```

escreva(func(6));

Comentários:

Temos duas funções - uma função recursiva f(x) e outra função de ordem superior func(a). Estamos chamando primeiramente a função de ordem superior, com func(6). Então, vamos ver o que acontece.

Como o parâmetro a = 6, vamos cair no bloco alternativo - vamos retornar a + f(a - 1), ou seja, a + (f5). Sendo assim, precisamos subir para a função f(x) e resolvê-la com x = 5. E nela, novamente, como não atendemos a condição do Se, vamos executar o retorno - f(x-1) * x

Laço 1 - f(5) = f(4) * 5
Laço 2 - f(4) = f(3) * 4
Laço 3 - f(3) = f(2) * 3
Laço 4 - f(2) = f(1) * 2

Por fim, no laço 5 teremos f(1), e, como x == 1, caímos no bloco base - retornando 1. Como chegamos a um valor, podemos subir somando.

Laço 5 - f(1) = 1
Laço 4 - f(2) = f(1) * 2 = 1 * 2 = 2
Laço 3 - f(3) = f(2) * 3 = 2 * 3 = 6
Laço 2 - f(4) = f(3) * 4 = 6 * 4 = 24
Laço 1 - f(5) = f(4) * 5 = 24 * 5 = 120

Então, encontramos que f(5) é igual a 120. Então é essa a resposta... só que não! Agora temos que voltar para a função que foi chamada inicialmente, a `func`, nós paramos a execução do bloco com `a + f(5)`. Portanto, teremos a + 120 = 120 + 6 = 126.

Com isso terminamos a sua introdução à lógica de programação. É essencial que você entende o conteúdo dessa aula, ela irá facilitar (e muito) o aprendizado das demais linguagens de programação - que criam uma estrutura definida para implementar as diversas formas de interação que vimos na aula de hoje. Não esqueça de resolver as questões para aplicar o conhecimento da aula.

QUESTÕES COMENTADAS

01. (INQC/CPTRANS/2024) Observe o seguinte algoritmo:

```
Algoritmo Maior
Var
  num1, num2, maior: inteiro:
Inicio
   Leia (num1, num2);
   Se (num1>num2) então
      maior ← num1;
   senão
      maior ← num2;
   fimse;
   escreva (maior);
Fim
```

A ferramenta utilizada para construção desse algoritmo é:

a) diagrama de Nassi-Shneiderman
b) diagrama hierárquico de fluxo
c) pseudocódigo
d) fluxograma

Comentários:

Veja que não temos uma estrutura padrão para de linguagem - estamos usando uma linguagem estruturada apenas para descrever um algoritmo. Isso caracteriza o uso de pseudolinguagem.

Gabarito: Letra C

---

02. (CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

O pseudocódigo consiste em um texto estruturado com comandos escritos em linguagem humana, no qual se apoia a criação dos algoritmos computacionais.

Comentários:

Perfeito. O objetivo da pseudolinguagem, ou pseudocódigo, é estruturar algoritmos de programação através de uma linguagem natural, humana, para que qualquer um possa entender a "lógica" por detrás do programa.

Gabarito: Correto

---

03. (CEBRASPE/CAU BR/2024) Com relação à lógica de programação, julgue o próximo item.

A estrutura de controle IF, que pode ser classificada como do tipo iteração, determina o caminho que o algoritmo deve seguir, de acordo com determinada condição.

Comentários:

Temos 3 estruturas de controle num código:

- Estruturas sequenciais
- Estruturas de seleção
- Estruturas de iteração (ou repetição)

O IF, ou Se, é uma estrutura de **seleção**, que elenca uma série de blocos de código, e um deles será selecionado para execução conforme determinada condição pré-definida.

Gabarito: Errado

---

04. (IGEDUC/CM V Santo Antão/2024) Julgue o item a seguir.

O pseudocódigo é considerado uma linguagem de programação formal e executável, seguindo uma sintaxe e semântica específicas, semelhante a linguagens como Java e Python. Sua principal aplicação é na implementação direta de algoritmos em ambientes de desenvolvimento integrados, sem a necessidade de conversão para outra linguagem de programação.

Comentários:

Cuidado! O pseudocódigo não conta com uma estrutura definida e, portanto, não é uma linguagem executável. Diferentemente de linguagens como Java e Python.

Gabarito: Errado

---

05. (CEBRASPE/TST/2024)

```
programa principal
    inteiro i, contagem = 10, limite = 10;
```

```
    para (i = 0; i > limite; i = i + 1) faça
        contagem = contagem – 1;
    fim para
    imprime(contagem);
fim programa
```

Com base no algoritmo precedente, escrito em pseudocódigo, assinale a opção que corresponde ao tipo de estrutura em que se realiza o decréscimo da variável contagem.

a) estrutura de controle
b) estrutura de repetição
c) estrutura condicional
d) atributo
e) função

Comentários:

O bloco de código apresenta uma estrutura Para, que é uma das formas de implementarmos uma estrutura de repetição (iteração), principalmente nos casos em que sabemos a quantidade de laços (iterações, repetições) que queremos realizar.

Gabarito: Letra B

---

06. (CEBRASPE/TST/2024) Uma das vantagens do uso de funções predefinidas é

a) o reaproveitamento de código.
b) a passagem de parâmetros.
c) o recebimento do retorno das funções.
d) o grau de abstração do sistema.
e) a facilidade de implementação das funções.

Comentários:

Funções pré-definidas fazem parte das funções nominadas, que permitem seu reuso através de uma chamada posterior, utilizando o nome da função - que pode ser nativa à linguagem, ou criada pelo desenvolvedor. Com isso em mente, vamos às alternativas.

a) Certo. De fato, uma das maiores vantagens trazidas pelas funções predefinidas é a capacidade de reaproveitar o bloco de código ao longo do programa.
b) Errado. Passagem de parâmetros é um processo feito em qualquer tipo de função, não é inerente às funções predefinidas.
c) Errado. Toda função retorna determinado valor.

d) Errado. A abstração do sistema envolve o foco em partes mais importantes do código - o que não está ligado, necessariamente, às funções predefinidas.
e) Errado. Não é porque uma função foi definida previamente que ela será de fácil implementação. Existem funções predefinidas complexas, como as de ordem superior.

Portanto, a alternativa correta é a letra A.

Gabarito: Letra A

---

07. (IBFC/TRF 5/2024) Sobre Estruturas de Seleção/Condição, analise as afirmativas e dê valores Verdadeiro (V) ou Falso (F).

( ) A instrução else é obrigatória em todas as estruturas de condição.
( ) A instrução else em uma estrutura de condição é executada quando a condição no if é avaliada como falsa.
( ) Estruturas de condição não são necessárias em programação, pois é possível realizar todas as operações sem avaliar condições.

Assinale a alternativa que apresenta a sequência correta de cima para baixo.

a) F - F - F
b) F - V - F
c) V - F - V
d) V - V - V

Comentários:

Vamos analisar cada item.

( ) A instrução else é obrigatória em todas as estruturas de condição.
Falso. O *else*, ou Senão é uma forma de adicionarmos um caso residual - e nem sempre ele é obrigatório, até mesmo no uso de estruturas *If* (Se)

( ) A instrução else em uma estrutura de condição é executada quando a condição no if é avaliada como falsa.
Verdadeiro. Numa estrutura *if...else*, ou *Se...senão*, a execução do bloco *Else* só é implementada quando a condição verificada no bloco *if* for falsa.

( ) Estruturas de condição não são necessárias em programação, pois é possível realizar todas as operações sem avaliar condições.
Falso. Não temos outra forma de implementar as funcionalidades das estruturas condicionais sem o uso delas - e as condições são partes estruturantes e essenciais de um código.

Portanto, ficamos com F-V-F.

Gabarito: Letra B

---

08. (CEBRASPE/FINEP/2024) Assinale a opção que corresponde ao comando de salto incondicional utilizado em programação estruturada.

a) goto
b) struct
c) if
d) else
e) for

Comentários:

Na execução de um código, dada determinada situação, pode ser interessante (ou obrigatório) que linhas de código sejam puladas para a correta execução de um programa - como, por exemplo, se determinado bloco Se for executado, podemos pular instruções que introduziriam alternativas à execução desse bloco. A instrução que realiza esse "salto" é a goto (*vá para*).

Gabarito: Letra A

---

09. (CEBRASPE/POLC AL/2023) Julgue o próximo item, no que se refere a estruturas de controle de fluxo.

```
 // codigo 1
i = 1
fim = 7
enquanto i < fim faça
escreva(i)
i = i + 1
fim enquanto
depois = i

// codigo 2
fim = 6
i = 0
enquanto i < fim faça
  i = i + 1
  escreva(i)
```

fim enquanto
depois = i

Considerando a estrutura precedente, é correto afirmar que o código 1 e o código 2 apresentam o mesmo resultado.

Comentários:
Temos dois *loops* Enquanto. Ambos possuem a mesma estrutura - serão executados enquanto a variável i for menor que a variável fim (i < fim) e, a cada iteração, incrementarão o valor da variável i em 1 unidade. Vamos analisar a saída de cada código.

Código 1:

Iremos fazer 6 laços (de i = 1 a i = 6), e, a cada laço, iremos:

- Escrever o valor de i
- Incrementá-lo em uma unidade

Então, os laços serão os seguintes:

- Laço 1 - i = 1:
  - escreva (i) → 1
  - i = i + 1 → i = 2
- Laço 2 - i = 2:
  - escreva(i) → 2
  - i = i + 1 → i = 3
- Laço 3 - i = 3:
  - escreva(i) → 3
  - i = i + 1 → i = 4
- Laço 4 - i = 4:
  - escreva(i) → 4
  - i = i + 1 → i = 5
- Laço 5 - i = 5
  - escreva(i) → 5
  - i = i + 1 → i = 6
- Laço 6 - i = 6
  - escreva(i) → 6
  - i = i + 1 → i = 7

No laço 6, quebramos a condição e o bloco Enquanto. As saídas dessa estrutura foram (1, 2, 3, 4, 5, 6).

Código 2:

Iremos também fazer 6 laços, de i = 0 a i = 5. Aqui a ordem da execução da função muda:

- Primeiro somamos 1 ao valor de i
- Depois escrevemos o valor de i

Então, teremos os seguintes laços:

- Laço 1 - i = 0
  - i = i + 1 → i = 1
  - escreva(i) → 1
- Laço 2 - i = 1
  - i = i + 1 → i = 2
  - escreva(i) → 2
- Laço 3 - i = 2
  - i = i + 1 → i = 3
  - escreva(i) → 3
- Laço 4 - i = 3
  - i = i + 1 → i = 4
  - escreva(i) → 4
- Laço 5 - i = 4
  - i = i + 1 → i = 5
  - escreva(i) → 5
- Laço 6 - i = 5
  - i = i + 1 → i = 6
  - escreva(i) → 6

A saída aqui será (1, 2, 3, 4, 5, 6). Veja que temos a mesma saída - isso acontece pois a escrita do valor da variável acontece antes do incremento no código 1, e depois do incremento no código 2. Sendo assim, correta a afirmativa.

Gabarito: Correto

---

10. (VUNESP/TCM SP/2023) Analise o algoritmo a seguir, apresentado na forma de uma pseudolinguagem.

```
Início
[
  Tipo MAT = matriz[1..3,1..3] de inteiros;
  MAT: M;
  Inteiro: i, j, X, Y;
  X ← 0;
```

```
Y ← 0;
Para i de 1 até 3 faça
[
  Para j de 1 até 3 faça
  [
    Se i = j
      Então M[i,j] ← i + j + 1;
      Senão M[i,j] ← i + 2*j + 1;
  ]
]
Para i de 1 até 3 faça
[
  X ← X + M[i,i];
  Y ← Y + M[1,i];
]
Imprima (X+Y);
] Fim.
```

Ao final do algoritmo é impressa a soma (X+Y) que é igual a

a) 29.
b) 30.
c) 31.
d) 32.
e) 33.

Comentários:

Em questões longas como essa, aconselho que analise o código com muita calma e cuidado, para entender a ordem de execução dos comandos.

Nessa questão, estamos executando um *loop* Para, visando preencher uma matriz 3x3. Dentro desse *loop*, executamos outro Para que irá, de fato, preencher a lista. A interação de *loops* aninhados é a seguinte:

- Começamos a executar o *loop* externo com i = 1
- Executamos o loop interno de forma completa, com o j indo de 1 a 3
- Passamos para a execução do *loop* externo com i = 2, repetindo o processo, e para i = 3, repetindo o processo novamente

Então, como já sabemos quais valores começamos a execução do bloco mais interno do código, vamos começar.

Laço 1 - i = 1; j = 1

Vamos executar um Se, que irá preencher:

- O elemento $M_{i, j}$ com o valor i + j + 1, se i = j
- O elemento $M_{i, j}$ com o valor i + 2*j + 1, caso contrário.

Como temos i = j no primeiro laço, caímos no bloco Se, então:

- $M_{1, 1}$ = i + j + 1;
- $M_{1, 1}$ = 1 + 1 + 1 = 3

Laço 2 - i = 1; j = 2

Agora, i ≠ j, então caímos no bloco do *senão*. Então:

- $M_{1, 2}$ = i + 2*j + 1
- $M_{1, 2}$ = 1 + 2*2 + 1 = 6

Laço 3 - i = 1; j = 3

Novamente, vamos ao bloco senão.

- $M_{1, 3}$ = i + 2*j + 1
- $M_{1, 3}$ = 1 + 2*3 + 1 = 8

Com isso encerramos o primeiro laço do loop externo, com i = 1. Repetiremos esse processo pra i = 2, e i = 3. Os resultados de cada iteração serão:

- Laço 3: $M_{2, 1}$ = 5
- Laço 4: $M_{2, 2}$ = 5
- Laço 5: $M_{2, 3}$ = 9
- Laço 6: $M_{3, 1}$ = 6
- Laço 7: $M_{3, 2}$ = 8
- Laço 8: $M_{3, 3}$ = 7

Com esses valores, conseguimos construir nossa matriz:

ÍNDICE	1	2	3
1	3	6	8

2	5	5	9
3	6	8	7

Bom, agora que temos a matriz, temos mais um *loop* para ser executado - e mais um *loop* Para, que vai de 1 até 3, ou seja, 3 iterações. Nele, fazemos duas operações:

- Atribuímos incrementalmente o valor de M[i, i] à variável X
- Atribuímos incrementalmente o valor de M[1, i] à variável Y

Vamos resolver esses laços agora.

Laço 1 - i = 1:

- $X = M_{1, 1}$
- X = 3

- Y = M[1, 1]
- Y = 3

Laço 2 - i = 2:

- $X = X + M_{2, 2}$
- X = 3 + 5
- X = 8

- $Y = Y + M_{1, 2}$
- Y = 3 + 6
- Y = 9

Laço 3 - i = 3:

- $X = X + M_{3, 3}$
- X = 8 + 7
- X = 15

- $Y = Y + M_{1, 3}$
- Y = 9 + 8
- Y = 17

Então, terminamos com X = 15 e Y = 17. Por fim, finalmente (*ufa*), fazemos a soma dos valores para encontrar a resposta:

X + Y = 15 + 17 = 32

Gabarito: Letra D

---

11. (COCP IFMT/IFMT/2023) Segundo Manzano & Oliveira (2016), um laço incondicional, entre outras coisas, é um tipo de laço que é iterativo, porém não é interativo e tem seu funcionamento controlado por uma variável denominada contador. Com essa descrição, o autor se refere ao laço do tipo:

Fonte: (MANZANO, José Augusto & OLIVEIRA, Jayr Figueiredo de. Algoritmos: Lógica para desenvolvimento de programação de computadores. 18 ed. São Paulo: Erica, 2016).

a) para/fim_para
b) repita/até_que
c) enquanto/fim_enquanto
d) laço/fim_laço
e) execute/enquanto_for

Comentários:

As características que ajudam a identificar qual estrutura de repetição está sendo utilizado é:

- É incondicional, isso é, não analisa uma condição para ser executado
- Seu funcionamento é controlado por um contador

Isso nos denota o uso da estrutura repetitiva `Para`, que utiliza um contador para verificar quantas iterações serão feitas. Porém, como ele não analisa o resultado da iteração, somente o contador, ele não é um *loop* interativo.

Gabarito: Letra A

---

12. (CEBRASPE/CNMP/2023) Julgue o item subsecutivo, que se referem a conceitos de programação e banco de dados.

As estruturas condicionais são utilizadas para testar variáveis lógicas.

Comentários:

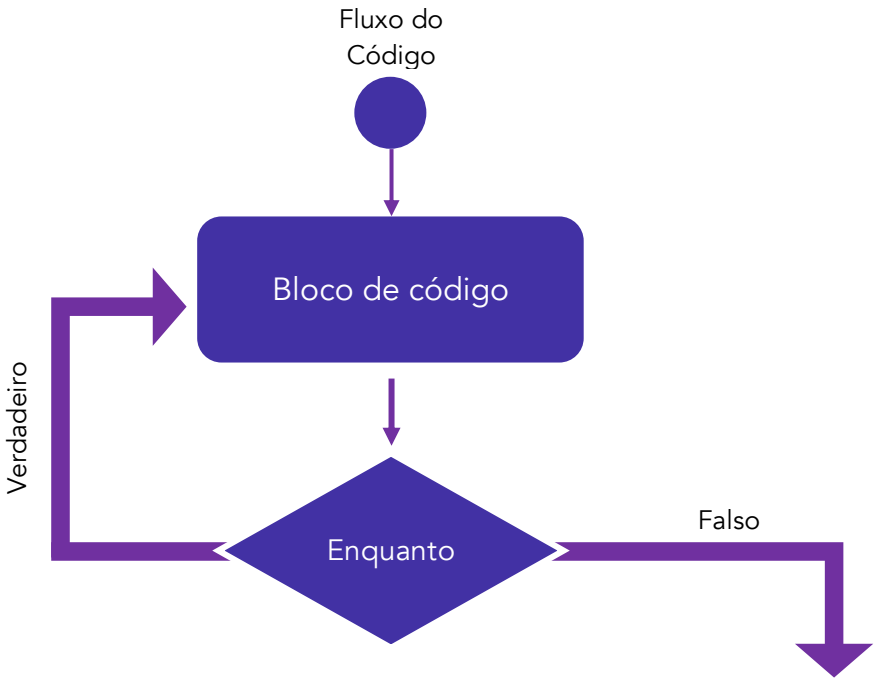
Perfeito. Quando usamos uma estrutura condicional, por exemplo o *Se...então..senão*, estamos fazendo uma verificação lógica - isso é, se o valor da sintaxe passada como parâmetro for verdadeiro, executaremos o bloco Se, se for falso, executaremos o Senão.

Gabarito: Correto

13. (FUNDATEC/CIGA SC/2023) Na linguagem Java, existe uma estrutura de controle de fluxo de execução que permite executar um bloco de código pelo menos uma vez e depois repeti-lo enquanto uma condição é verdadeira. Que estrutura é essa?

a) do-while
b) while
c) for
d) if-else
e) repeat-until

Comentários:
A estrutura que é executada ao menos uma vez, independentemente de ter sua condição de implementação verificada (se é verdadeira ou falsa), é o `Faça...enquanto` - ou *Do...while*.



Um adendo: a explicação também poderia ser respondida com o Repita...até, mas o Java não aceita nenhum comando desse gênero.

Gabarito: Letra D

14. (VUNESP/UNICAMP/2023) O programa a seguir, expresso na forma de uma pseudolinguagem (português estruturado), deve ser analisado e utilizado para responder às questões de números 23 e 24.

