IMD0029 - Estrutura de Dados Básicas 1 – 2018.2 – Prova 02
Prof. Eiji Adachi M. Barbosa

Nome:_____

Matrícula: _____

ANTES DE COMEÇAR A PROVA, leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 5,0 pontos na Unidade I e o valor de cada questão é informado no seu enunciado.
- Preze por respostas legíveis, bem organizadas e simples.
- As respostas devem ser fornecidas preferencialmente em caneta. Respostas fornecidas a lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- **Desvios éticos ou de honestidade levarão a nota igual a zero na Unidade 2.**

Questão 1: (1,0 ponto) Considere uma lista simplesmente encadeada com ponteiro para o primeiro nó da lista implementada conforme a estrutura abaixo:

```
List{
   Node* first;
}
```

```
Node {
   Node* next;
   int value;
}
```

Para esta estrutura, implemente um método recursivo que conte quantos nós com valor par a lista encadeada possui. Seu método deverá seguir a assinatura abaixo:

```
int List::CountEven();
```

Obs.1: *Se necessário, crie métodos auxiliares para implementar sua solução.*
Obs.2: *Pode assumir que existem métodos get's e set's para os atributos das classes List e Node.*

Questão 2: (1,5 ponto) Na notação usual de expressões aritméticas, os operadores são escritos entre os operandos; por isso, a notação é chamada infixa. Na notação pós-fixa, ou notação polonesa inversa, os operadores são escritos depois dos operandos. Eis alguns exemplos:

Pós-Fixa	Infixa
AB+	(A+B)
AB+C*	((A+B)*C)
ABC+*	(A*(B+C))
ABC+*D/E-	(((A*(B+C))/D)-E)
AB*CD*+EF*GH*-/	(((A*B)+(C*D))/((E*F)-(G*H)))

Nesta questão, faça uma função que recebe como entrada uma string representando uma expressão aritmética em notação pós-fixa e retorna uma outra string representando esta mesma expressão aritmética em notação infixa. Sua função deverá obrigatoriamente usar uma pilha, ter complexidade assintótica **$\Theta(n)$** – 'n' é o tamanho da string de entrada – e seguir a assinatura:

```
string Convert (string expression)
```

Obs.1: Considere que a expressão de entrada contém apenas letras representando os operandos da expressão e símbolos representando as quatro operações aritméticas básicas ('+', '-', '', '/'). Ou seja, cada caractere da string de entrada ou é um operando, representado por uma única letra, ou é um operador.*
Obs.2: Considere que a expressão de entrada sempre será bem formada. Ou seja, sua solução não precisa checar se ela está bem formada na notação pós-fixa.

Questão 3: (1,5 ponto) Muitas aplicações acessam um mesmo dado em curtos períodos de tempo. Para este tipo de aplicação, uma estratégia para otimizar o acesso aos dados comumente implementada em listas encadeadas é a estratégia da contagem de acessos. Nesta estratégia, mantém-se um atributo extra em cada nó para contar o número de vezes que aquele nó foi acessado. Desta forma, a cada vez que um nó é buscado, o seu atributo que conta o número de acessos é

incrementado. Além disso, após cada busca realizada com sucesso na lista, os nós devem ser rearranjados de modo que fiquem ordenados em ordem decrescente em relação ao número de acesso dos nós. Considere uma lista duplamente encadeada com sentinelas cabeça (*head*) e cauda (*tail*) implementada conforme a estrutura abaixo:

```
List{
   Node* head;
   Node* tail;
}
```

```
Node {
   Node* next;
   Node* previous;
   int value;
   int count;
}
```

Dada esta estrutura de lista duplamente encadeada, implemente o método de busca que segue a estratégia da contagem de acessos descrita acima. Sua implementação deve seguir a assinatura:

`Node List::Search(int K) – Retorna o nó cujo conteúdo tem valor igual a K.`

Questão 4: (1,0 ponto) Para cada uma das afirmações a seguir, marque V (verdadeiro) ou F (falso), **justificando sucintamente** sua resposta. Marcações de V ou F **sem justificativas não serão aceitas.**

1 – ( ) Usando um array, é possível implementarmos uma Pilha com operações de inserir e remover (*push* e *pop*) com complexidade assintótica O(1), mas não é possível implementarmos uma Fila com operações de inserir e remover (*queue* e *dequeue*) com complexidade assintótica O(1).

2 – ( ) Considere duas listas simplesmente encadeadas: a Lista I possui apenas um ponteiro para o primeiro nó da lista, e a Lista II possui um ponteiro para o primeiro nó da lista e um outro ponteiro para o último nó da lista. Em relação à complexidade assintótica da operação de "Inserir no Fim", a Lista II é mais eficiente do que a Lista I.

3 – ( ) Considere duas listas simplesmente encadeadas: a Lista I possui apenas um ponteiro para o primeiro nó da lista, e a Lista II possui um ponteiro para o primeiro nó da lista e um outro ponteiro para o último nó da lista. Em relação à complexidade assintótica da operação de "Remover no Fim", a Lista II é mais eficiente do que a Lista I.

4 – ( ) Usando uma lista simplesmente encadeada contendo apenas um ponteiro para o primeiro nó da lista, é possível construir uma estrutura que segue a estratégia First-In First-Out (FIFO) e cujas operações de inserção e remoção de elementos têm complexidade assintótica constante.

5 – ( ) Usando uma lista simplesmente encadeada contendo apenas um ponteiro para o primeiro nó da lista, é possível construir uma estrutura que segue a estratégia First-In Last-Out (FILO) e cujas operações de inserção e remoção de elementos têm complexidade assintótica constante.