` FTL066 – Programação em Tempo Real
Quinta Lista de Exercícios

1) Ada termina cada construtor com **end** <nome do construtor>; C não usa um marcador final. Quais são as vantagens e desvantagens dos projetos destas linguagens?

Vantagem de que os blocos de código passam a ser mais visíveis e isso possibilita uma melhor visualização e diminui a possibilidade de erros, o que pode acabar reduzindo a possibilidade de vulnerabilidades. Entretanto, isso acaba deixando o código mais volumoso.

2) Java e C são *case-sensitive*; Ada não é. Quais são os argumentos a favor e contra do *case sensitive*?

Argumentos a favor do case sensitive:

1. Distinção entre identificadores: O case sensitive permite que os programadores diferenciem identificadores com base em letras maiúsculas e minúsculas. Isso permite a criação de identificadores mais descritivos e semânticos, melhorando a legibilidade e compreensão do código.
2. Consistência: A sensibilidade a maiúsculas e minúsculas permite que os programadores tenham consistência na nomenclatura de identificadores e na chamada de funções e métodos. Isso facilita a leitura e a manutenção do código, pois o programador pode identificar rapidamente se um identificador está sendo usado corretamente.
3. Maior expressividade: A sensibilidade a maiúsculas e minúsculas permite que os programadores usem variações de letras maiúsculas e minúsculas para comunicar diferentes conceitos ou significados. Por exemplo, é comum usar letras maiúsculas para indicar constantes ou nomes de classes em Java, enquanto letras minúsculas são usadas para variáveis.

Argumentos contra o case sensitive:

1. Dificuldade de digitação: A sensibilidade a maiúsculas e minúsculas pode aumentar a dificuldade de digitar código, especialmente para programadores que não estão acostumados a esse estilo. Erros de digitação devido ao uso incorreto de maiúsculas e minúsculas podem levar a bugs difíceis de detectar.
2. Inconsistências e erros sutis: A sensibilidade a maiúsculas e minúsculas pode introduzir inconsistências e erros sutis no código se não for seguida de forma consistente. Por exemplo, se um identificador for digitado incorretamente com uma letra maiúscula em vez de minúscula, pode levar a problemas de compilação ou comportamento inesperado.
3. Portabilidade e interoperabilidade: Em alguns casos, a sensibilidade a maiúsculas e

minúsculas pode criar problemas de portabilidade e interoperabilidade entre diferentes sistemas e ambientes. Por exemplo, ao trabalhar em projetos que envolvem diferentes sistemas operacionais ou ambientes de desenvolvimento, pode ser necessário lidar com diferenças na forma como os sistemas tratam a sensibilidade a maiúsculas e minúsculas.

É importante lembrar que a escolha de usar ou não sensibilidade a maiúsculas e minúsculas em uma linguagem de programação é uma decisão de design feita pelos criadores da linguagem, levando em consideração diferentes fatores, como legibilidade, consistência e praticidade. Cada abordagem tem suas vantagens e desvantagens, e a escolha depende das necessidades e preferências da linguagem e da comunidade de programadores.

3) Uma linguagem deve sempre solicitar que sejam dados valores iniciais para as variáveis? Cite pelo menos um problema relacionado a não inicialização de variáveis usadas no programa.

1. Comportamento indefinido: Se uma variável não for inicializada e for usada antes de ser atribuído um valor, seu conteúdo será indefinido. Isso pode levar a resultados inesperados, erros ou comportamento inconsistente no programa.
2. Bugs difíceis de detectar: Se uma variável não for inicializada e for usada em cálculos ou em condições de controle de fluxo, o programa pode produzir resultados incorretos ou falhar. Identificar a causa desses bugs pode ser difícil, pois a falta de inicialização pode resultar em comportamento imprevisível.
3. Vazamento de informações sensíveis: Em alguns casos, uma variável pode conter dados sensíveis, como senhas, informações pessoais ou informações confidenciais. Se uma variável não for inicializada e ainda assim for usada, pode haver um vazamento acidental desses dados sensíveis para outras partes do programa ou até mesmo para usuários não autorizados.
4. Uso incorreto de memória: Se uma variável não for inicializada e for usada em operações que dependem de seu valor, como alocação de memória, cópia de dados ou operações aritméticas, pode ocorrer um comportamento inesperado ou mesmo erros de violação de acesso à memória.

4) O uso do comando **exit** em Ada leva a programas legíveis e confiáveis?

Sim, o comando exit é sempre usando no fim de um bloco de programação, como condições ifs, loops e ela pode ajudar a saber quando o fim de um bloco acontece, ajudando na legibilidade do código o que possivelmente ajudaria ao programador a evitar erros.

No entanto, confiabilidade não é determinada apenas pelo uso do exit. A estruturação adequada do código, o uso correto de tipos e a utilização de exceções para tratar erros são também primordiais.