```
Início
Inteiro: a, b, c, i, M;
a ← 0;
b ← 2;
c ← 4;
M ← 10;
Para i de 1 até 5 faça
[
  Se (a < b)
      Então
      [
        a ← a + 1;
        c ← a + b;
      ]]
M ← M + a + b - c;
Imprima M;
Fim.
```

No programa, o número de vezes que a condição

Se (a < b)

dentro do loop Para... se mostra verdadeira é:

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

Comentários:

Temos um *loop* Para executado 5 vezes (de i = 1 até i = 5). A cada iteração, se a < b, faremos uma operação incremental em *a*, e atualizaremos o valor de *c* com o valor de a + b. Como para nossa análise só importa o valor de *a* e *b*, vamos ignorar as interações com a variável *c*. Vamos ver os laços:

Laço 1 - i = 1; a = 0, b = 2

Caímos no bloco Se, então, teremos:

- a = a + 1

- a = 0 + 1 = 1

Laço 2 - i = 2; a = 1; b = 2

Novamente, caímos no bloco Se. Teremos:

- a = a + 1
- a = 1 + 1 = 2

A partir do laço 3, teremos a = b (2), então não teremos mais o valor Verdadeiro para a condicional Se (a < b). Assim, podemos afirmar que o número de vezes que a condição é considerada verdadeira é igual a 2.

Gabarito: Letra B

---

15. (FUNDATEC/IFFAR/2023) Sobre a utilização de estruturas de repetição em algoritmos, assinale a alternativa que apresenta estrutura que executa um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida.

a) repita...até_que
b) faça...enquanto
c) caso...seja...faça...senão...fim_caso
d) enquanto...faça...fim_enquanto
e) para...de...até...faça...fim_para

Comentários:

A estrutura de repetição que sempre executará um bloco de código antes de verificar a condição, fazendo essa verificação após a execução, é o **`Repita...Até`**. Cuidado - o `Faça...enquanto` tem uma estrutura muito similar, também irá executar o bloco ao menos uma vez, mas faz a verificação antes da executá-lo.

Gabarito: Letra A

---

16. (CONSULPLAN/IF PA/2023) As estruturas de controle de fluxo são elementos fundamentais da lógica de programação. Assinale, a seguir, um exemplo de estrutura de controle de fluxo em lógica de programação.

a) Loop.
b) Classe.
c) Variável.
d) Constante.

Comentários:

Ferramentas de controle de fluxo são aquelas que interrompem o fluxo "normal" de leitura do código, o fluxo contínuo e sequencial. Uma das formas de controlarmos o fluxo é através das estruturas de repetição, ou loops. Com eles, um determinado bloco de código é executado *n* vezes, enquanto a condição for verdadeira ou conforme um contador.

Gabarito: Letra A

---

17. (CONSUPLAN/IF PA/2023) Qual das seguintes instruções de pseudocódigo seria utilizada para fazer o programa repetir um conjunto de ações até que uma determinada condição seja satisfeita?

a) SE...ENTÃO
b) PARA...ATÉ
c) CASO...SEJA
d) ENQUANTO...FAÇA

Comentários:

Uma estrutura, para repetir inúmeras vezes, deve se encaixar no conceito das estruturas de repetição. Dentre as alternativas, temos duas delas - o PARA...ATÉ, e o ENQUANTO...FAÇA. A estrutura que faz uma sequência de repetições até que determinada condição seja implementada é o ENQUANTO...FAÇA. O PARA...ATÉ usará um contador para ter uma quantidade definida de repetições.

Gabarito: Letra D

---

18. (FUNDATEC/IFC/2023) Em relação a algoritmos, analise as assertivas abaixo:

I. Um algoritmo representa uma sequência de regras.
II. Essas regras devem ser executadas em uma ordem preestabelecida.
III. Cada algoritmo possui um conjunto finito de regras.
IV. Essas regras devem possuir um significado e ser formalizadas segundo alguma convenção.

Quais estão corretas?

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

Comentários:

Vamos analisar cada item.

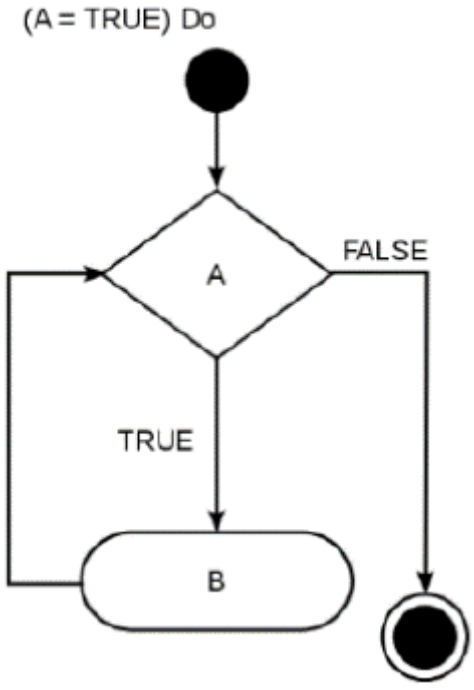
I. Certo. O algoritmo é uma sequência de regras que visa executar determinado programa.
II. Certo. A ordem corresponde ao fluxo de execução do código - usualmente sequencial.
III. Certo. Os algoritmos possuem regras para definir sua correta interpretação.
IV. Certo. Mesmo em pseudocódigo, temos algumas regras e padronizações que devem ser feitas e mantidas, imagine escrever cada função, cada loop de um jeito diferente num mesmo código, isso geraria uma confusão total.

Portanto, todos os itens estão corretos.

Gabarito: Letra E

---

19. (FUNCER/Pref. São Tomé/2023) Dentro da lógica de programação é uma estrutura que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador. A estrutura de repetição básica abaixo está se referindo:



a) do while.
b) for.
c) while.
d) if.

Comentários:

A estrutura apresentada inicia, em A, com uma verificação se determinado item possui o valor TRUE (verdadeiro) ou FALSE (falso). Se o valor for TRUE, executamos o bloco B e, em seguida, verificamos novamente se seu valor é verdadeiro ou falso. Se o valor for FALSE, seguimos na execução normal do código. Essa estrutura é chamada de WHILE, ou Enquanto.

Gabarito: Letra C

---

20. (CETAP/SEMAS PA/2023) Em que situação a recursividade é apropriada como solução para um problema de programação?

a) Quando o problema pode ser dividido em subproblemas independentes e semelhantes ao problema inicial.
b) Quando o problema é simples, ilógico e linear.
c) Quando o problema requer a utilização de laços de repetição.
d) Quando o problema é de natureza elétrica, o que demanda maior capacidade de processamento.
e) Quando o problema requer uma solução iterativa, ou seja, uma sequência de soluções que admitem certa margem de erro, mas que vão melhorando conforme novas iterações são executadas.

Comentários:

Vamos analisar cada alternativa.

a) Certo. A recursividade é a aplicação da "especialização do problema", já que dividimos uma função, que objetiva "resolver um problema", em subproblemas independentes a cada recursão.
b) Errado. A recursividade exige uma lógica mais complexa, lógica e não-linear.
c) Errado. Quando o problema requer laços de repetição, são usados os laços (estruturas de repetição).
d) Errado. Problema elétrico? Isso é problema para a engenharia elétrica, não para programação haha
e) Errado. Assim como na letra B, se a o problema requer iterações, usa-se uma estrutura que permita iterações - as estruturas de repetição.

Portanto, correta a alternativa A.

Gabarito: Letra A

---

21. (CEBRASPE/EMPREL/2023)

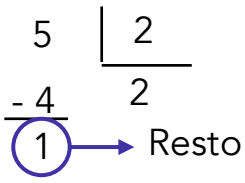
```
calc = 5 % 2
imprime(calc)
```

Assinale a opção que apresenta o resultado da execução do pseudocódigo precedente.

a) 1
b) 2
c) 10
d) 20
e) 25

Comentários:

A sintaxe 5%2 indica que estamos fazendo um módulo de 5 por 2, e seu retorno será o resto da divisão.



Portanto, o valor impresso será equivalente ao resto da divisão de 5 por 2, que é igual a 1.

Gabarito: Letra A

---

22. (CEBRASPE/Pref. Fortaleza/2023)

```
estrutura S
S.topo=0

estrutura-vazia(S){
   se (S.topo=0)
      retorna Verdadeiro;
   fim se
}

EMPURRA (S,x){
   S.topo = S.topo+1;
   S[S.topo] = x;
```

```
}

PUXA (S)
   se (Stack-Empty(S))
      escreva_erro("Underflow");
   senao
      S.topo = S.topo-1;
      retorna S[S.topo+1];
   fim se
```

Considerando a estrutura S precedente, inicialmente vazia e armazenada no arranjo S[1 ... 6], julgue o próximo item, a respeito de construção de algoritmos e estrutura de dados.

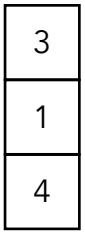
O resultado final das operações na sequência EMPURRA(S, 4), EMPURRA (S, 1), EMPURRA (S, 3), PUXA(S), EMPURRA (S, 8) e PUXA (S) é S[4,1,3,8].

Comentários:

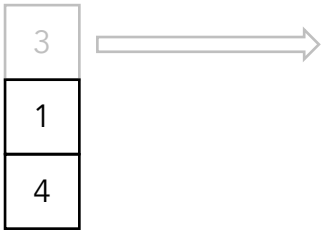
A questão traz algumas estruturas condicionais visando preencher uma pilha de dados. Temos a chamada da função EMPURRA e PUXA com os seguintes parâmetros:

- EMPURRA(S, 4)
- EMPURRA(S, 1)
- EMPURRA(S, 3)
- PUXA(S)
- EMPURRA(S, 8)
- PUXA(S)

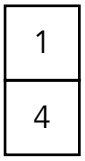
Basicamente, cada uma dessas funções EMPURRA irá alocar o número passado no parâmetro *x* no topo da pilha, enquanto as funções PUXA irão retirar o elemento do topo. Então, alocaremos, inicialmente, os números 4, 1 e 3 na pilha:



Em seguida, retiramos o valor do topo com PUXA(S).



Os próximos passos envolvem um EMPURRA e um PUXA em sequência e, portanto, não teremos alterações na pilha, já que o elemento vai ser adicionado e removido logo em seguida. Ao fim, ficamos com a seguinte estrutura:



Portanto, é incorreto afirmar que ficamos com S[4, 1, 3, 8] - ficaremos com S[4, 1].

Gabarito: Errado

---

23. (CEBRASPE/Pref. Fortaleza/2023) No que concerne a conceitos de algoritmos e blocos de comandos, julgue o item seguinte.

As funções são um bloco de código ou comandos constituindo um conjunto específico de instruções repetíveis, que recebem uma ou mais entradas e produzem alguma saída.

Comentários:

Perfeito! As funções trazem blocos de código, pequenos programas, que podem ser reutilizados em diversos pontos do código. Além disso, para ser caracterizada como uma função, a operação deve produzir alguma saída (retorno).

Gabarito: Correto

---

24. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Um loop que sempre se repetirá em um determinado número de vezes é representado pelo código SE ENTÃO SENÃO FIM SE.

Comentários:

O SE ENTÃO SENÃO é uma estrutura condicional, de seleção, e não um *loop*. Os *loops* englobam o PARA, ENQUANTO, FAÇA ENQUANTO, REPITA ATÉ, e outros. Portanto, incorreta a afirmativa.

Gabarito: Errado

---

25. (CEBRASPE/Pref. Fortaleza/2023) No que concerne a conceitos de algoritmos e blocos de comandos, julgue o item seguinte.

Algoritmo é uma lista de instruções que conduzem ações especificadas, passo a passo, em rotinas embasadas em hardware ou software.

Comentários:

Um algoritmo é, exatamente, uma lista de instruções que passará determinado valor de entrada por diversas ações e passos, que podem envolver rotinas embasadas em hardware (linguagens de baixo nível) ou em software (linguagens de alto nível). Então, correta a definição.

Gabarito: Correto

---

26. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Ao final da execução do algoritmo a seguir, o valor 0 será apresentado.

```
constante A = 50

enquanto (A > 0)
    A -= 5;
fim enquanto

escreva (A);
```

Comentários:

Essa questão tem algumas camadas de conhecimento sendo exploradas. Para começar, podemos afirmar errado pois a variável A é do tipo constante, ou seja, não poderá ser alterada. Porém, vamos ignorar esse fato por enquanto e vamos analisar a estrutura do Enquanto.

Temos uma condição A > 0, ou seja, enquanto A for maior que 0, iremos executar um bloco de código. O bloco de código é a operação A -= 5, que é uma operação de atribuição: cada vez que ela for chamada, retiraremos 5 unidades do valor de A. Portanto, iríamos reduzir o valor de 5 em 5 - teríamos 50, 45, 40... 0.

Veja que, apesar da condição ser A > 0, quando tivermos o valor de A = 5, a condição ainda será verdadeira e executaremos a operação A -= 5, resultando em A = 0. Nesse momento, como A possui o valor 0, a condição não será mais estabelecida e não executaremos mais o código - mas o valor do A que seria impresso realmente seria 0. Note a importância da ordem de leitura do código.

Porém, novamente, como estamos tratando de uma constante, não podemos alterar seu valor de forma externa - então o valor da operação escreva (A) (que é similar ao imprimir(A), que estamos fazendo até agora) será 50.

Gabarito: Errado

---

27. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

No algoritmo a seguir, o comando x = x + 10 é executado quatro vezes.

```
x = 10;
para (y = 40; y < 100; y = y + 16)
    x = x + 10;
fim para
```

Comentários:

O Para inicia com a variável em 40, com o limite superior menor que 100, e incrementa o valor em 16 unidades a cada laço. Então, teremos execuções para y igual a 40, 56, 72 e 84. A próxima iteração seria com y = 100, que passa a não obedecer mais à condição de verificação (y < 100). Portanto, de fato, executamos o comando 4 vezes.

Gabarito: Correto

---

28. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Se os valores a = 3, b = 4 e c = 8 forem entradas do algoritmo a seguir e o sistema no qual o algoritmo for executado utilizar números decimais com quatro casas de precisão, então a execução do referido algoritmo apresentará em tela o resultado 10.3923.

```
sp = (a + b + c)/2;
ar = sp*(sp - a)*(sp - b)*(sp - c);

se (ar < 0)
   escreva (“Não é possível obter resultado.”);

senão
   escreva (“Resultado: “);
   escreva(raiz_quadrada(ar));
fimse
```

Comentários:

Vamos desenvolver os cálculos.

sp = (3 + 4 + 8)/2 = 15/2 = 7.5
ar = 7.5 * (7.5 - 3) * (7.5 - 4) * (7.5 - 8) = -59.0625

Como temos um valor de ar menor que 0 (ar < 0), executaremos o bloco Se, tendo como retorno o texto “Não é possível obter resultado”, e não o valor 10.3923.

Gabarito: Errado

---

29. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Após executado, o algoritmo a seguir apresentará 720 como resultado final.

