IMD0029 - Estrutura de Dados Básicas 1 – Turma 03 – 2016.1 – 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 10,0 pontos 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).
- Algumas questões possuem observações. Antes de tirar dúvidas, leia atentamente às observações feitas em cada questão.

Questão 1: (2,0 pts) Assuma que você possui à sua disposição uma Lista já implementada corretamente e que disponibiliza as seguintes funções:

```
Lista Criar();
Bool Inserir( Lista, Elemento, i );
Bool Remover( Lista, i );
Elemento Buscar( Lista, i );
```

No caso da função `Inserir`, o parâmetro `i` é um inteiro que significa que, depois de inserido, o `Elemento` estará na i-ésima posição da `Lista`. Já no caso das funções `Remover` e `Buscar`, o parâmetro `i` é um inteiro que significa que se deseja remover ou buscar o elemento que está na i-ésima posição da lista. Considere que a primeira posição da lista é igual a 1.

Com base na estrutura Lista já implementada, criamos a seguinte estrutura `Foo`:

```
Estrutura Foo
{
     int qtdItens;
     Lista itens;
}
```

A estrutura `Foo` deverá prover as seguintes funções:

-
```
/* Insere um Elemento na estrutura Foo */
Bool Inserir( Foo, Elemento );
```

-
```
/* Remove um Elemento da estrutura Foo, retornando o elemento removido */
Elemento Remover( Foo );
```

Neste contexto, reuse a implementação da Lista para implementar as funções `Inserir` e `Remover` da estrutura `Foo` seguindo a estratégia FIFO (*First-in, First-Out*, isto é, primeiro a entrar, primeiro a sair). Além disso, a estrutura `Foo` não deverá permitir a inserção de elementos repetidos.

Questão 2: (2,0 pts) Uma *Lista Auto-Ajustável* é uma especialização da Lista Encadeada que reorganiza seus elementos com o objetivo de melhorar o tempo médio da função `Buscar`. A ideia geral das Listas Auto-Ajustáveis é mover os nós que são mais buscados para mais próximo do seu início. Existem diversas estratégias para reorganizar uma Lista Auto-Ajustável. Na estratégia *Transposição de Nós*, quando a função `Buscar` encontra com sucesso um determinado nó, este nó é trocado de posição com o seu predecessor. Desta forma, o nó que foi buscado com sucesso move-se uma posição em direção ao início da lista. Neste contexto, implemente a função `Buscar` seguindo a estratégia *Transposição de Nós* e conforme a assinatura:

```
/* Busca um nó cujo atributo id é igual ao parâmetro Id */
Nó Buscar( Lista, Id );
```

Para esta questão, considere que a função `Buscar` irá operar sobre as seguintes estruturas que definem uma *Lista Duplamente Encadeada com Sentinelas*:

```
Estrutura Lista{
      qtdNós;
      Nó cabeça;
      Nó cauda;
}
```

```
Estrutura Nó{
      id;
      conteúdo;
      Nó próximo;
      Nó anterior;
}
```

Obs.1: Os nós devem ser efetivamente trocados de posição. Não é válido trocar apenas o conteúdo dos nós.

Questão 3: Implemente as funções abaixo para uma Tabela de Dispersão que trata colisões pelo método do *Endereçamento Aberto com Sondagem Quadrática*.

a) (2,0 pts) `Bool Remover( Tabela, Chave )`

b) (2,0 pts) `Item Buscar( Tabela, Chave )`

Para esta questão, considere que a Tabela de Dispersão a ser manipulada é definida conforme as seguintes estruturas:

```
Estrutura Tabela
{
      qtdItens;
      tamanho;
      Chave chaves[tamanho];
      Item itens[tamanho];
}
```

```
Estrutura Chave
{
      Id;
}
```

```
Estrutura Item
{
      Conteudo;
}
```

Obs.1: Assuma que o tipo `Chave` não é um tipo numérico e que as seguintes funções já estão implementadas e podem ser usadas: `int PreHash( Chave )` e `int Hash( preHash, tamanho )`.
Obs.2: Para esta questão, não é necessário se preocupar com redimensionamento dinâmico.

Questão 4: (2,0 pts) Quando uma Tabela de Dispersão começa a ficar com o seu fator de carga elevado, a eficiência de suas operações começa a decair devido a uma maior probabilidade de colisões. Nestes casos, vimos em sala de aula que uma estratégia para contornar este problema é implementar Tabelas de Dispersão com *Redimensionamento Dinâmico*. Considerando as mesmas estruturas da questão anterior, implemente a seguinte função que poderá ser usada para redimensionar uma Tabela de Dispersão:

```
Tabela Redimensionar( Tabela T, int N );
```

A função Redimensionar deverá receber como parâmetros de entrada uma Tabela `T` e um inteiro `N` e retornar uma nova tabela de dispersão de tamanho N contendo todos os itens e chaves da tabela de dispersão passada como parâmetro de entrada.

Obs.1: Para esta questão, a função `bool Inserir( Tabela, Chave, Item )` não está implementada. Ou seja, você <u>não pode</u> usar a função `Inserir` dentro da função redimensionar.
Obs.2: Assuma que o tipo `Chave` não é um tipo numérico e que as funções `int PreHash( Chave )` e `int Hash( preHash, tamanho )` já estão implementadas e podem ser usadas.