5) Escreva programas para a solução dos seguintes problemas usando a linguagem Ada95:

A solução se encontra no github:
https://github.com/victor-souza1997/quinta-lista-ada/tree/main/quinta

a) Lido um string, escreva-o na ordem inversa (palíndromo).

b) Lidos dois vetores A, de tamanho m e B, de tamanho n, calcule o vetor C, soma de A e B.

c) Lido um string, conte quantas vogais existem.

d) Lido um string, escrever quantas e quais são as vogais.

e) O produto interno entre dois vetores A e B é o escalar obtido por $\Sigma\, a_i x b_i$, onde n é o tamanho dos dois vetores.

f) Lidos dois strings A e B, verifique se o string B está contido em A.

g) Lido um array numérico de n elementos, ache o maior.

h) Lido um conjunto de n elementos, ordená-lo crescentemente.

i) Leia uma matriz A e calcule a média aritmética de seus elementos.

j) Lidas duas matrizes A, de dimensões mxn e B, de dimensões pxq, calcule a matriz C, soma de A e B.

k) Lida uma matriz, gere sua matriz transposta. **Dica:** Matriz transposta, em matemática, é o resultado da troca de linhas por colunas em uma determinada matriz. A matriz transposta de uma matriz qualquer M é representada por Mt.

l) Lidas duas matrizes A, de dimensões mxn e B, de dimensões pxq, calcule a matriz C, produto de A e B.

m) Lida uma matriz quadrada, calcule a somatória dos elementos da diagonal principal.

n) Lidas duas matrizes de tamanho 3x3 verifique se uma é inversa da outra.

o) Lida uma matriz, calcule a porcentagem de elementos nulos desta matriz.

6) Implemente um procedimento que irá imprimir a área de qualquer objeto geométrico de um tipo derivado a partir do tipo ***Object***.

```
package Object is

 type Object is tagged
 record

 X_Coord: Float;

 Y_Coord: Float;

 end record;
```

```
 function Distance(O: Object) return Float;
 function Area(O: Object) return Float;
end Objects;
```

7) Em um mundo tradicional mulheres não tem barbas e homens não amamentam os filhos. Porém, todas as pessoas têm uma data de nascimento. Declare um tipo ***Person*** com o componente comum ***Birth*** do tipo ***Date*** (conforme mostrado abaixo) e então derive os tipos ***Man*** e ***Woman*** que tenham componentes adicionais indicando se eles possuem barba ou não e quantos filhos eles amamentam respectivamente.

```
type Month_Name is (Jan, Feb, Mar, Apr, May, Jun,
                    Jul, Aug, Sep, Oct, Nov, Dec);
type Date is
 record
 Day: Integer range 1..31;
 Month: Month_Name;
 Year: Integer;
 end record;
```

8) Declare procedimentos ***Print_Details*** para ***Person***, ***Man*** e ***Woman*** que fornecem informações a respeito dos valores atuais dos componentes deles. Então declare um procedimento ***Analyze_Person*** que recebe um parâmetro do tipo de uma classe ampla ***Person'Class*** e chama o procedimento apropriado ***Print_Details***.

9) Escreva a especificação e o corpo do pacote ***Queues*** usando uma implementação de lista encadeada.

10) Escreva um pacote ***generic*** para ***Queues*** tal que filas de qualquer tipo possam ser declaradas.

11) Escreva a especificação e o corpo do pacote ***Stacks*** usando uma implementação de lista encadeada.

12) Escreva um pacote ***generic*** para ***Stacks*** tal que pilhas de qualquer tipo possam ser declaradas.

13) Crie um tipo de dado abstrato (TDA) chamado Retângulo. O TDA tem atributos

comprimento e largura cada um com valor default igual a 1. Ele tem funções “membro” que calculam o *comprimento*, *largura*, *perímetro* e *área* do retângulo. Forneça as funções *set* e *get*, tanto para o comprimento como para a largura. A função *set* deve verificar se o comprimento e a largura são números de ponto flutuante maiores que 0.0 e menores que 20.0. Além disso, a função *set* especifica se as coordenadas fornecidas de fato especificam um retângulo. Lembre que o comprimento é a maior das duas dimensões. Inclua uma função quadrado, que determina se um retângulo é um quadrado.

14) Crie um TDA Racional para fazer aritmética com frações. Escreva um programa para testar seu TDA. Use variáveis inteiras para representar os dados ***private*** do TDA – o numerador e o denominador. Forneça uma “função” construtor que permita que um objeto deste TDA seja inicializado quando é declarado. O construtor deve conter valores *default* no caso de nenhum inicializador ser fornecido e deve armazenar a fração em formato reduzido. Forneça funções membro para cada um dos seguintes itens:

a) Adição, subtração, multiplicação e divisão de dois números do tipo Racional. b) Imprimir números do tipo Racional no formato a / b alinhado a esquerda. c) Imprimir números do tipo Racional em formato de ponto flutuante com 3 dígitos de precisão.

21/10/2021