```
função f(x)
  se (x == 0 ou x == 1)
    retorna 1;
  fimse
  retorna f(x - 1)*x
fimfunção

função func(a)
  se (a == 0)
    retorna 2
```

```
  fimse
  retorna a + f(a - 1)
fimfunção

escreva(func(6));
```

Comentários:

Temos duas funções - uma função recursiva f(x) e outra função de ordem superior func(a). Estamos chamando primeiramente a função de ordem superior, com func(6). Então, vamos ver o que acontece.

Como o parâmetro a = 6, vamos cair no bloco alternativo - vamos retornar a + f(a - 1), ou seja, a + (f5). Sendo assim, precisamos subir para a função f(x) e resolvê-la com x = 5. E nela, novamente, como não atendemos a condição do Se, vamos executar o retorno - f(x-1) * x

Laço 1 - f(5) = f(4) * 5
Laço 2 - f(4) = f(3) * 4
Laço 3 - f(3) = f(2) * 3
Laço 4 - f(2) = f(1) * 2

Por fim, no laço 5 teremos f(1), e, como x == 1, caímos no bloco base - retornando 1. Como chegamos a um valor, podemos subir somando.

Laço 5 - f(1) = 1
Laço 4 - f(2) = f(1) * 2 = 1 * 2 = 2
Laço 3 - f(3) = f(2) * 3 = 2 * 3 = 6
Laço 2 - f(4) = f(3) * 4 = 6 * 4 = 24
Laço 1 - f(5) = f(4) * 5 = 24 * 5 = 120

Então, encontramos que f(5) é igual a 120. Então é essa a resposta... só que não! Agora temos que voltar para a função que foi chamada inicialmente, a func, nós paramos a execução do bloco com a + f(5). Portanto, teremos a + 120 = 120 + 6 = 126.

De toda a forma, a resposta não será 720 - portanto, a afirmativa está incorreta.

Gabarito: Errado

---

30. (CEBRASPE/Pref. Fortaleza/2023) A respeito de recursividade, julgue o próximo item.

Uma grande vantagem da utilização da recursividade é o baixo consumo de memória.

Comentários:

Muito pelo contrário - essa é uma desvantagem da recursividade, já que ela tem um alto consumo de memória. Iremos executar várias iterações dentro de uma só função, o que irá necessitar uma alocação maior de memória.

Gabarito: Errado

---

31. (CEBRASPE/Pref. Fortaleza/2023) Julgue o item que se segue, relativo a conceitos de avaliação de expressões.

Após o algoritmo a seguir ser executado, o valor da variável soma1 será maior que o da variável soma2.

```
vetor a[7];
real soma1, soma2;
inteiro i;

a = [1,3,9,27,81,243,729];}
soma1 = 0;
i = 0;

enquanto (i < 7) faça
   soma1 = soma1 + a[i]
   i = i + 1
fim enquanto

soma2 = 1 * (1-3^7)/(1-3)

escreva(soma1)
escreva(soma2)
```

Comentários:

A questão traz algumas variáveis, dentre elas um vetor a com 7 elementos. Vamos achar o valor de soma1 e soma2, para verificarmos o apontamento da afirmativa. Vamos começar com o soma1.

O loop Enquanto irá somando o valor atual de soma1 com a respectiva posição no vetor. Então, como i = 0, e o aumento a cada laço é incremental, faremos 7 loops (de i = 0 até i = 6). Vamos ver o primeiro laço:

Laço 1 - i = 0:

soma1 = soma1 + a[0]
soma1 = 0 + 1 = 1

Laço 2 - i = 1:
soma1 = soma1 + a[1]
soma1 = 1 + 3 = 4

Laço 3 - i = 2:
soma1 = soma1 + a[2]
soma1 = 4 + 9 = 13

Laço 4 - i = 3:
soma1 = soma1 + a[3]
soma1 = 13 + 27 = 40

Laço 5 - i = 4:
soma1= soma1 + a[4]
soma1 = 40 + 81 = 121

Laço 6 - i = 5
soma1 = soma1 + a[5]
soma1 = 121 + 243 = 364

Laço 7 - i = 6
soma1 = soma1 + a[6]
soma1 = 364 + 769 = 1039

Portanto, o valor final da variável soma1 é 1.039. Na hora da prova, se você tiver entendido a lógica - que estamos somando todos os valores do vetor - a resolução se torna muito mais rápido. Só iríamos somar os valores de a, e encontraríamos o mesmo valor.

Agora, vamos para a soma2.

soma2 = 1 * (1 - (3^7)) / (1-3)
soma2 = 1 * (1 - 2187) / (-2)
soma2 = 1 * (-2186) / (-2)
soma2 = 1 * 1.093
soma2 = 1.093

Então, o valor de soma2 (1.093) é maior que o valor de soma1 (1.039). Sendo assim, afirmativa incorreta.

Gabarito: Errado

32. (VUNESP/TJM SP/2023) Analise o programa a seguir, apresentado na forma de uma pseudolinguagem (Português Estruturado).

```
Início
  Inteiro: x, y, z, i;
  x ← 3;
  y ← 3;
  z ← 3;
  Para i de 1 até 6 faça
  [      Se (z = 3)
         Então
         [
            x ← x+1;
            y ← y+2;
            z ← z-1;
         ]
         Senão
         [
           z ← z+1;
         ] ]
   z ← x + y + z;
Fim.
```

O valor da variável z ao final da execução desse programa será:

a) 15
b) 16
c) 17
d) 18
e) 19

Comentários:

Não deixe o tamanho da sintaxe te assustar. Encare com calma, linha por linha, para entender o que está acontecendo. Iniciamos o código declarando 3 variáveis do tipo inteiro, x, y e z - todas com o valor igual a 3. Em seguida, iniciamos um loop `Para` numa notação um pouco diferente do que vimos, mas basicamente estamos iniciando em 1, indo até 6 com incrementos de uma unidade - ou seja, seria como Para (i = 1, i <= 6; i++) - ou seja, 6 laços.

O bloco de código executado a cada laço é um `Se...então...senão`. A condição `Se` é z = 3 e, como iniciamos o programa com z = 3, iniciaremos o bloco fazendo as operações elencadas no bloco `Se`. Vamos lá!

Laço 1 - i = 1; z = 3 (bloco Se);
x = x + 1 = 3 + 1; **x = 4**
y = y + 2 = 3 + 2; **y = 5**
z = z - 1 = 3 - 1; **z = 2** <- como caímos em z diferente de 3, o próximo bloco será no senão.

Laço 2 - i = 2; z = 2 (bloco Senão);
z = z + 1 = 2 + 1; **z = 3**

Laço 3 - i = 3; z = 3 (Bloco Se)
x = x + 1 = 4 +1; x = 5
y = y + 2 = 5 + 2; **y = 7**
z = z - 1 = 3 - 1; **z = 2**

Laço 4 - i = 4; z = 2 (bloco Senão)
z = z + 1 = 2 + 1; **z = 3**

Laço 5 - i = 5; z = 3 (bloco Se)
x = x + 1 = 5 + 1; **x = 6**
y = y + 2 = 7 + 2; **y = 9**
z = z - 1 = 3 - 1; **z = 2**

Laço 6 - i = 6; z = 2 (bloco Senão)
z = z + 1 = 2 + 1; **z = 3**

Bom, fizemos os laços, e terminamos com os seguintes valores:

x = 6
y = 9
z = 3

Por fim, temos uma nova atribuição de valor à variável z antes de finalizar o programa:

z = x + y + z
z = 6 + 9 + 3 = 18

Portanto, o valor final de Z será igual a 18.

Gabarito: Letra D

33. (IDCAP/CREA BR/2023) É uma maneira de resolver problemas decompondo-os repetidamente em subproblemas do mesmo tipo. Um exemplo clássico de uso desse tipo de algoritmo para resolver problemas é a Torre de Hanoi. O trecho acima diz respeito a(o):

a) Algoritmo de programação dinâmica.
b) Algoritmo recursivo.
c) Algoritmo backtracking.
d) Algoritmo de força bruta.
e) Algoritmo de divisão e conquista.

Comentários:

Quando decompomos um problema dentro de pequenos problemas menores, estamos fazendo o que é chamado de recursividade. Determinada função, por exemplo, passa a chamar a si diversas vezes, até que seja atingido um bloco base - que interrompe a necessidade da recursão.

Gabarito: Letra B

---

34. (FGV/TCE SP/2023) A série de Fibonacci é definida da seguinte forma: o primeiro e o segundo termos valem 1, e os demais são obtidos pela soma de seus dois antecessores. Em termos gráficos, ela define uma espiral, sendo utilizada em diversas áreas, que vão da biologia até o mercado financeiro. Um algoritmo para cálculo do termo de ordem n da série é apresentado a seguir.

```
algoritmo Fibonacci
var x, a, b, i, f: inteiro

início
leia(x);
a <- 1
b <- 1
f <- 1
i <- 2



enquanto i <= x faça
   f <- a + b
   a <- b
   b <- f
   i <- i + 1
fim enquanto
```

escreva("Fibonacci para o termo: ", f)

fim algoritmo

Executando o algoritmo, se for informado o valor 7 para x, será impressa a mensagem "Fibonacci para o termo: ": '

a) 7
b) 21
c) 28
d) 64
e) 128

Comentários:

Estamos executando a função leia(x) com x = 7. Com isso, executaremos o *loop* Enquanto com a condição i ≤ 7. Vamos fazer os laços.

LAÇO 1 - i = 2

- f = a + b; f = 1 + 1; f = 2
- a = b; a = 1
- b = f; b = 2
- i = i + 1; i = 2 + 1; i = 3

Um ponto de atenção - quando fazemos b = f, usamos o valor "atualizado" de f, já que ele foi modificado dentro do laço, antes de chamarmos a operação.

LAÇO 2 - i = 3

- f = a + b; f = 1 + 2; f = 3
- a = b; a = 2
- b = f; b = 3
- i = i + 1; i = 4

LAÇO 3 - i = 4

- f = a + b; f = 2 + 3; f = 5
- a = b; a = 3
- b = f; b = 5

- i = i + 1; i = 5

LAÇO 4 - i = 5

- f = a + b; f = 3 + 5; f = 8
- a = b; a = 5
- b = f; b = 8
- i = i + 1; i = 6

LAÇO 6 - i = 6

- f = a + b; f = 5 + 8; f = 13
- a = b; a = 8
- b = f; b = 13
- i = i + 1; i = 7

LAÇO 7 - i = 7

- f = a + b; f = 8 + 13; f = 21

Como estamos no último laço, os demais valores não nos importam. A saída, que envolve o valor de *f*, será, portanto, 21.

Gabarito: Letra B

35. (FGV/TCE SP/2023) Marta está definindo um algoritmo para descrever um menu de funções do sistema, apresentando as opções baseadas em números, seguido da leitura da opção, com a saída ocorrendo após a digitação do número zero.

Para gerenciar o fluxo de execução, que envolve a exibição do menu e leitura da opção, repetindo-se até que seja digitada a opção zero, Marta deve utilizar a estrutura de controle:

a) enquanto - faça;
b) se - então;
c) repita - até;
d) para - faça;
e) se - então - senão.

Comentários:

O fluxo de execução de um código é a ordem com que o interpretador lê esse código e executa operações. Estamos seguindo um fluxo de execução de bloco de código → verificação de

condição, até que determinada condição seja verificada como verdadeira. Essa estrutura se caracteriza como uma repetição do tipo **`Repita...Até`**.

Gabarito: Letra C

---

36. (VUNESP/TRF 3/2023) Um trecho de programa, expresso na forma de uma pseudolinguagem (Português Estruturado), é apresentado a seguir.

```
Início
   Inteiro: i, j, k, a;
    ...
   Para i de 3 até 6 faça
        Para j de 4 até 7 faça
            Para k de 2 até 8 faça
            [
                a ← a + 1;
             ]
     ...
Fim.
```

É correto afirmar que, a variável a, após a execução de desse trecho de programa, terá sido incrementada

a) 8 vezes.
b) 54 vezes.
c) 336 vezes.
d) 112 vezes.
e) 6 vezes.

Comentários:

Ótima questão para entendermos o funcionamento de *loops* aninhados. Para cada iteração do loop mais externo, serão executadas todas as iterações do *loop* interno, e do seu *loop* interno também. Então, basicamente, teremos:

- Iteração geral 1 - i = 3
    - Iteração bloco aninhado 1 - j = 4
        - Iterações internas, de k = 2 a k = 8 (ou seja, 7 iterações).
    - Iteração bloco aninhado 2 - j = 5
        - Iterações internas, de k = 2 a k = 8 (ou seja, 7 iterações).
    - Iteração bloco aninhado 3 - j = 6
        - Iterações internas, de k = 2 a k = 8 (ou seja, 7 iterações).

- Iteração bloco aninhado 4 - j = 7
  - Iterações internas, de k = 2 a k = 8 (ou seja, 7 iterações).

Veja que, para uma iteração do bloco externo, executamos 28 iterações internas. Como vamos de 3 a 6, isso é, temos 4 execuções, executaremos 4 x 28 = 112 iterações.

Uma forma mais direta de achar esse valor é multiplicar as vezes que cada bloco Para irá ocorrer - temos 7 no bloco interno, 4 no bloco intermediário e 4 no bloco externo. Então, 7 * 4 * 4 = 112.

Gabarito: Letra D

---

37. (CETREDE/UFC/2022) Marque a alternativa correta que apresenta o resultado do algoritmo detalhado a seguir.

```
 ALGORITMO
DECLARE n1, n2 NUMÉRICO
n1 ⟵ 6
n2 ⟵ aplica_processamento(n1)
ESCREVA (“O processamento aplicado em “, n1, “ resulta em “ n2)
FIM_ALGORITMO

SUB-ROTINA aplica_processamento(x NUMÉRICO)
   SE x = 1
        RETORNE x
   SENÃO
        RETORNE x*aplica_processamento(x-1)
FIM SUB-ROTINA aplica_processamento
```

a) 180
b) 90
c) 120
d) 720

Comentários:

O primeiro passo para entendermos a questão é entender qual valor está sendo passado para a sub-rotina. Conseguimos identificar nesse trecho:

```
n1 ⟵ 6
n2 ⟵ aplica_processamento(n1)
```

Ou seja, vamos chamar a função aplica_processamento com o valor de n1 - assim, x = 6. Como x = 6, caímos no bloco SENÃO, que traz uma função recursiva. Essa função será executada até chegarmos a x = 1, portanto:

- Laço 1 = 6 * aplica_processamento(5)
- Laço 2 = 5 * aplica_processamento(4)
- Laço 3 = 4 * aplica_processamento(3)
- Laço 4 = 3 * aplica_processamento(2)
- Laço 5 = 2 * aplica_processamento(1)
- Laço 6 → aplica_processamento(1) = 1

Agora, podemos voltar "subindo" os laços.

- Laço 6 = 1
- Laço 5 = 2 * 1 = 2
- Laço 4 = 3 * 2 = 6
- Laço 3 = 4 * 6 = 24
- Laço 2 = 5 * 24 = 120
- Laço 1 = 6 * 120 = 720

Portanto, o retorno da função, dada uma entrada de 6, será de 720.

Gabarito: Letra D

---

38. (CEBRASPE/PETROBRAS/2022) Com relação a tipos abstratos de dados, julgue o próximo item.

No trecho de código abaixo, o valor final da variável op é 4.

```
inteiro op=8
op = 5
op = 4
```

Comentários:

Temos uma declaração inicial da variável op, que é do tipo inteiro, e depois temos novas declarações. Essas novas declarações substituem o valor original - portanto, de fato, o valor final da variável será 4, equivalente à última atribuição de valor.

Gabarito: Correto

---

39. (CEBRASPE/PETROBRAS/2022) Julgue o item subsequente, a respeito de algoritmos para ordenação e pesquisa e de programação recursiva.

Uma função é dita recursiva quando, dentro dela, é feita uma ou mais chamada a ela mesma.

Comentários:

Veja que essa é uma cobrança recorrente - tanto do ponto de vista teórico, que corresponde a quebra do programa em subprogramas, quanto do ponto de vista prático, que corresponde a uma função chamando a si uma ou mais vezes. Portanto, correta a afirmativa.

Gabarito: Correto

---

40. (SELECON/AMAZUL/2022) Na construção de algoritmos estruturados são utilizadas estruturas de controle, como a conhecida por REPITA ... ATÉ ... FIM REPITA. Neste contexto, NÃO é uma afirmativa válida para essa estrutura de controle:

a) A condição de teste da estrutura é inserida no INÍCIO da estrutura de controle.
b) A condição de teste da estrutura é inserida no FIM da estrutura de controle.
c) A saída do loop ocorre quando o teste da condição de controle retoma valor VERDADEIRO .
d) A execução do programa permanece no loop se o teste da condição de controle retoma valor FALSO.

Comentários:

Vamos analisar cada alternativa.

a) Errado. A condição no bloco REPITA...ATÉ é implementada no fim da estrutura de controle, não no início.
b) Certo. Vide letra A.
c) Certo. A saída do ENQUANTO é feita quando o valor for VERDADEIRO.
d) Certo. Se o valor da condição for FALSO, permaneceremos no *loop* até que esse valor seja VERDADEIRO.

Sendo assim, a alternativa incorreta é a letra A.

Gabarito: Letra A

---

41. (VUNESP/ALESP/2022) É fornecido a seguir um algoritmo expresso na forma de uma pseudolinguagem (português estruturado).

