Lista de exercícios 2 - Estrutura de linguagens

1. V/F
a) Uma linguagem interpretada não pode ser baseada em pilha.
Falsa - Não teria razão para uma linguagem interpretada não usar pilha. Temos a jdm baseadas em pilhas. É necessário que uma linguagem interpretada tenha pilhas. Numa linguagem totalmente interpetada não podemos prever quanto de memória será utilizada, por isso precisamos de pilha.

b) Uma linguagem estática permite chamadas recursivas de função.
Falsa - Não permite. Sem pilha, não podemos fazer recursividade.

c) O uso da pilha faz com que os programas consumam mais memória que os programas equivalentes em linguagens estáticas (quando for possível ter um equivalente em linguagem estática).
Falsa - O uso da pilha usa a mesma memória, mas usamos um endereçamento relativo a base. O consumo de memória não tende a aumentar, o que muda é a forma de endereçamento.

d) E possível manter uma lista encadeada na pilha.
Falsa - Poderia ter, mas teriamos muitos problemas que inviabilizariam seu uso.
Problemas: A lista morreria com o fim do escopo. O tamanho da pilha que temos ele deve ser calculado estaticamente. Uma lista encadeada pode crescer infinitamente. O problema é que o tamanho não pode ser calculado a priori, por isso a alocação dinâmica para cada nó.

e) Uma linguagem totalmente dinâmica não usa a pilha
Falsa - Usamos memória heap, que não influi em nada na pilha. Em C e Java podemos usar alocações dinâmicas, mas para variáveis locais nós continuamos a ter o uso de pilhas.

2.

```
int n = 0;
void FuncInt (k) {
        n = n + 1;
        k = k + 7;
        printf("n: %d k: %d", n, k);
        k++;
}

void main() {
        func(n);
        printf("n: %d", n);
}
```

a) Chamada por valor.
n: 1 k: 7
n: 1

b) Chamada por valor-resultado - Durante a função, k e n não tem relação.
n: 1 k: 7
n: 8 (o valor de k é copiado para n ao sair da função.)

c) Chamada por referência - n e k apontam para o mesmo lugar.

n: 8 k: 8
n: 9

3.

a) - chamada por valor (copiar da variavel)
1) a
2) a

b) - chamada por nome (passo a expressão v[i])
1) a
2) 0

4. Descreva a função do loader quando se deseja carregar uma biblioteca compartilhada que foi compilada com as diretivas de Código Independente de Posição (acrônimo em inglês PIC - Position Independent Code).

Função do loader quando se deseja carregar uma biblioteca compartilhada.

Codigo independente da posição - os códigos não podem utilizar endereços absolutos.

No carregamento da bibliotea o loader deve corrigir todos os endereços de variáveis.

Já que o código sem reposição usa-se o endereçamento relativo, em que existe uma tabela contendo os endereços das variáveis globais.

O que varia é a posição de cada tabela, com a biblioteca sendo carregada uma única vez(não confirmado).

5.

a) qtd npares
```
npares :: (Integral a) => [a] -> a
npares [] = 0
npares(x:xs)
        |x `mod` 2 == 0 = 1 + (npares xs)
        |otherwise = npares xs
```

b) qtd vogais
```
nvogal (Integral i) => String -> i
nvogal [] = 0
nvogal (x:xs)
        |elem x "aeiouAEIOU" = 1+(nvogal xs)
        |otherwise = nvogal xs
```

c) qtd vogais com list compression
```
nvogal' xs = length([k | k <- xs, k `elem` "aeiouAEIOU"])
```

d) split que divide uma lista ao meio
```
split xs = (tira xs, drop , xs)
        where n = (length xs) `div` 2
```

```
tira n (x:xs)
        |n < 1 = []
        |otherwise = x : tira (n-1) xs
                    [x] ++ Tira (n-1) x
```

6. bubble sort

```
bubble [] = []
bubble xs = menor : bubble ys
    where(menor, ys) = removemenor xs
removemenor ys = (menor, [x | x <- xs, x > menor])
```