IMD0029 - Estrutura de Dados Básicas 1 – 2018.1 – 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 II 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 à anulação da prova do candidato (nota igual a zero).

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 a lista encadeada possui. Seu método deverá seguir a assinatura abaixo:

```
int List::count();
```

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) 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;
   int value;
}
```

Implemente um método que remove os elementos duplicados de uma lista e retorna quantos elementos duplicados foram removidos. Este método deve funcionar corretamente mesmo para listas não-ordenadas. Seu método deverá seguir a assinatura abaixo:

```
int List::removeDuplicates();
```

Obs.1: *Não é permitido usar outras estruturas auxiliares, como listas ou arrays; só é permitido operar sobre a lista de entrada. Para tanto, é permitido a criação de ponteiros auxiliares. As operações básicas sobre a lista não podem ser reusadas; faça sua solução apenas manipulando os nós da lista. Se necessário, você pode definir funções auxiliares para melhor organizar o código da sua solução.*
Obs.2: *Remover duplicadas quer dizer que ao menos um dos elementos duplicados permanece na lista.*
Obs.3: *Pode assumir que existem métodos get's e set's para os atributos das classes List e Node.*

Questão 3: (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 de expressões infixas e correspondentes expressões pós-fixas:

Infixa	**Pós-fixa**
(A+(B*C))	ABC*+
(((A*(B+C))/D)-E)	ABC+*D/E-

Nesta questão, implemente uma função que recebe como entrada uma string representando uma expressão em notação pós-fixa e retorna uma string com a mesma expressão representada em notação infixa. Sua função deverá ser implementada usando obrigatoriamente uma estrutura **PILHA** e seguir a assinatura abaixo:

```
string postfixToInfix( string exp );
```

Obs.1: *Assuma que a estrutura pilha já está implementada e você pode reusar as suas operações básicas: push, pop, top, empty.*
Obs.2: *Assuma que a string de entrada só possui operandos e os operadores + - / *.*

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 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.

2 – ( ) 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.

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 "Inserir no Fim", a Lista II é mais eficiente do que a Lista I.

4 – ( ) 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.

5 – ( ) 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).