```
Início
  Inteiro: x, y, z, j;
  Leia x, y;
  z ← 19;
  Para j de 1 até 3 faça
  [
     Se z > (x+y)
       Então
         z ← z - 3;
       Senão
         z ← z + 5;
  ]
  z ← z + z;
  Imprima z;
Fim.
```

Considerando os valores lidos para as variáveis x e y como sendo 8 e 9, respectivamente, então o valor impresso para a variável z no final do algoritmo é:

a) 42
b) 36
c) 32
d) 48
e) 40

Comentários:

Vamos executar o código com x = 8 e y = 9. Como x + y = 17 é menor que 19, caímos no bloco Se. Então, nosso primeiro laço será da seguinte forma:

LAÇO 1 - j = 1

- z = z - 3
- z = 19 - 3
- z = 16

LAÇO 2 - j = 2

Agora, vamos para o bloco Senão, já que z < x + y.

- z = z + 5

- z = 16 + 5
- z = 21

LAÇO 3 - j = 3

No último bloco, voltamos ao bloco Se, já que 21 > 17.

- z = z - 3
- z = 21 - 3
- z = 18

Chegamos ao valor de z, que é 18. Por fim, temos uma nova atribuição ao valor de z:

- z = z + z
- z = 18 + 18
- z = 36

Sendo assim, a resposta imprimida pelo comando é 36.

Gabarito: Letra B

---

42. (IFTO/IFTO/2022) Qual das definições abaixo melhor descreve o que é um algoritmo?

a) Uma linguagem de programação usada para desenvolvimento de aplicativos web.
b) Um protocolo usado na camada de rede para estabelecer uma comunicação entre dois dispositivos.
c) Tecnologia usada para enviar e receber dados do servidor sem precisar recarregar a página inteira.
d) Mecanismo para adicionar estilo a um documento web.
e) Uma sequência de passos que devem serem executados para alcançar um determinado objetivo.

Comentários:

Vamos analisar cada alternativa.

a) Errado. A alternativa descreve uma linguagem web, como o HTML e JavaScript.
b) Errado. Os protocolos são padrões para troca de informações entre diferentes computadores numa rede.
c) Errado. A alternativa descreve o funcionamento de APIs assíncronas.
d) Errado. Esses mecanismos são expressos em um arquivo chamado CSS.
e) Certo. A alternativa traz corretamente a definição de algoritmos.

Gabarito: Letra E

---

43. (SELECON/AMAZUL/2022) Na operação de soma dos números binários 10001010 + 01011111, o resultado será igual a:

a) 00101011
b) 10010110
c) 00110101
d) 11101001

Comentários:

Precisamos converter os números binários para a base decimal, de forma que seja possível fazer a soma. Temos:

- $10001010 = 2^1 + 2^3 + 2^7 = 138$
- $01011111 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^6 = 95$

Então, 95 + 138 = 233, que é expresso como 11101001 em base binária.

Gabarito: Letra D

---

44. (FGV/SEMSA Manaus/2022) Considere a função fat especificada abaixo em pseudocódigo, que recebe um número inteiro positivo e calcula o fatorial do mesmo de forma recursiva, devolvendo um número inteiro.

```
function fat (n as integer) as integer
    if n > 1
    then ...
    else return 1
    endif;
```

Para que a função fat funcione corretamente, o trecho ". . . " deve ser substituído por

a) return fat(n + 1)
b) return fat(n)
c) return fat(n-1)
d) return n * fat(n)
e) return n * fat(n-1)

Comentários:

A questão traz um programa com um bloco de código faltante. O objetivo é que seja calculado o fatorial do algoritmo de forma recursiva - isso nos indica que devemos chamar a própria função fat dentro do código.
Como não queremos uma recursão infinita, devemos ajeitar o parâmetro da função chamada para que ela chegue a obedecer a condição do bloco base (no else) - que será cumprida quando o valor de n ≤ 1. Portanto, a cada iteração devemos fazer fat(n-1). Assim, se n for igual a 4, por exemplo, iremos reduzi-lo uma unidade a cada laço, até chegarmos a n = 1.

Além disso, o fatorial é composto pela multiplicação de um número pelos seus predecessores - então, por exemplo, o fatorial de 3 (3!) será igual a 3 x 2 x 1. Sendo assim, precisamos passar o parâmetro atual para o cálculo - ficando, portanto, com n * fat(n-1) na recursão.

Vamos rodar n = 3, para ver se a recursão funciona:

- LAÇO 1 - 3 * fat(2)
- LAÇO 2 - 2 * fat(1)
- LAÇO 3 - fat(1) = 1

Subimos preenchendo os valores.

- LAÇO 2 - 2*1 = 2
- LAÇO 1 - 3*2 = 6

Assim, podemos afirmar que o algoritmo age de forma correta.

Gabarito: Letra E

---

45. (QUADRIX/CRF GO/2022) Acerca de programação, julgue o item.

Durante a execução de um programa, uma variável pode armazenar vários valores a cada instante, ou seja, ela pode armazenar até dez registros ao mesmo tempo.

Comentários:

"A cada instante" é uma constatação um pouco forte - mas, de fato, podemos frequentemente mudar o valor de uma variável, afinal, ela é variável. É o que acontece numa recursão, por exemplo. Porém, a variável irá armazenar apenas o valor atual, não múltiplos valores ao mesmo tempo - a não ser que seja uma variável de estrutura de dados. Mesmo assim, não teremos os limites de dez registros ao mesmo tempo. Portanto, podemos concluir que a afirmativa está incorreta.

Gabarito: Erado

---

46. (DIRENS/EEAR/2022) Relacione as colunas quanto às Estruturas de Controle do PHP. Em seguida, assinale a alternativa com a sequência correta.

1 – Executa um conjunto de instruções; baseado em um contador.
2 – Avalia a condição e, se for verdadeira, continua executando.
3 – Realiza uma avaliação, desvio condicional.

( ) IF
( ) FOR
( ) WHILE

a) 2 - 1 - 3
b) 1 - 2 - 3
c) 3 - 2 - 1
d) 3 - 1 - 2

Comentários:

Vamos identificar cada item.

1 – Executa um conjunto de instruções; baseado em um contador.
Trata-se do FOR (ou PARA, em português).

2 – Avalia a condição e, se for verdadeira, continua executando.
Descreve a estrutura do WHILE (ENQUANTO).

3 – Realiza uma avaliação, desvio condicional.
Traz a definição da condicional IF (SE...ENTÃO)

Portanto, temos 3 - 1 - 2.

Gabarito: Letra D

---

47. (IDECAN/UNILAB/2022) Observe o algoritmo a seguir:

```
1. escreva("Digite um numero....: ")
2. leia(n)
3. para i de 1 ate 18 faca
4. se vet[i] = n entao
5. p := i
```

6. fimse
7. fimpara

Na linha 5, qual a função da variável p?

a) Recebe o número digitado pelo usuário.
b) Recebe a posição no vetor vet do número encontrado.
c) Recebe o valor do número encontrado na posição i do vetor vet.
d) Recebe a soma dos números do vetor vet.

Comentários:

Na linha 5, temos o valor de *i* sendo atribuído à variável *p*. O *i* é a posição do vetor que estamos navegando - portanto, podemos afirmar que i recebe a posição no vetor *vet* do respectivo valor de i.

Gabarito: Letra B

---

48. (FUNDATEC/SBC/2022) Sobre os testes de condições em linguagem de programação, é correto afirmar que:

a) Um teste de condição pode ser realizado através da instrução while.
b) A componente else do if é obrigatório.
c) if-else a condição é avaliada e, caso seja verdadeira, é executada a instrução associada ao else.
d) if-else permite a existência de instruções que não são executadas em um programa.
e) if-else não pode ser executado de forma encadeada.

Comentários:

Vamos avaliar cada item.

a) Errado. O WHILE é uma instrução de repetição, não de condição.
b) Errado. O ELSE (equivalente ao Senão) é opcional.
c) Errado. Se a condição for verdadeira, executamos o bloco do IF.
d) Certo. O IF-ELSE é uma instrução de seleção, ou seja, de inúmeros blocos, somente um será executado.
e) Errado. Podemos ter IF-ELSE encadeados.

Portanto, a alternativa correta é a letra D.

Gabarito: Letra D

---

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

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

Comentários:

Perfeito! Os laços são as repetições, feitas na estrutura de repetição. Comandos como Enquanto, Repita...até, Faça...até e Para são formas de implementarmos essas estruturas.

Gabarito: Correto

---

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

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

Comentários:

Se e Senão são estruturas de seleção, condicionais, e não estruturas de repetição. De resto, a explicação está correta - se a condição for verdadeira, executaremos um bloco, caso contrário, executaremos o outro bloco.

Gabarito: Errado

---

51. (IBFC/AFEAM/2022) Dada a forma geral a seguir, existente em várias linguagens de programação, assinale a alternativa que apresenta corretamente a que tipo de estrutura se refere.

```
switch (expressão) {
case expressão_constante1:sentença1;
. . .
case constante_n: sentença_n;
[default: sentençan + 1]
}
```

a) Controle múltiplo
b) Repetição múltipla
c) Desvio múltiplo
d) Seleção múltipla

Comentários:

O SWITCH...CASE é uma forma em inglês de implementarmos a estrutura condicional ESCOLHA...CASO. Essa é uma estrutura de seleção múltipla, que permite a escolha de nenhum, um ou múltiplos casos, onde cada um comportará um bloco de código distinto.

Gabarito: Letra D

---

52. (IBFC/DPE MT/2022) Assinale a alternativas que esteja tecnicamente correta quanto ao pseudocódigo, em inglês, que representa uma estrutura de repetição.

a) Switch
b) Return
c) While
d) Function

Comentários:

Das estruturas apresentadas, a única que apresenta uma estrutura de repetição é o WHILE. A estrutura SWITCH é uma estrutura de seleção múltipla, RETURN é um comando para atribuir retorno a uma função, e FUNCTION é função, em inglês.

Gabarito: Letra C

---

53. (QUADRIX/SEE DF/2022) Acerca dos aspectos das linguagens de programação e das estruturas de dados e da programação orientada a objetos (POO), julgue o item a seguir.

Na lógica de programação, o operador lógico "e" possui a função de disjunção, enquanto o operador "não" possui a função de negação.

Comentários:

Cuidado! O "e" lógico, em programação, tem valor de **conjunção**, isso é, só será verdadeiro se todos os seus elementos também forem. Já o "não", de fato, é uma negação lógica.

Gabarito: Errado

---

54. (FCM/IF AM/2022) Sobre listas, pilhas e filas, associe corretamente as colunas

ESTRUTURAS DE DADOS
1 - Listas Lineares
2 - Pilha

3 - Fila

DESCRIÇÕES
( ) São utilizadas quando se deseja processar itens de acordo com a ordem “primeiro-que-chega, primeiro- atendido”.
( ) Também são chamadas listas LIFO.
( ) Existe uma ordem linear, que é a “ordem de chegada”.
( ) Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.
( ) Os itens são colocados um sobre o outro. O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.

A sequência que apresenta a associação correta é

a) 2, 3, 1, 2, 3.
b) 3, 2, 2, 1, 2.
c) 2, 2, 3, 1, 2.
d) 3, 2, 2, 1, 3.
e) 3, 2, 3, 1, 2.

Comentários:

Vamos analisar cada descrição, e encontrar o item associado.

( ) São utilizadas quando se deseja processar itens de acordo com a ordem “primeiro-que-chega, primeiro- atendido”.
A descrição se refere ao modelo FIFO, usado nas filas. (3)

( ) Também são chamadas listas LIFO.
LIFO, ou Last In First Out, caracterizam as pilhas. (2)

( ) Existe uma ordem linear, que é a “ordem de chegada”.
A ordem linear é presente nas filas, já que o primeiro elemento “a chegar” toma a frente da fila. (3)

( ) Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.
A questão descreve as listas - inclusive, essa é uma das principais características que diferem as listas dos vetores.

( ) Os itens são colocados um sobre o outro. O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.

O item descreve a estrutura das pilhas (2).

Portanto, ficamos com 3-2-3-1-2.

Gabarito: Letra E

---

55. (IDECAN/SEFAZ RR/2022) Quando criamos um programa de computador, utilizamos a seguinte sequência de operações na grande maioria das vezes: entrada de dados, processamento e saída. Selecione a estrutura de seleção que nunca testa uma ou mais variáveis de acordo com uma condição.

a) if
b) else if / elif
c) else
d) while
e) for

Comentários:

Quando estamos rodando uma condicional, temos o bloco do SE..ENTÃO, que irá analisar uma condição e executar determinar bloco de código se essa condição for verdadeira. Em seguida, podemos ter o bloco SENÃO (else), que, em casos de seleção binária, nunca irá fazer uma avaliação do valor da condição - somente será executada como um "caso residual".

Gabarito: Letra C

---

56. (IDECAN/SEFAZ RR/2022) Carlos trabalha como desenvolvedor de software e recebe a demanda de criar um trecho de código usando um loop de repetição que somente pode ser usado quando se sabe a quantidade de vezes que o loop vai acontecer. Selecione o loop de repetição que Carlos deve utilizar.

a) if
b) for
c) while
d) switch
e) do - while

Comentários:

O loop escolhido quando se sabe a quantidade de repetições que teremos é o loop Para - o FOR. Nele, teremos um contador, onde definiremos o valor inicial, final e a forma de incremento (ou decremento), com isso conseguimos definir a quantidade de repetições que teremos.

Gabarito: Letra B

---

57. (VUNESP/PC RR/2022) Analise o procedimento a seguir, expresso na forma de uma pseudolinguagem (português estruturado).

```
Início
   Inteiro: a1, a2, a3, i;
   Leia a1;
   a2 ← 1;
   a3 ← 5;
   Para i de 1 até 4 faça
    [
       Se (a1+a2) < a3
            Então
               a2 ← a2 + 1;
            Senão
              a3 ← a3 + 2;
     ]
     a1 ← a2 + a3;
     Imprima a1;
Fim.
```

Assumindo que o valor lido para a variável a1 tenha sido 3, então o resultado impresso ao final do procedimento para a variável a1 é igual a:

a) 3
b) 7
c) 9
d) 11
e) 14

Comentários:

Temos um *loop* para de 4 repetições (de i = 1, até i = 4). Nele, verificaremos um condição - vamos ver o que acontece se rodarmos esse programa com a1 = 3.

LAÇO 1 - i = 1:

Temos os seguintes valores:

- a1 + a2 = 3 + 1 = 4
- a3 = 5

Então, como a3 > (a1 + a2), executaremos o bloco SE.

- a2 = a2 + 1
- a2 = 1 + 1
- a2 = 2

LAÇO 2 - i = 2

Agora, temos a seguinte análise:

- a1 + a2 = 3 + 2 = 5
- a3 = 5

Agora, passamos a executar o bloco Senão, já que a soma é igual ou maior que a3. Então, teremos:

- a3 = a3 + 2
- a3 = 5 + 2
- a3 = 7

LAÇO 3 - i = 3

A situação agora é essa:

- a1 + a2 = 5
- a3 = 7

Voltamos ao bloco Então, já que a3 > (a1 + a2). Vamos executá-lo.

- a2 = a2 + 1
- a2 = 2 + 1
- a2 = 3

LAÇO 4 - i = 4

No último laço, temos o seguinte:

- a1 + a2 = 3 + 3 = 6
- a3 = 7

Continuamos no bloco Senão, já que a soma de (a1 + a2) não superou ou igual a3. Portanto:

- a2 = a2 + 1
- a2 = 3 + 1
- a2 = 4

Por fim, iremos imprimir a soma de a2 e a3 com seus valores finais - 4 e 7. A saída será 4 + 7 = 11.

Gabarito: Letra D

---

58. (VUNESP/PC RR/2022) Considere o trecho de um procedimento a seguir, expresso na forma de uma pseudolinguagem (português estruturado).

```
Inteiro: a, i, j, k, m;
...
a ← 0;
...
Para i de 1 até 3 faça
 [
    Para j de 1 até 3 faça
     [
       Para k de 1 até 3 faça
        [
          Para m de 1 até 3 faça
           [
             a ← a + 2;
           ]
        ]
    ]
]
```

Ao final da execução desse trecho de código, o valor presente na variável a será igual a:

a) 12
b) 27
c) 54
d) 81
e) 162

Comentários:

Vou lhe ensinar um jeito rápido de resolver essa questão. O total de laços executado será igual à multiplicação de laços ocorridos em cada bloco Para. Em todos os blocos teremos 3 execuções - portanto, 3 * 3 * 3 * 3 = 3^4 = 81. A cada execução do laço mais interno, teremos o incremento de 2 unidades ao valor da variável a - que parte de 0. Portanto, teremos 0, 2, 4, 6, 8... até completarmos os 81 laços. A saída final será, portanto, 81 * 2 = 162.

Gabarito: Letra E

---

59. (SS CENTEC/CENTEC/2022) endo que A=3, B=7, C=4 e D=B, marque a alternativa CORRETA de acordo com as afirmações abaixo.

I. (B + A) <= C
II. (A > C) AND (C <= D)
III. (A+B) > 10 OR (A+B) = (C+D)
IV. (A>=C) AND (D >= C)

a) Apenas II, III e IV.
b) Apenas, III e IV.
c) Apenas IV.
d) Todas afirmativas são verdadeiras.
e) Nenhuma alternativa é verdadeira.

Comentários:

Vamos analisar cada item - e assinalar aqueles com retorno verdadeiro.

I. (B + A) <= C
Falso. B + A = 10; C = 4. Portanto, temos 10 ≤ 4 - que é falso.

II. (A > C) AND (C <= D)
Falso. Temos duas operações. (A > C), ou (3 > 4) tem valor FALSO, e (C <= D), ou (4 ≤ 8) tem valor VERDADEIRO. Como estamos operando com o E lógico (AND), ambos os lados precisariam ser verdadeiros para termos o retorno verdadeiro.

III. (A+B) > 10 OR (A+B) = (C+D)
Falso. ((A + B) > 10), ou (10 > 10) é FALSO, e ((A + B) = (C + D)), ou (10 = 7) também é FALSO. Como ambos os lados do operador OR (ou lógico) são falsos, a afirmação é falsa.

IV. (A>=C) AND (D >= C)

Falso. Temos (A >= C), ou (3 ≥ 4), que é FALSO, e (D >= C), ou (7 ≥ 4), que é VERDADEIRO. Porém, na disjunção (AND, ou E lógico), precisamos de ambos os lados verdadeiros para termos um retorno verdadeiro - sendo assim, a afirmação é falsa.

Pela análise, concluímos que nenhuma alternativa é verdadeira.

Gabarito: Letra E

---

60. (QUADRIX/PRODAM/2022) A respeito das estruturas de repetição, assinale a alternativa que apresenta a estrutura que se consiste em um laço de repetição determinado, ou seja, utilizado quando é conhecida antecipadamente — antes de iniciado o laço — a quantidade de vezes em que os comandos devem ser executados.

a) DO WHILE
b) WHILE
c) FOR
d) IF
e) CASE

Comentários:

A estrutura que analisa a quantidade de vezes que determinada estrutura de repetição irá ser repetida antes da execução do código, baseado em um contador, é o PARA, ou FOR, em inglês.

Gabarito: Letra C

---

61. (CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

Os dados de um algoritmo devem ser definidos por tipos para que seus conteúdos possam ser submetidos a operações corretas, inerentes a cada tipo de dado.

Comentários:

A definição do tipo de dado de uma variável é essencial, já que cria restrições a o que pode ou não ser alocado nessa variável. Essa definição pode ser explícita ou implícita, a depender da tipagem da linguagem. Nesse sentido, correta a afirmativa.

Gabarito: Correto

---

62. (IGEDUC/CM V Santo Antão/2024) Julgue o item a seguir.

Na programação, os operadores lógicos são utilizados principalmente para operações matemáticas complexas, como cálculos de derivadas e integrais.

Esses operadores, incluindo adição, subtração, multiplicação e divisão, são fundamentais na construção de algoritmos para aplicações matemáticas avançadas.

Comentários:

Muito cuidado! Operadores lógicos são os operadores E, OU e NÃO, responsáveis por fazer comparações lógicas. Operadores matemáticos, como a soma e subtração, são responsáveis por inserir aplicações matemáticas. A afirmativa define operadores matemáticas, e não lógicos, e, por esse motivo, está incorreta.

Gabarito: Errado

---

63. (IGEDUC/CM V Santo Antão/2024) Julgue o item a seguir.

Na programação, o uso de variáveis e constantes é fundamental para o armazenamento e manipulação de dados. Variáveis podem ter seus valores alterados, enquanto constantes mantêm valores fixos durante a execução do programa. Dentre os tipos de dados comuns em programação estão inteiros (int), decimais (float, double), caracteres (char), strings e valores booleanos (boolean).

Comentários:

Perfeito! Dentro da programação, temos dois locais para alocação de valores na memória - as variáveis, que podem ter os valores alterados durante a execução dos códigos, e as constantes, que devem ter seus valores inalterados. E, de fato, os tipos de dado repassados pela questão estão corretos.

Gabarito: Errado

---

64. (CEBRASPE/INPI/2024) Acerca de estrutura de dados e algoritmos, julgue o item a seguir.

A passagem de um vetor por valor é mais eficiente que a passagem por parâmetro, considerando aspectos de tempo de processamento e espaço em memória, estando ambas as situações sob as mesmas condições de recursos.

Comentários:

Passar um vetor por valor implica em criar uma cópia do vetor original, o que pode ser muito ineficiente em termos de tempo de processamento e espaço de memória, especialmente se o vetor for grande. Isso ocorre porque cada elemento do vetor precisa ser copiado individualmente, o que pode levar a um aumento significativo no uso da memória e no tempo de processamento.

Por outro lado, passar um vetor por referência (ou seja, como um parâmetro) é geralmente mais eficiente, pois apenas a referência ao vetor original é passada. Isso significa que não é necessário criar uma cópia do vetor, economizando assim tempo de processamento e espaço de memória.

Portanto, em geral, a passagem de um vetor por referência é preferível à passagem por valor, especialmente quando se trabalha com vetores grandes. No entanto, é importante notar que passar um vetor por referência significa que qualquer alteração feita ao vetor dentro da função afetará o vetor original. Se isso não for desejado, então uma cópia do vetor deve ser feita de qualquer maneira.

Gabarito: Errado

---

65. (CESGRANRIO/UNEMAT/2024) Em linguagens de programação, o escopo sintático refere- se

a) à área do código onde uma variável pode ser referenciada.
b) à hierarquia de operadores utilizados para realizar operações em expressões.
c) às regras que determinam a forma correta das estruturas de controle de fluxo.
d) ao conjunto de palavras-chave reservadas da linguagem.
e) ao número máximo de parâmetros permitido em uma função, exclusivamente.

Comentários:

O escopo sintático em linguagens de programação engloba a área onde variáveis podem ser referenciadas. Isso significa que o escopo sintático determina as partes do código onde uma variável é acessível. As principais limitações de escopo são o escopo global e local.

Por exemplo, se uma variável é definida dentro de uma função, seu escopo é geralmente limitado a essa função. Isso significa que a variável pode ser referenciada (ou seja, lida ou modificada) apenas dentro dessa função. Se tentarmos referenciar a variável fora da função, o programa provavelmente gerará um erro, pois a variável está fora do seu escopo sintático.

Gabarito: Letra A

---

66. (CEBRASPE/POLC AL/2023) Julgue o item a seguir, acerca de funções e procedimentos.

As funções executam um ou mais comandos e sempre retornam um resultado para quem fez uma chamada à função.

Comentários:

Perfeito! As funções executam um ou mais blocos de código, com comandos, quase como "subprogramas". E, além disso, para ser caracterizado como uma função (e não um procedimento), sua execução deve sempre retornar algum resultado.

Gabarito: Errado

---

67. (FUNDATEC/IFC/2023) A _____ ocorre quando se armazena um valor em uma variável; quando deseja-se um desvio no fluxo de nosso código baseado em uma condição booleana, usa-se o comando _____; para realizar uma repetição de instrução, pode-se usar a instrução _____; a fim de não repetir códigos, tem a possibilidade de fazer uso de _____; por fim, para interromper um laço de repetição de forma abrupta, usa-se _____.

Assinale a alternativa que preenche, correta e respectivamente, as lacunas do trecho acima.

a) composição – if – do-while – funções – break
b) atribuição – test – repetir – funções – parar
c) atribuição – if – do-while – funções – break
d) persistência – test – for – métodos – stop
e) persistência – if – for – métodos – stop

Comentários:

Vamos preencher as lacunas:

- Quando armazenamos um valor em uma variável, estamos fazendo uma ATRIBUIÇÃO.
- O comando de desvio no fluxo, que fará uma seleção, é o IF.
- A estrutura de repetição pode ser o Para, Enquanto, ou outros - porém, nas alternativas, a única que traz uma estrutura correta é o FAÇA...ENQUANTO, ou DO-WHILE.
- Para interromper um laço usamos uma ferramenta chamada de QUEBRA, ou BREAK - que trará uma interrupção forçada do laço, mesmo que determinada condição ainda seja verdadeira.

Portanto, ficamos com os preenchimentos ATRIBUIÇÃO, IF, DO-WHILE, BREAK.

Gabarito: Letra C

---

68. (FUNCERN/CM Natal/2023) As variáveis de programação são fundamentais para o dia a dia do programador. Elas orientam o programa a executar operações. Uma variável em um programa é definida como

a) um valor fixo.
b) um local na memória para armazenar dados.
c) uma palavra-chave reservada.
d) um comando executável

Comandos:

Por definição, uma variável é um local na memória do programa onde armazenamos determinados dados, valores, para que possam ser reutilizadas ao longo do programa.

Gabarito: Letra B

---

69. (VUNESP/EPC/S2023) Considere que, ao se chamar uma função: a) foram passados os valores de variáveis para ela; b) o valor de cada variável na função chamadora é copiado nas variáveis fictícias correspondentes da função chamada; c) as alterações feitas nas variáveis fictícias na função chamada não têm efeito nos valores das variáveis reais na função chamadora.

Esse método é conhecido como chamada

a) cruzada.
b) exclusiva.
c) reversa.
d) por valor.
e) por referência.

Comentários:

Temos duas formas de atribuição - atribuições por valor, e por referência. Isso já nos permite eliminar as três primeiras alternativas. Quando passamos um valor de uma variável por valor, estamos fazendo uma cópia desse valor original, desassociando a referência com o destino. Dessa forma, podemos alterar a variável de origem do valor sem termos impactos na nova variável. E é exatamente essa abordagem a trazida pela questão.

Se tivéssemos uma variável por referência, teríamos apenas um ponteiro referenciando o dado original, e qualquer alteração nesse dado original traria alterações para a variável que o referencia.

Gabarito: Letra D

---

70. (VUNESP/TJ RS/2023) Considere o seguinte programa, na forma de uma pseudolinguagem (português estruturado).

```
Início
  Inteiro: p, q, S, cont;
  p ← 5;
  q ← 3;
  Para cont de 1 até 5 faça
  [
    Se (p+q) > 2*p
      Então
        p ← p-1;
      Senão
        q ← q+2;
    ]
    S ← p + q;
Fim.
```

Ao final da execução desse programa, o conteúdo da variável S será:

a) 8.
b) 9.
c) 10.
d) 11.
e) 12.

Comentários:

Questão padrão da Vunesp. Teremos um laço Para com 5 iterações - vamos resolvê-las.

LAÇO 1 - cont = 1:

- (p + q) > 2*p ⇒ 8 > 10 → FALSO. Então, executamos o bloco Senão.
- q = q + 2
- q = 3 + 2
- q = 5

LAÇO 2 - cont = 2:

- (p + q) > 2*p ⇒ 10 > 10 → FALSO. Então, executamos o bloco Senão.

- q = q + 2
- q = 5 + 2
- q = 7

LAÇO 3 - cont = 3:

- (p + q) > 2*p ⇒ 12 > 10 → VERDADEIRO. Executaremos o bloco Se.
- p = p - 1
- p = 5 - 1
- p = 4

LAÇO 4 - cont = 4:

- (p + q) > 2*p ⇒ 11 > 8 → VERDADEIRO. Executaremos o bloco Se.
- p = p - 1
- p = 4 - 1
- p = 3

LAÇO 5 - cont = 5:

- (p + q) > 2*p ⇒ 10 > 6 → VERDADEIRO. Executaremos o bloco Se.
- p = p - 1
- p = 3 - 1 = 2

Agora, basta somarmos o valor de p e q para encontrarmos o valor da variável S.

- S = p + q
- S = 2 + 7
- S = 9

Gabarito: Letra B

71. (VUNESP/DPE SP/2023) Os parâmetros que são passados para uma função de um programa são denominados parâmetros reais, enquanto que os parâmetros recebidos por uma função são denominados parâmetros formais. Neste contexto, em uma chamada de função por valor, os valores dos parâmetros reais

a) são copiados para os parâmetros formais da função, e as alterações nestes efetuadas dentro da função refletem-se em alterações nos parâmetros reais.
b) são copiados para os parâmetros formais da função, e as alterações nestes efetuadas dentro da função não implicam em alterações nos parâmetros reais.
c) não são copiados para os parâmetros formais da função, e as alterações efetuadas nestes dentro da função, refletem-se em alterações nos parâmetros reais.

d) não são copiados para os parâmetros formais da função, e as alterações efetuadas nestes dentro da função, não implicam em alterações nos parâmetros reais.
e) não são copiados para os parâmetros formais da função, e esta não pode realizar alterações em seus parâmetros formais.

Comentários:

Nas chamada de função por valor, os parâmetros são copiados para a função, e, graças a isso, eventuais alterações não impactam os valores originais, ou parâmetros reais. Nesse sentido, a alternativa que traz a explicação correta é a letra B.

Gabarito: Letra B

---

72. (CENTEC/SEDUC CE/2023) Uma variável é um espaço na memória do computador destinado a um dado que é alterado durante a execução do algoritmo. Para funcionar corretamente, as variáveis precisam ser definidas por nomes e tipos. Analise o código abaixo e assinale a resposta CORRETA, respectivamente do tipo primitivo da variável e o resultado, linha 09.

```
01 Programa {
02   funcao inicio() {
03     real resultado
04
05     resultado = 5.0 + 4.0 * 2.0
06     escreva(resultado)
07
08     resultado = (5.0 + 4.0) * 2.0
09     escreva(resultado)
10
11 resultado = 1.0 + 2.0 / 3.0 * 4.0
12 escreva(resultado)
13   }
14 }
```

a) Real e 18.0
b) Cadeia e 22.0
c) Real e 40.0
d) Inteiro e 20.0
e) Real e 22.0

Comentários:

Na linha 9, estamos escrevendo o resultado da função passada logo acima, na linha 8. Vamos resolvê-la:

- resultado = (5.0 * 4.0) * 2.0
- resultado = 9.0 * 2.0
- resultado = 18.0

Então, o resultado será 18.0 e o número do tipo real - detalhe que, apesar do resultar ser um número inteiro (18), a presença de uma casa decimal nos traz o tipo real.

Gabarito: Letra A

---

73. (IBFC/TRF 5/2024) Estruturas de dados são constantemente utilizadas em algoritmos para resolução de problemas, desde os mais simples aos mais complexos, desta forma, estrutura de dados utiliza o princípio "Último a entrar, primeiro a sair"(LIFO):

a) Fila
b) Lista Encadeada
c) Pilha
d) Árvore

Comentários:

A estrutura de dados que traz a abordagem LIFO, ou seja, que numa eventual remoção, o elemento do topo, aquele adicionado mais recentemente, será retirado, é a pilha.

Gabarito: Letra C

---

74. (CEBRASPE/INPI/2024) Acerca de estrutura de dados e algoritmos, julgue o item a seguir.

Pilhas são tipos de estruturas de dados que permitem a remoção direta de qualquer elemento de sua estrutura.

Comentários:

As pilhas são caracterizadas pela abordagem LIFO - ou seja, as remoções só podem ser feitas no elemento que está no topo da pilha, elemento adicionado mais recentemente. Não é possível retirar outros elementos. Portanto, a afirmativa está incorreta.

Gabarito: Errado

---

75. (CEBRASPE/SEPLAN RR/2023) Julgue o item a seguir acerca dos conceitos de estrutura de dados.

Sempre que houver uma remoção na estrutura de dados denominada fila, o elemento removido será aquele que está na estrutura há mais tempo.

Comentários:

Perfeito! Na estrutura em fila, o elemento retirado será aquele que está alocado na frente da fila, que é composta justamente do elemento mais antigo que foi adicionado ao conjunto de dados - assim como uma fila indiana. Portanto, correta a afirmativa.

Gabarito: Correto

---

76. (VUNESP/TCM SP/2023) Considere uma estrutura de dados do tipo pilha, inicialmente vazia, que possui as operações típicas de inserção e remoção de elementos, denominadas PUSH e POP.

Nessa estrutura, foram executadas as seguintes operações, nesta ordem.

PUSH 1
PUSH 2
POP
PUSH 3
POP
PUSH 4
POP
PUSH 5

Após a realização de todas essas operações, o número de elementos na pilha e o valor armazenado no topo da pilha serão, respectivamente,

a) 0 e 0.
b) 1 e 1.
c) 1 e 5.
d) 2 e 1.
e) 2 e 5.

Comentários:

Como estamos numa estrutura de PILHA, as operações PUSH, um "Empilhar", em português, irão alocar determinado número no topo da pilha. Já as operações de POP irão desempilhar o dado do topo, isso, é, removê-lo do conjunto. Analisando as ações, temos:

PUSH 1, PUSH 2 $\Rightarrow$ [1, 2]
POP $\Rightarrow$ [1]
PUSH 3 $\Rightarrow$ [1, 3]

POP ⇒ [1]
PUSH 4 ⇒ [1, 4]
POP ⇒ [1]
PUSH 5 ⇒ [1, 5]

Portanto, a pilha será composta dos elementos [1, 5] - mas cuidado, essa não é a nossa resposta. Queremos o número de elementos (2) e o valor armazenado no topo (5) - portanto, a resposta será 2 e 5.

Gabarito: Letra E

---

77. (VUNESP/Pref. Marília/2023) Considere uma estrutura de dados com a propriedade de que, sempre que houver a remoção de um elemento nela armazenado, o elemento a ser removido é aquele que se encontra armazenado na estrutura há menos tempo.

Essa é a definição aderente a uma estrutura de dados denominada

a) fila.
b) pilha.
c) lista simples.
d) lista encadeada.
e) lista duplamente encadeada.

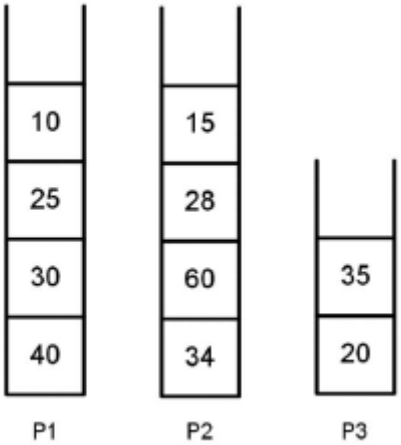
Comentários:

Quando, numa remoção, removemos o elemento mais recente do conjunto, estamos usando a abordagem LIFO - característica das pilhas.

Gabarito: Letra B

---

78. (CESGRANRIO/BB/2023) A Figura a seguir exibe o conteúdo de três pilhas: P1, P2 e P3.



Admita que um método Java, chamado exibePilha, receba essas três pilhas como parâmetros e execute os seguintes passos:

1. Cria duas pilhas auxiliares, A1 e A2, inicialmente vazias;
2. Remove um elemento de P1 e o insere em A1. Em seguida, remove um elemento de P2 e o insere em A1. Repete esses dois procedimentos até que P1 e P2 fiquem, ambas, vazias;
3. Remove um elemento de P3 e o insere em A1. Repete esse procedimento até que P3 fique vazia;
4. Remove um elemento de A1 e o insere em A2. Repete esse procedimento até que A1 fique vazia;
5. Remove um elemento de A2 e o exibe no console. Repete esse procedimento 4 vezes.

O que será exibido no console, quando o método exibePilha for executado, tendo P1, P2 e P3 sido passadas como parâmetros?

a) 10 15 25 28
b) 10 25 30 40
c) 15 10 28 25
d) 20 35 34 40
e) 40 34 30 60

Comentários:

Vamos analisar os passos, para construir as pilhas A1 e A2.

> 1. Cria duas pilhas auxiliares, A1 e A2, inicialmente vazias;

Essa ação apenas cria as duas pilhas.

> 2. Remove um elemento de P1 e o insere em A1. Em seguida, remove um elemento de P2 e o insere em A1. Repete esses dois procedimentos até que P1 e P2 fiquem, ambas, vazias;

Vamos fazer, de forma intercalada:

- Tirar o topo de P1 e inserir em A1
- Tirar o topo de P2 e inserir em A1

Portanto, A1 ficará com os valores [10, 15, 25, 28, 30, 60, 40, 34]

> 3. Remove um elemento de P3 e o insere em A1. Repete esse procedimento até que P3 fique vazia;

Esvaziando a P3 para A1, teremos A1 = [10, 15, 25, 28, 30, 60, 40, 34, 35, 20]

4. Remove um elemento de A1 e o insere em A2. Repete esse procedimento até que A1 fique vazia;

Esse procedimento faz com que A2 seja o inverso de A1. Então teremos A2 = [20, 35, 34, 40, 60, 30, 28, 25, 15, 10]

5. Remove um elemento de A2 e o exibe no console. Repete esse procedimento 4 vezes.

Vamos remover os 4 elementos do topo da pilha (numa representação horizontal, são os 4 últimos elementos) - vou destacá-los em negrito. A2 = [20, 35, 34, 40, 60, 30, 28, 25, 15, 10]

Topo da pilha

A saída será, portanto, [10, 15, 25, 28].

Gabarito: Letra A

---

79. (FGV/TCE SP/2023) Marcos é um estudante de programação de computadores e encontrou o algoritmo apresentado a seguir em seus estudos.

```
algoritmo Marcos
var
i, a: inteiro
v: vetor [1..5] de inteiro
início
a <- 0

para i de 1 até 5 faça:
  leia(v[i]);
fim para

para i de 1 até 5 faça:
   v[i] <- v[i] + v[6-i]
   a <- a + v[i]
fim para

escreva(a)
fim algoritmo
```

Considerando o vetor com índice inicial 1 e final 5, e utilizando os valores {2, 1, 2, 1, 2}, Marcos obterá a impressão do valor:

a) 3
b) 8
c) 9
d) 16
e) 19

Comentários:

Estamos rodando o vetor inicialmente com os valores {2, 1, 2, 1, 2}, com índice iniciando a contagem em 1. O primeiro Para do código irá ler o elemento de cada posição da array, é apenas uma forma do código saber o número que está sendo trabalhado. O foco aqui é no segundo Para:

```
para i de 1 até 5 faça:
    v[i] <- v[i] + v[6-i]
    a <- a + v[i]
fim para
```

Faremos 5 laços, vamos lá.

LAÇO 1 - i = 1:

- v[1] = v[1] + v[6-1]
- v[1] = 2 + 2
- **v[1] = 4**

- a = a + v[1]
- a = 0 + 4
- **a = 4**

LAÇO 2 - i = 2

- v[2] = v[2] + v[4]
- v[2] = 1 + 1
- **v[2] = 2**

- a = a +v[2]
- a = 4 + 2
- **a = 6**

LAÇO 3 - i = 3

- v[3] = v[3] + v[3]
- v[3] = 2 + 2
- v[3] = 4

- a = a + v[3]
- a = 6 + 4
- a = 10

LAÇO 4 - i = 4

Aqui um cuidado, o valor de v[6-4] = v[2], será consultado conforme alteração feita no LAÇO 2 - portanto, seu valor será igual a 2, não o valor original de v[1], que seria 1.

- v[4] = v[4] + v[2]
- v[4] = 1 + 2
- v[4] = 3

- a = a + v[4]
- a = 10 + 3
- a = 13

LAÇO 5 - i = 5

- v[5] = v[5] + v[1]
- v[5] = 2 + 4
- v[5] = 6

- a = a + v[5]
- a = 13 + 6
- a = 19

Assim, terminamos com a variável a = 19, e com o vetor v = {4, 2, 4, 3, 6}. Perceba que esses valores alterados no vetor são comunicados aos laços - muita atenção. Como a questão quer o valor de *a*, a resposta será 19.

Gabarito: Letra E

80. (SUGEP UFRPE/UFRPE/2022) A estrutura de dados "vetor" (array) é um arranjo unidimensional que pode acomodar múltiplos dados. Sobre essas estruturas de dados, assinale a alternativa incorreta.

a) Os dados de um vetor são mapeados numa área contígua da memória.
b) Os dados de um vetor são do mesmo tipo.
c) Cada um dos dados de um vetor pode ser acessado informando-se o identificador do vetor e o inteiro que indica a ordem do dado na sequência.
d) Os dados de um vetor são armazenados na memória ordenadamente, em modo crescente.
e) Pode-se atribuir um dado a um elemento de qualquer posição do vetor, independentemente do que foi atribuído aos demais elementos.

Comentários:

Vamos analisar cada item.

a) Correto. No armazenamento em memória, os vetores (ou arrays) são armazenados de forma contígua, ou seja, contínua, um ao lado do outro.
b) Correto. Os vetores são estruturas homogêneas, portanto só aceita um mesmo tipo de dado.
c) Correto. A forma de acessar elementos num vetor é feita através de um índice - sua posição relativa no conjunto de dados.
d) Errado. Eles são armazenados de forma ordenada, o que quer dizer que as posições são crescentes, mas não seus valores.
e) Correto. Como trabalhamos com índices, podemos acessar qualquer posição, diferentemente de estruturas como filas e pilhas.

Sendo assim, a alternativa incorreta é a letra D.

Gabarito: Letra D

QUESTÕES COMENTADAS

01. (INQC/CPTRANS/2024) Observe o seguinte algoritmo:

```
Algoritmo Maior
Var
  num1, num2, maior: inteiro:
Inicio
   Leia (num1, num2);
   Se (num1>num2) então
      maior ← num1;
   senão
      maior ← num2;
   fimse;
   escreva (maior);
Fim
```

A ferramenta utilizada para construção desse algoritmo é:

a) diagrama de Nassi-Shneiderman
b) diagrama hierárquico de fluxo
c) pseudocódigo
d) fluxograma

02. (CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

O pseudocódigo consiste em um texto estruturado com comandos escritos em linguagem humana, no qual se apoia a criação dos algoritmos computacionais.

03. (CEBRASPE/CAU BR/2024) Com relação à lógica de programação, julgue o próximo item.

A estrutura de controle IF, que pode ser classificada como do tipo iteração, determina o caminho que o algoritmo deve seguir, de acordo com determinada condição.

04. (IGEDUC/CM V Santo Antão/2024) Julgue o item a seguir.

O pseudocódigo é considerado uma linguagem de programação formal e executável, seguindo uma sintaxe e semântica específicas, semelhante a linguagens como Java e Python. Sua principal aplicação é na implementação direta de algoritmos em ambientes de desenvolvimento integrados, sem a necessidade de conversão para outra linguagem de programação.

05. (CEBRASPE/TST/2024)

```
programa principal
    inteiro i, contagem = 10, limite = 10;
    para (i = 0; i > limite; i = i + 1) faça
        contagem = contagem – 1;
    fim para
    imprime(contagem);
fim programa
```

Com base no algoritmo precedente, escrito em pseudocódigo, assinale a opção que corresponde ao tipo de estrutura em que se realiza o decréscimo da variável contagem.

a) estrutura de controle
b) estrutura de repetição
c) estrutura condicional
d) atributo

06. (CEBRASPE/TST/2024) Uma das vantagens do uso de funções predefinidas é

a) o reaproveitamento de código.
b) a passagem de parâmetros.
c) o recebimento do retorno das funções.
d) o grau de abstração do sistema.
e) a facilidade de implementação das funções.

07. (IBFC/TRF 5/2024) Sobre Estruturas de Seleção/Condição, analise as afirmativas e dê valores Verdadeiro (V) ou Falso (F).

( ) A instrução else é obrigatória em todas as estruturas de condição.
( ) A instrução else em uma estrutura de condição é executada quando a condição no if é avaliada como falsa.
( ) Estruturas de condição não são necessárias em programação, pois é possível realizar todas as operações sem avaliar condições.

Assinale a alternativa que apresenta a sequência correta de cima para baixo.

a) F - F - F
b) F - V - F
c) V - F - V

d) V - V - V

08. (CEBRASPE/FINEP/2024) Assinale a opção que corresponde ao comando de salto incondicional utilizado em programação estruturada.

a) goto
b) struct
c) if
d) else
e) for

09. (CEBRASPE/POLC AL/2023) Julgue o próximo item, no que se refere a estruturas de controle de fluxo.

```
 // codigo 1
i = 1
fim = 7
enquanto i < fim faça
escreva(i)
i = i + 1
fim enquanto
depois = i

// codigo 2
fim = 6
i = 0
enquanto i < fim faça
  i = i + 1
  escreva(i)
fim enquanto
depois = i
```

Considerando a estrutura precedente, é correto afirmar que o código 1 e o código 2 apresentam o mesmo resultado.

10. (VUNESP/TCM SP/2023) Analise o algoritmo a seguir, apresentado na forma de uma pseudolinguagem.

```
Início
[
  Tipo MAT = matriz[1..3,1..3] de inteiros;
  MAT: M;
```

```
Inteiro: i, j, X, Y;
X ← 0;
Y ← 0;
Para i de 1 até 3 faça
[
   Para j de 1 até 3 faça
   [
      Se i = j
        Então M[i,j] ← i + j + 1;
        Senão M[i,j] ← i + 2*j + 1;
   ]
]
Para i de 1 até 3 faça
[
  X ← X + M[i,i];
  Y ← Y + M[1,i];
]
Imprima (X+Y);
] Fim.
```

Ao final do algoritmo é impressa a soma (X+Y) que é igual a

a) 29.
b) 30.
c) 31.
d) 32.
e) 33.

11. (COCP IFMT/IFMT/2023) Segundo Manzano & Oliveira (2016), um laço incondicional, entre outras coisas, é um tipo de laço que é iterativo, porém não é interativo e tem seu funcionamento controlado por uma variável denominada contador. Com essa descrição, o autor se refere ao laço do tipo:

Fonte: (MANZANO, José Augusto & OLIVEIRA, Jayr Figueiredo de. Algoritmos: Lógica para desenvolvimento de programação de computadores. 18 ed. São Paulo: Erica, 2016).

a) para/fim_para
b) repita/até_que
c) enquanto/fim_enquanto
d) laço/fim_laço
e) execute/enquanto_for

12. (CEBRASPE/CNMP/2023) Julgue o item subsecutivo, que se referem a conceitos de programação e banco de dados.

As estruturas condicionais são utilizadas para testar variáveis lógicas.

13. (FUNDATEC/CIGA SC/2023) Na linguagem Java, existe uma estrutura de controle de fluxo de execução que permite executar um bloco de código pelo menos uma vez e depois repeti-lo enquanto uma condição é verdadeira. Que estrutura é essa?

a) do-while
b) while
c) for
d) if-else
e) repeat-until

14. (VUNESP/UNICAMP/2023) O programa a seguir, expresso na forma de uma pseudolinguagem (português estruturado), deve ser analisado e utilizado para responder às questões de números 23 e 24.

```
Início
Inteiro: a, b, c, i, M;
a ← 0;
b ← 2;
c ← 4;
M ← 10;
Para i de 1 até 5 faça
[
  Se (a < b)
      Então
      [
        a ← a + 1;
        c ← a + b;
      ]]
M ← M + a + b - c;
Imprima M;
Fim.
```

No programa, o número de vezes que a condição

Se (a < b)

dentro do loop Para... se mostra verdadeira é:

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

15. (FUNDATEC/IFFAR/2023) Sobre a utilização de estruturas de repetição em algoritmos, assinale a alternativa que apresenta estrutura que executa um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida.

a) repita...até_que
b) faça...enquanto
c) caso...seja...faça...senão...fim_caso
d) enquanto...faça...fim_enquanto
e) para...de...até...faça...fim_para

16. (CONSULPLAN/IF PA/2023) As estruturas de controle de fluxo são elementos fundamentais da lógica de programação. Assinale, a seguir, um exemplo de estrutura de controle de fluxo em lógica de programação.

a) Loop.
b) Classe.
c) Variável.
d) Constante.

17. (CONSUPLAN/IF PA/2023) Qual das seguintes instruções de pseudocódigo seria utilizada para fazer o programa repetir um conjunto de ações até que uma determinada condição seja satisfeita?

a) SE...ENTÃO
b) PARA...ATÉ
c) CASO...SEJA
d) ENQUANTO...FAÇA

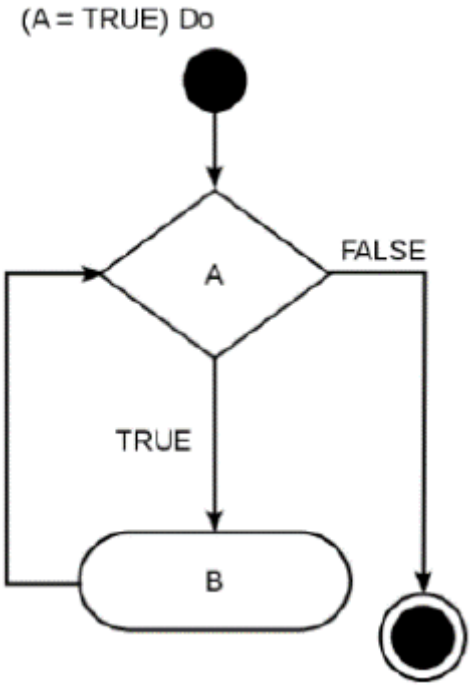
18. (FUNDATEC/IFC/2023) Em relação a algoritmos, analise as assertivas abaixo:

I. Um algoritmo representa uma sequência de regras.
II. Essas regras devem ser executadas em uma ordem preestabelecida.
III. Cada algoritmo possui um conjunto finito de regras.
IV. Essas regras devem possuir um significado e ser formalizadas segundo alguma convenção.

Quais estão corretas?

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

19. (FUNCER/Pref. São Tomé/2023) Dentro da lógica de programação é uma estrutura que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador. A estrutura de repetição básica abaixo está se referindo:



a) do while.
b) for.
c) while.
d) if.

20. (CETAP/SEMAS PA/2023) Em que situação a recursividade é apropriada como solução para um problema de programação?

a) Quando o problema pode ser dividido em subproblemas independentes e semelhantes ao problema inicial.
b) Quando o problema é simples, ilógico e linear.
c) Quando o problema requer a utilização de laços de repetição.

d) Quando o problema é de natureza elétrica, o que demanda maior capacidade de processamento.
e) Quando o problema requer uma solução iterativa, ou seja, uma sequência de soluções que admitem certa margem de erro, mas que vão melhorando conforme novas iterações são executadas.

21. (CEBRASPE/EMPREL/2023)

```
calc = 5 % 2
imprime(calc)
```

Assinale a opção que apresenta o resultado da execução do pseudocódigo precedente.

a) 1
b) 2
c) 10
d) 20
e) 25

22. (CEBRASPE/Pref. Fortaleza/2023)

```
estrutura S
S.topo=0

estrutura-vazia(S){
   se (S.topo=0)
      retorna Verdadeiro;
   fim se
}

EMPURRA (S,x){
   S.topo = S.topo+1;
   S[S.topo] = x;
}

PUXA (S)
   se (Stack-Empty(S))
      escreva_erro(“Underflow”);
   senao
```

```
    S.topo = S.topo-1;
    retorna S[S.topo+1];
fim se
```

Considerando a estrutura S precedente, inicialmente vazia e armazenada no arranjo S[1 ... 6], julgue o próximo item, a respeito de construção de algoritmos e estrutura de dados.

O resultado final das operações na sequência EMPURRA(S, 4), EMPURRA (S, 1), EMPURRA (S, 3), PUXA(S), EMPURRA (S, 8) e PUXA (S) é S[4,1,3,8].

23. (CEBRASPE/Pref. Fortaleza/2023) No que concerne a conceitos de algoritmos e blocos de comandos, julgue o item seguinte.

As funções são um bloco de código ou comandos constituindo um conjunto específico de instruções repetíveis, que recebem uma ou mais entradas e produzem alguma saída.

24. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Um loop que sempre se repetirá em um determinado número de vezes é representado pelo código SE ENTÃO SENÃO FIM SE.

25. (CEBRASPE/Pref. Fortaleza/2023) No que concerne a conceitos de algoritmos e blocos de comandos, julgue o item seguinte.

Algoritmo é uma lista de instruções que conduzem ações especificadas, passo a passo, em rotinas embasadas em hardware ou software.

26. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Ao final da execução do algoritmo a seguir, o valor 0 será apresentado.

```
constante A = 50

enquanto (A > 0)
    A -= 5;
fim enquanto

escreva (A);
```

27. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

No algoritmo a seguir, o comando x = x + 10 é executado quatro vezes.

```
x = 10;
para (y = 40; y < 100; y = y + 16)
    x = x + 10;
fim para
```

28. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Se os valores a = 3, b = 4 e c = 8 forem entradas do algoritmo a seguir e o sistema no qual o algoritmo for executado utilizar números decimais com quatro casas de precisão, então a execução do referido algoritmo apresentará em tela o resultado 10.3923.

```
sp = (a + b + c)/2;
ar = sp*(sp - a)*(sp - b)*(sp - c);

se (ar < 0)
   escreva (“Não é possível obter resultado.”);

senão
   escreva (“Resultado: “);
   escreva(raiz_quadrada(ar));
fimse
```

29. (CEBRASPE/Pref. Fortaleza/2023) Com relação a estruturas de controle, seleção, repetição e desvio, julgue o item a seguir.

Após executado, o algoritmo a seguir apresentará 720 como resultado final.

```
função f(x)
  se (x == 0 ou x == 1)
     retorna 1;
  fimse
  retorna f(x - 1)*x
fimfunção
```

```
função func(a)
  se (a == 0)
    retorna 2
  fimse
  retorna a + f(a - 1)
fimfunção

escreva(func(6));
```

30. (CEBRASPE/Pref. Fortaleza/2023) A respeito de recursividade, julgue o próximo item.

Uma grande vantagem da utilização da recursividade é o baixo consumo de memória.

31. (CEBRASPE/Pref. Fortaleza/2023) Julgue o item que se segue, relativo a conceitos de avaliação de expressões.

Após o algoritmo a seguir ser executado, o valor da variável soma1 será maior que o da variável soma2.

```
vetor a[7];
real soma1, soma2;
inteiro i;

a = [1,3,9,27,81,243,729];}
soma1 = 0;
i = 0;

enquanto (i < 7) faça
   soma1 = soma1 + a[i]
   i = i + 1
fim enquanto

soma2 = 1 * (1-3^7)/(1-3)

escreva(soma1)
escreva(soma2)
```

32. (VUNESP/TJM SP/2023) Analise o programa a seguir, apresentado na forma de uma pseudolinguagem (Português Estruturado).

```
Início
  Inteiro: x, y, z, i;
```

```
x ← 3;
y ← 3;
z ← 3;
Para i de 1 até 6 faça
[       Se (z = 3)
        Então
        [
           x ← x+1;
           y ← y+2;
           z ← z-1;
        ]
        Senão
        [
           z ← z+1;
        ] ]
   z ← x + y + z;
Fim.
```

O valor da variável z ao final da execução desse programa será:

a) 15
b) 16
c) 17
d) 18
e) 19

33. (IDCAP/CREA BR/2023) É uma maneira de resolver problemas decompondo-os repetidamente em subproblemas do mesmo tipo. Um exemplo clássico de uso desse tipo de algoritmo para resolver problemas é a Torre de Hanoi. O trecho acima diz respeito a(o):

a) Algoritmo de programação dinâmica.
b) Algoritmo recursivo.
c) Algoritmo backtracking.
d) Algoritmo de força bruta.
e) Algoritmo de divisão e conquista.

34. (FGV/TCE SP/2023) A série de Fibonacci é definida da seguinte forma: o primeiro e o segundo termos valem 1, e os demais são obtidos pela soma de seus dois antecessores. Em termos gráficos, ela define uma espiral, sendo utilizada em diversas áreas, que vão da biologia até o mercado financeiro. Um algoritmo para cálculo do termo de ordem n da série é apresentado a seguir.

algoritmo Fibonacci

```
var x, a, b, i, f: inteiro

início
leia(x);
a <- 1
b <- 1
f <- 1
i <- 2

enquanto i <= x faça
   f <- a + b
   a <- b
   b <- f
   i <- i + 1
fim enquanto
escreva("Fibonacci para o termo: ", f)
fim algoritmo
```

Executando o algoritmo, se for informado o valor 7 para x, será impressa a mensagem "Fibonacci para o termo: ": '

a) 7
b) 21
c) 28
d) 64
e) 128

35. (FGV/TCE SP/2023) Marta está definindo um algoritmo para descrever um menu de funções do sistema, apresentando as opções baseadas em números, seguido da leitura da opção, com a saída ocorrendo após a digitação do número zero.

Para gerenciar o fluxo de execução, que envolve a exibição do menu e leitura da opção, repetindo-se até que seja digitada a opção zero, Marta deve utilizar a estrutura de controle:

a) enquanto - faça;
b) se - então;
c) repita - até;
d) para - faça;
e) se - então - senão.

36. (VUNESP/TRF 3/2023) Um trecho de programa, expresso na forma de uma pseudolinguagem (Português Estruturado), é apresentado a seguir.

```
Início
   Inteiro: i, j, k, a;
    ...
   Para i de 3 até 6 faça
       Para j de 4 até 7 faça
           Para k de 2 até 8 faça
           [
               a ← a + 1;
            ]
    ...
Fim.
```

É correto afirmar que, a variável a, após a execução de desse trecho de programa, terá sido incrementada

a) 8 vezes.
b) 54 vezes.
c) 336 vezes.
d) 112 vezes.
e) 6 vezes.

37. (CETREDE/UFC/2022) Marque a alternativa correta que apresenta o resultado do algoritmo detalhado a seguir.

```
 ALGORITMO
DECLARE n1, n2 NUMÉRICO
n1 ⟵  6
n2 ⟵ aplica_processamento(n1)
ESCREVA (“O processamento aplicado em “, n1, “ resulta em “ n2)
FIM_ALGORITMO

SUB-ROTINA aplica_processamento(x NUMÉRICO)
   SE x = 1
        RETORNE x
   SENÃO
        RETORNE x*aplica_processamento(x-1)
FIM SUB-ROTINA aplica_processamento
```

a) 180
b) 90
c) 120

d) 720

38. (CEBRASPE/PETROBRAS/2022) Com relação a tipos abstratos de dados, julgue o próximo item.

No trecho de código abaixo, o valor final da variável op é 4.

inteiro op=8
op = 5
op = 4

Comentários:

Temos uma declaração inicial da variável op, que é do tipo inteiro, e depois temos novas declarações. Essas novas declarações substituem o valor original - portanto, de fato, o valor final da variável será 4, equivalente à última atribuição de valor.

Gabarito: Correto

---

39. (CEBRASPE/PETROBRAS/2022) Julgue o item subsequente, a respeito de algoritmos para ordenação e pesquisa e de programação recursiva.

Uma função é dita recursiva quando, dentro dela, é feita uma ou mais chamada a ela mesma.

40. (SELECON/AMAZUL/2022) Na construção de algoritmos estruturados são utilizadas estruturas de controle, como a conhecida por REPITA ... ATÉ ... FIM REPITA. Neste contexto, NÃO é uma afirmativa válida para essa estrutura de controle:

a) A condição de teste da estrutura é inserida no INÍCIO da estrutura de controle.
b) A condição de teste da estrutura é inserida no FIM da estrutura de controle.
c) A saída do loop ocorre quando o teste da condição de controle retoma valor VERDADEIRO.
d) A execução do programa permanece no loop se o teste da condição de controle retoma valor FALSO.

41. (VUNESP/ALESP/2022) É fornecido a seguir um algoritmo expresso na forma de uma pseudolinguagem (português estruturado).

```
Início
  Inteiro: x, y, z, j;
  Leia x, y;
  z ← 19;
```

```
  Para j de 1 até 3 faça
  [
     Se z > (x+y)
       Então
         z ← z - 3;
       Senão
         z ← z + 5;
  ]
  z ← z + z;
  Imprima z;
Fim.
```

Considerando os valores lidos para as variáveis x e y como sendo 8 e 9, respectivamente, então o valor impresso para a variável z no final do algoritmo é:

a) 42
b) 36
c) 32
d) 48
e) 40

42. (IFTO/IFTO/2022) Qual das definições abaixo melhor descreve o que é um algoritmo?

a) Uma linguagem de programação usada para desenvolvimento de aplicativos web.
b) Um protocolo usado na camada de rede para estabelecer uma comunicação entre dois dispositivos.
c) Tecnologia usada para enviar e receber dados do servidor sem precisar recarregar a página inteira.
d) Mecanismo para adicionar estilo a um documento web.
e) Uma sequência de passos que devem serem executados para alcançar um determinado objetivo.

43. (SELECON/AMAZUL/2022) Na operação de soma dos números binários 10001010 + 01011111, o resultado será igual a:

a) 00101011
b) 10010110
c) 00110101
d) 11101001

44. (FGV/SEMSA Manaus/2022) Considere a função fat especificada abaixo em pseudocódigo, que recebe um número inteiro positivo e calcula o fatorial do mesmo de forma recursiva, devolvendo um número inteiro.

```
function fat (n as integer) as integer
    if n > 1
    then ...
    else return 1
    endif;
```

Para que a função fat funcione corretamente, o trecho ". . . " deve ser substituído por

a) return fat(n + 1)
b) return fat(n)
c) return fat(n-1)
d) return n * fat(n)
e) return n * fat(n-1)

45. (QUADRIX/CRF GO/2022) Acerca de programação, julgue o item.

Durante a execução de um programa, uma variável pode armazenar vários valores a cada instante, ou seja, ela pode armazenar até dez registros ao mesmo tempo.

46. (DIRENS/EEAR/2022) Relacione as colunas quanto às Estruturas de Controle do PHP. Em seguida, assinale a alternativa com a sequência correta.

1 – Executa um conjunto de instruções; baseado em um contador.
2 – Avalia a condição e, se for verdadeira, continua executando.
3 – Realiza uma avaliação, desvio condicional.

( ) IF
( ) FOR
( ) WHILE

a) 2 - 1 - 3
b) 1 - 2 - 3
c) 3 - 2 - 1
d) 3 - 1 - 2

47. (IDECAN/UNILAB/2022) Observe o algoritmo a seguir:

```
1. escreva("Digite um numero....: ")
2. leia(n)
3. para i de 1 ate 18 faca
4. se vet[i] = n entao
5. p := i
6. fimse
7. fimpara
```

Na linha 5, qual a função da variável p?

a) Recebe o número digitado pelo usuário.
b) Recebe a posição no vetor vet do número encontrado.
c) Recebe o valor do número encontrado na posição i do vetor vet.
d) Recebe a soma dos números do vetor vet.

48. (FUNDATEC/SBC/2022) Sobre os testes de condições em linguagem de programação, é correto afirmar que:

a) Um teste de condição pode ser realizado através da instrução while.
b) A componente else do if é obrigatório.
c) if-else a condição é avaliada e, caso seja verdadeira, é executada a instrução associada ao else.
d) if-else permite a existência de instruções que não são executadas em um programa.
e) if-else não pode ser executado de forma encadeada.

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

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

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

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

51. (IBFC/AFEAM/2022) Dada a forma geral a seguir, existente em várias linguagens de programação, assinale a alternativa que apresenta corretamente a que tipo de estrutura se refere.

switch (expressão) {
case expressão_constante1:sentença1;
. . .
case constante$_n$: sentença_n;
[default: sentençan + 1]
}

a) Controle múltiplo
b) Repetição múltipla
c) Desvio múltiplo
d) Seleção múltipla

52. (IBFC/DPE MT/2022) Assinale a alternativas que esteja tecnicamente correta quanto ao pseudocódigo, em inglês, que representa uma estrutura de repetição.

a) Switch
b) Return
c) While
d) Function

53. (QUADRIX/SEE DF/2022) Acerca dos aspectos das linguagens de programação e das estruturas de dados e da programação orientada a objetos (POO), julgue o item a seguir.

Na lógica de programação, o operador lógico "e" possui a função de disjunção, enquanto o operador "não" possui a função de negação.

54. (FCM/IF AM/2022) Sobre listas, pilhas e filas, associe corretamente as colunas

ESTRUTURAS DE DADOS
1 - Listas Lineares
2 - Pilha
3 - Fila

DESCRIÇÕES
( ) São utilizadas quando se deseja processar itens de acordo com a ordem "primeiro-que-chega, primeiro- atendido".
( ) Também são chamadas listas LIFO.
( ) Existe uma ordem linear, que é a "ordem de chegada".
( ) Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.

( ) Os itens são colocados um sobre o outro. O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.

A sequência que apresenta a associação correta é

a) 2, 3, 1, 2, 3.
b) 3, 2, 2, 1, 2.
c) 2, 2, 3, 1, 2.
d) 3, 2, 2, 1, 3.
e) 3, 2, 3, 1, 2.

55. (IDECAN/SEFAZ RR/2022) Quando criamos um programa de computador, utilizamos a seguinte sequência de operações na grande maioria das vezes: entrada de dados, processamento e saída. Selecione a estrutura de seleção que nunca testa uma ou mais variáveis de acordo com uma condição.

a) if
b) else if / elif
c) else
d) while
e) for

56. (IDECAN/SEFAZ RR/2022) Carlos trabalha como desenvolvedor de software e recebe a demanda de criar um trecho de código usando um loop de repetição que somente pode ser usado quando se sabe a quantidade de vezes que o loop vai acontecer. Selecione o loop de repetição que Carlos deve utilizar.

a) if
b) for
c) while
d) switch
e) do - while

57. (VUNESP/PC RR/2022) Analise o procedimento a seguir, expresso na forma de uma pseudolinguagem (português estruturado).

```
Início
   Inteiro: a1, a2, a3, i;
   Leia a1;
   a2 ← 1;
   a3 ← 5;
   Para i de 1 até 4 faça
```

```
    [
       Se (a1+a2) < a3
           Então
              a2 ← a2 + 1;
           Senão
             a3 ← a3 + 2;
     ]
     a1 ← a2 + a3;
     Imprima a1;
Fim.
```

Assumindo que o valor lido para a variável a1 tenha sido 3, então o resultado impresso ao final do procedimento para a variável a1 é igual a:

a) 3
b) 7
c) 9
d) 11
e) 14

58. (VUNESP/PC RR/2022) Considere o trecho de um procedimento a seguir, expresso na forma de uma pseudolinguagem (português estruturado).

```
Inteiro: a, i, j, k, m;
...
a ← 0;
...
Para i de 1 até 3 faça
 [
     Para j de 1 até 3 faça
      [
        Para k de 1 até 3 faça
         [
           Para m de 1 até 3 faça
            [
              a ← a + 2;
            ]
         ]
    ]
]
```

Ao final da execução desse trecho de código, o valor presente na variável a será igual a:

a) 12
b) 27
c) 54
d) 81
e) 162

59. (SS CENTEC/CENTEC/2022) endo que A=3, B=7, C=4 e D=B, marque a alternativa CORRETA de acordo com as afirmações abaixo.

I. (B + A) <= C
II. (A > C) AND (C <= D)
III. (A+B) > 10 OR (A+B) = (C+D)
IV. (A>=C) AND (D >= C)

a) Apenas II, III e IV.
b) Apenas, III e IV.
c) Apenas IV.
d) Todas afirmativas são verdadeiras.
e) Nenhuma alternativa é verdadeira.

60. (QUADRIX/PRODAM/2022) A respeito das estruturas de repetição, assinale a alternativa que apresenta a estrutura que se consiste em um laço de repetição determinado, ou seja, utilizado quando é conhecida antecipadamente — antes de iniciado o laço — a quantidade de vezes em que os comandos devem ser executados.

a) DO WHILE
b) WHILE
c) FOR
d) IF
e) CASE

61. (CEBRASPE/CAU BR/2024) Julgue o item a seguir, relativo a algoritmos de linguagem de programação.

Os dados de um algoritmo devem ser definidos por tipos para que seus conteúdos possam ser submetidos a operações corretas, inerentes a cada tipo de dado.

62. (IGEDUC/CM V Santo Antão/2024) Julgue o item a seguir.

Na programação, os operadores lógicos são utilizados principalmente para operações matemáticas complexas, como cálculos de derivadas e integrais.

Esses operadores, incluindo adição, subtração, multiplicação e divisão, são fundamentais na construção de algoritmos para aplicações matemáticas avançadas.

63. (IGEDUC/CM V Santo Antão/2024) Julgue o item a seguir.

Na programação, o uso de variáveis e constantes é fundamental para o armazenamento e manipulação de dados. Variáveis podem ter seus valores alterados, enquanto constantes mantêm valores fixos durante a execução do programa. Dentre os tipos de dados comuns em programação estão inteiros (int), decimais (float, double), caracteres (char), strings e valores booleanos (boolean).

64. (CEBRASPE/INPI/2024) Acerca de estrutura de dados e algoritmos, julgue o item a seguir.

A passagem de um vetor por valor é mais eficiente que a passagem por parâmetro, considerando aspectos de tempo de processamento e espaço em memória, estando ambas as situações sob as mesmas condições de recursos.

65. (CESGRANRIO/UNEMAT/2024) Em linguagens de programação, o escopo sintático refere- se

a) à área do código onde uma variável pode ser referenciada.
b) à hierarquia de operadores utilizados para realizar operações em expressões.
c) às regras que determinam a forma correta das estruturas de controle de fluxo.
d) ao conjunto de palavras-chave reservadas da linguagem.
e) ao número máximo de parâmetros permitido em uma função, exclusivamente.

66. (CEBRASPE/POLC AL/2023) Julgue o item a seguir, acerca de funções e procedimentos.

As funções executam um ou mais comandos e sempre retornam um resultado para quem fez uma chamada à função.

67. (FUNDATEC/IFC/2023) A _____ ocorre quando se armazena um valor em uma variável; quando deseja-se um desvio no fluxo de nosso código baseado em uma condição booleana, usa-se o comando _____; para realizar uma repetição de instrução, pode-se usar a instrução _____; a fim de não repetir códigos, tem a possibilidade de fazer uso de _____; por fim, para interromper um laço de repetição de forma abrupta, usa-se _____.

Assinale a alternativa que preenche, correta e respectivamente, as lacunas do trecho acima.

a) composição – if – do-while – funções – break
b) atribuição – test – repetir – funções – parar
c) atribuição – if – do-while – funções – break
d) persistência – test – for – métodos – stop
e) persistência – if – for – métodos – stop

68. (FUNCERN/CM Natal/2023) As variáveis de programação são fundamentais para o dia a dia do programador. Elas orientam o programa a executar operações. Uma variável em um programa é definida como

a) um valor fixo.
b) um local na memória para armazenar dados.
c) uma palavra-chave reservada.
d) um comando executável

69. (VUNESP/EPC/S2023) Considere que, ao se chamar uma função: a) foram passados os valores de variáveis para ela; b) o valor de cada variável na função chamadora é copiado nas variáveis fictícias correspondentes da função chamada; c) as alterações feitas nas variáveis fictícias na função chamada não têm efeito nos valores das variáveis reais na função chamadora.

Esse método é conhecido como chamada

a) cruzada.
b) exclusiva.
c) reversa.
d) por valor.
e) por referência.

70. (VUNESP/TJ RS/2023) Considere o seguinte programa, na forma de uma pseudolinguagem (português estruturado).

```
Início
  Inteiro: p, q, S, cont;
  p ← 5;
  q ← 3;
  Para cont de 1 até 5 faça
  [
     Se (p+q) > 2*p
       Então
          p ← p-1;
       Senão
          q ← q+2;
```

```
    ]
    S ← p + q;
Fim.
```

Ao final da execução desse programa, o conteúdo da variável S será:

a) 8.
b) 9.
c) 10.
d) 11.
e) 12.

71. (VUNESP/DPE SP/2023) Os parâmetros que são passados para uma função de um programa são denominados parâmetros reais, enquanto que os parâmetros recebidos por uma função são denominados parâmetros formais. Neste contexto, em uma chamada de função por valor, os valores dos parâmetros reais

a) são copiados para os parâmetros formais da função, e as alterações nestes efetuadas dentro da função refletem-se em alterações nos parâmetros reais.
b) são copiados para os parâmetros formais da função, e as alterações nestes efetuadas dentro da função não implicam em alterações nos parâmetros reais.
c) não são copiados para os parâmetros formais da função, e as alterações efetuadas nestes dentro da função, refletem-se em alterações nos parâmetros reais.
d) não são copiados para os parâmetros formais da função, e as alterações efetuadas nestes dentro da função, não implicam em alterações nos parâmetros reais.
e) não são copiados para os parâmetros formais da função, e esta não pode realizar alterações em seus parâmetros formais.

72. (CENTEC/SEDUC CE/2023) Uma variável é um espaço na memória do computador destinado a um dado que é alterado durante a execução do algoritmo. Para funcionar corretamente, as variáveis precisam ser definidas por nomes e tipos. Analise o código abaixo e assinale a resposta CORRETA, respectivamente do tipo primitivo da variável e o resultado, linha 09.

```
01 Programa {
02   funcao inicio() {
03     real resultado
04
05     resultado = 5.0 + 4.0 * 2.0
06     escreva(resultado)
07
08     resultado = (5.0 + 4.0) * 2.0
09     escreva(resultado)
```

```
10
11 resultado = 1.0 + 2.0 / 3.0 * 4.0
12 escreva(resultado)
13    }
14 }
```

a) Real e 18.0
b) Cadeia e 22.0
c) Real e 40.0
d) Inteiro e 20.0
e) Real e 22.0

73. (IBFC/TRF 5/2024) Estruturas de dados são constantemente utilizadas em algoritmos para resolução de problemas, desde os mais simples aos mais complexos, desta forma, estrutura de dados utiliza o princípio "Último a entrar, primeiro a sair"(LIFO):

a) Fila
b) Lista Encadeada
c) Pilha
d) Árvore

74. (CEBRASPE/INPI/2024) Acerca de estrutura de dados e algoritmos, julgue o item a seguir.

Pilhas são tipos de estruturas de dados que permitem a remoção direta de qualquer elemento de sua estrutura.

75. (CEBRASPE/SEPLAN RR/2023) Julgue o item a seguir acerca dos conceitos de estrutura de dados.

Sempre que houver uma remoção na estrutura de dados denominada fila, o elemento removido será aquele que está na estrutura há mais tempo.

76. (VUNESP/TCM SP/2023) Considere uma estrutura de dados do tipo pilha, inicialmente vazia, que possui as operações típicas de inserção e remoção de elementos, denominadas PUSH e POP.

Nessa estrutura, foram executadas as seguintes operações, nesta ordem.

PUSH 1
PUSH 2
POP
PUSH 3

POP
PUSH 4
POP
PUSH 5

Após a realização de todas essas operações, o número de elementos na pilha e o valor armazenado no topo da pilha serão, respectivamente,

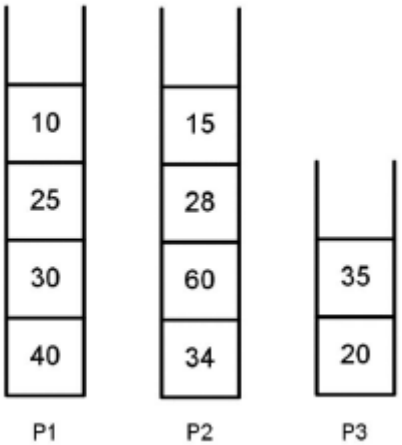
a) 0 e 0.
b) 1 e 1.
c) 1 e 5.
d) 2 e 1.
e) 2 e 5.

77. (VUNESP/Pref. Marília/2023) Considere uma estrutura de dados com a propriedade de que, sempre que houver a remoção de um elemento nela armazenado, o elemento a ser removido é aquele que se encontra armazenado na estrutura há menos tempo.

Essa é a definição aderente a uma estrutura de dados denominada

a) fila.
b) pilha.
c) lista simples.
d) lista encadeada.
e) lista duplamente encadeada.

78. (CESGRANRIO/BB/2023) A Figura a seguir exibe o conteúdo de três pilhas: P1, P2 e P3.



Admita que um método Java, chamado exibePilha, receba essas três pilhas como parâmetros e execute os seguintes passos:

1. Cria duas pilhas auxiliares, A1 e A2, inicialmente vazias;
2. Remove um elemento de P1 e o insere em A1. Em seguida, remove um elemento de P2 e o insere em A1. Repete esses dois procedimentos até que P1 e P2 fiquem, ambas, vazias;
3. Remove um elemento de P3 e o insere em A1. Repete esse procedimento até que P3 fique vazia;
4. Remove um elemento de A1 e o insere em A2. Repete esse procedimento até que A1 fique vazia;
5. Remove um elemento de A2 e o exibe no console. Repete esse procedimento 4 vezes.

O que será exibido no console, quando o método exibePilha for executado, tendo P1, P2 e P3 sido passadas como parâmetros?

a) 10 15 25 28
b) 10 25 30 40
c) 15 10 28 25
d) 20 35 34 40
e) 40 34 30 60

79. (FGV/TCE SP/2023) Marcos é um estudante de programação de computadores e encontrou o algoritmo apresentado a seguir em seus estudos.

```
algoritmo Marcos
var
i, a: inteiro
v: vetor [1..5] de inteiro
início
a <- 0

para i de 1 até 5 faça:
  leia(v[i]);
fim para

para i de 1 até 5 faça:
   v[i] <- v[i] + v[6-i]
   a <- a + v[i]
fim para

escreva(a)
fim algoritmo
```

Considerando o vetor com índice inicial 1 e final 5, e utilizando os valores {2, 1, 2, 1, 2}, Marcos obterá a impressão do valor:

a) 3
b) 8
c) 9
d) 16
e) 19

80. (SUGEP UFRPE/UFRPE/2022) A estrutura de dados "vetor" (array) é um arranjo unidimensional que pode acomodar múltiplos dados. Sobre essas estruturas de dados, assinale a alternativa incorreta.

a) Os dados de um vetor são mapeados numa área contígua da memória.
b) Os dados de um vetor são do mesmo tipo.
c) Cada um dos dados de um vetor pode ser acessado informando-se o identificador do vetor e o inteiro que indica a ordem do dado na sequência.
d) Os dados de um vetor são armazenados na memória ordenadamente, em modo crescente.
e) Pode-se atribuir um dado a um elemento de qualquer posição do vetor, independentemente do que foi atribuído aos demais elementos.

GABARITO

GABARITO

1. Letra C
2. Correto
3. Errado
4. Errado
5. Letra B
6. Letra A
7. Letra B
8. Letra A
9. Correto
10. Letra D
11. Letra A
12. Correto
13. Letra D
14. Letra B
15. Letra A
16. Letra A
17. Letra D
18. Letra E
19. Letra C
20. Letra
21. Letra A
22. Errado
23. Correto
24. Errado
25. Correto
26. Errado
27. Correto
28. Errado
29. Errado
30. Errado
31. Errado
32. Letra D
33. Letra B
34. Letra B
35. Letra C
36. Letra D
37. Letra D
38. Correto
39. Correto
40. Letra A
41. Letra B
42. Letra E
43. Letra D
44. Letra E
45. Errado
46. Letra D
47. Letra B
48. Letra D
49. Correto
50. Errado
51. Letra D
52. Letra C
53. Errado
54. Letra E
55. Letra C
56. Letra B
57. Letra D
58. Letra E
59. Letra E
60. Letra C
61. Correto
62. Errado
63. Errado
64. Errado
65. Letra A
66. Errado
67. Letra C
68. Letra B
69. Letra D
70. Letra B
71. Letra B
72. Letra A
73. Letra C
74. Errado
75. Correto
76. Letra
77. Letra B
78. Letra A
79. Letra E
80. Letra D

PROGRAMAÇÃO ASSÍNCRONA

Pode ser que a comunicação instantânea não esteja disponível em determinado momento, ou que seja inaplicável para certas aplicações. Nesses casos, usamos um paradigma de programação chamado de programação assíncrona. Com os subterfúgios da comunicação assíncrona, podemos criar aplicações com tarefas de longa duração sem ocorrência de problemas.

O seu funcionamento, na teoria, é simples: ao invés de realizarmos requisições e só prosseguirmos na aplicação quando tivermos a resposta, as requisições são lançadas e a aplicação segue independente da resposta do destino, aguardando-as e realizando as tarefas conexas no momento em que a resposta for feita. Isso garante maior responsividade, desempenho e eficiência dos recursos.

O uso mais proeminente no seu dia a dia é em download de arquivos. Sem a programação assíncrona, o aplicativo teria que aguardar a conclusão do download antes de permitir que o usuário interaja ou inicie outro download. Com a programação assíncrona, o usuário pode iniciar vários downloads simultaneamente, e o programa continua executando outras tarefas enquanto os downloads ocorrem em segundo plano. Assim que um download for concluído, uma notificação, ou *callback,* é acionado para informar ao usuário que o arquivo está pronto. Dessa forma, temos um funcionamento mais fluido para os usuários.

ATENÇÃO

PARA O TRECHO ABAIXO, É IDEAL QUE VOCÊ TENHA CONHECIMENTOS BÁSICOS EM PROGRAMAÇÃO, JÁ QUE EXPLORAREMOS A APLICAÇÃO DA PROGRAMAÇÃO ASSÍNCRONA EM CÓDIGOS.

javascript

```javascript
// Exemplo de download de arquivos
function downloadFile(file) {
    console.log(`Iniciando o download do arquivo: ${file}`);

    // Simulando um download demorado
    for (let i = 0; i < 3e9; i++) {
        // Operação demorada
    }

    console.log(`Download do arquivo ${file} concluído.`);
}

// Chamada de download síncrono
console.log("Início do programa");
downloadFile("documento.pdf");
```

Acima, temos um exemplo de aplicação síncrona. O *loop for* simula um download demorado e, somente após a sua conclusão, teremos a implementação do resultado - o *log* que indica que o download foi concluído. Antes disso, nenhuma outra ação é executada pela aplicação. Vamos ver como ficaria essa mesma interação, mas de forma assíncrona.

```javascript
// Aplicação Assíncrona - Exemplo de download de arquivos
function downloadFileAsync(file, callback) {
    console.log(`Iniciando o download do arquivo: ${file}`);

    // Simulando um download assíncrono com setTimeout
    setTimeout(() => {
        console.log(`Download do arquivo ${file} concluído.`);
        callback(); // Chamando a função de retorno após o download
    }, 3000); // Simulando um download de 3 segundos
}

// Chamada de download assíncrono
console.log("Início do programa");
downloadFileAsync("documento.pdf", () => {
    console.log("Programa concluído");
});
```

Agora, na versão assíncrona, a função *downloadFileAsync* simula um download assíncrono usando *setTimeout*. O programa não espera bloqueado durante o download - em vez disso, fornece uma função de retorno de chamada que será chamada quando o download estiver concluído. Isso permite que o programa continue executando outras tarefas enquanto aguarda a conclusão do download.

É possível, também, o uso de ***promises***. As *promises* são funções de programação assíncrona, introduzidas para lidar de forma mais eficiente e legível com os problemas assíncronos. Seu objetivo é evitar a excessividade de funções de *callback*, fornecendo um modelo mais limpo e concisos para lidar com as operações assíncronas.

Uma *promise* pode estar em um de três estados:

- Pending (Pendente): O estado inicial, quando a Promise está em execução ou aguardando a conclusão de uma operação assíncrona.
- Fulfilled (Realizada): A operação assíncrona foi concluída com sucesso, e o resultado está disponível.
- Rejected (Rejeitada): A operação assíncrona falhou, e um motivo (uma razão para a falha) está disponível.

Podemos implementar as *promises* de uma forma muito simples em javascript: com as funções ASYNC/AWAIT. Veja como implementaríamos a mesma função *downloadFileAsync* com o uso desse par de operadores, de forma muita mais simples e visual:

javascript

```javascript
// Exemplo de download de arquivos usando async/await
function downloadFileAsync(file) {
    return new Promise((resolve) => {
        console.log(`Iniciando o download do arquivo: ${file}`);

        // Simulando um download assíncrono com setTimeout
        setTimeout(() => {
            console.log(`Download do arquivo ${file} concluído.`);
            resolve(); // Resolvendo a Promise após o download
        }, 3000); // Simulando um download de 3 segundos
    });
}

// Função assíncrona que utiliza async/await
async function main() {
    console.log("Início do programa");

    // Utilizando await para esperar o download antes de continuar
    await downloadFileAsync("documento.pdf");

    console.log("Programa concluído");
}

// Chamada da função principal assíncrona
```

No exemplo acima, a função *downloadFileAsync* retorna uma *promise* que é resolvida após o download assíncrono. A função principal *main* é marcada como assíncrona com *async* e utiliza

await para aguardar a conclusão do download antes de prosseguir para a próxima etapa do programa. Isso torna o código mais legível e semelhante à execução síncrona, facilitando o entendimento do fluxo de controle assíncrono.

(AOCP/IF MA/2023) No JavaScript, a execução de código assíncrono pode ser gerenciada de várias maneiras. A respeito de como lidar com a execução assíncrona de código JavaScript, assinale a alternativa INCORRETA.

a) Utilizar callbacks, em que uma função é passada como argumento para outra função e é chamada quando a execução assíncrona é concluída.
b) Utilizar Promises, que representam o resultado de uma operação assíncrona que pode ser concluída no futuro.
c) Utilizar async/await, que permite escrever código assíncrono de maneira síncrona, facilitando a legibilidade do código.
d) Utilizar generators, que permitem a criação de funções cuja execução pode ser suspensa e retomada posteriormente.
e) Utilizar o operador "sincrono" para forçar a execução síncrona do código assíncrono.

Comentários:

Essa é uma questão que envolve mais JavaScript que a programação assíncrona, mas trouxe para te mostrar a quantidade de diferentes abordagens que podemos ter para tratar a programação assíncrona. De todas as afirmativas apontadas, a única que não encontra respaldo - e portanto, está incorreta - é a letra E, já que não existe operador síncrono. (Gabarito: Letra E)

(FCC/SEFAZ SP/2009) Para uma Web Service síncrona, quem chamou a função

a) não precisa esperar o retorno, tal qual para uma Web Service assíncrona, porém, para a forma síncrona pode manter mais uma linha de execução no código e para a forma assíncrona não pode.
b) deve esperar o retorno para prosseguir e, para uma Web Service assíncrona, não precisa esperar o retorno, podendo manter mais uma linha de execução no código.
c) deve esperar o retorno para prosseguir e, para uma Web Service assíncrona, não precisa esperar o retorno, não podendo manter mais uma linha de execução no código.
d) não precisa esperar o retorno, podendo manter mais uma linha de execução no código e, para uma Web Service assíncrona, deve esperar o retorno para prosseguir.
e) não precisa esperar o retorno, não podendo manter mais uma linha de execução no código e, para uma Web Service assíncrona, deve esperar o retorno para prosseguir.

Comentários:

Como lhes trouxe no começo da aula, aplicações síncronas só podem prosseguir a execução após a resposta à solicitação, ao passo que aplicações assíncronas podem realizar inúmeras solicitações e prosseguir na execução antes mesmo de receberem respostas.

Nesse sentido, correto o apontamento feito pela letra B. (Gabarito: Letra B)

(CEBRASPE/TCE-PA/2016) Julgue o item subsequente, relativos a SOA, web services e servidor web.

A comunicação assíncrona pode ser implementada com o objetivo de proporcionar baixo acoplamento em SOA.

Comentários:

SOA, ou arquitetura orientada a serviços, é um paradigma de arquitetura de softwares que organiza as aplicações como conjuntos de serviços interconectados - similar aos microsserviços, mas na SOA não há independência entre eles. Nesse contexto, é correto afirmar que as soluções assíncronas fornecem baixo acoplamento, pois não haverá uma relação de dependência entre o pedido e a resposta para que o programa continue sendo executado. (Gabarito: Correto)

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.