IMD0029 - Estrutura de Dados Básicas 1 – Turma 03 – 2016.1 – Prova Reposição
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) Implemente uma função que recebe como parâmetros de entrada: (i) um vetor de inteiros ordenados, com elementos repetidos e (ii) um valor inteiro X. Esta função deve retornar um par `(a,b)`, em que `a` é o índice da primeira ocorrência de X no vetor e `b` o índice da última ocorrência de X no vetor. Sua função deverá ter obrigatoriamente complexidade O(lg(n)) e deverá seguir a seguinte assinatura:

```
Par BuscaIntervalor(int v[MAX_TAM], int X);
```

Abaixo alguns exemplos do comportamento esperado desta função:

```
BuscaIntervalo( [ 1, 2, 3, 3, 4, 4, 4, 5, 6, 7], 2)  == (1,1)
BuscaIntervalo( [ 1, 2, 3, 3, 4, 4, 4, 5, 6, 7], 3)  == (2,3)
BuscaIntervalo( [ 1, 2, 3, 3, 4, 4, 4, 5, 6, 7], 4)  == (4,6)
BuscaIntervalo( [ 1, 2, 3, 3, 4, 4, 4, 5, 6, 7], 10) == (-1,-1)
```

Perceba que quando o valor X não é encontrado no vetor, é retornado um par igual a (-1,-1).

Obs.1: Considere que o tamanho do vetor de entrada é igual a `MAX_TAM` e que seus índices começam em 0.
Obs.2: Para retornar um par use a seguinte sintaxe: `retorne (a,b)` (ou `return (a,b)`).

Questão 2: (2,0 pts) Implemente uma função de Partição que opere sobre uma lista duplamente encadeada com sentinelas cabeça e cauda. Considere que a lista de entrada não contém elementos repetidos. Sua função deverá seguir a seguinte assinatura:

```
void Particionar( Lista lista );
```

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

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

```
Estrutura Nó{
    int valor;
    Nó próximo;
    Nó anterior;
}
```

Obs.1: É proibido modificar as estruturas do enunciado. Também é proibido criar uma outra lista ou um vetor auxiliar na solução desta questão. Ou seja, é obrigatório operar apenas sobre a lista passada como parâmetro. Questões que desobedeçam tais instruções serão desconsideradas.

Questão 3: (2,0 pts) Implemente uma função que recebe um vetor de listas e faz a intercalação de todas elas, retornando uma nova lista contendo todos os elementos destas listas intercalados. Considere que as listas contidas no vetor de entrada estão ordenadas em ordem crescente. Sua função deverá seguir a seguinte assinatura:

```
Lista Intercalar( Lista v[MAX_TAM] );
```

Para esta questão, considere que a função `Intercalar` irá operar sobre estruturas Listas iguais àquela definida na questão anterior.

Obs.1: Considere que o tamanho do vetor de entrada é igual a `MAX_TAM`.
Obs.2: É proibido modificar as estruturas do enunciado. É obrigatório realizar a intercalação operando sobre as listas encadeadas. Questões que desobedeçam tais instruções serão desconsideradas.

Questão 4: (2,0 pts) Você foi designado a implementar uma tabela de dispersão da seguinte forma. Em caso de colisões, a sua implementação deverá dar prioridade aos pares (chave, item) mais novos. Isto é, se durante a inserção de um novo par (chave, item) ocorrer uma colisão num determinado índice X, o novo par ocupará a posição X e o par que estava anteriormente na posição X deverá ser deslocado para uma nova posição. Esta nova posição deverá ser buscada usando a estratégia de sondagem quadrática. Nesta questão, é necessário implementar apenas a função de inserir seguindo a seguinte assinatura:

```
bool Inserir( Tabela, Chave, Item);
```

Para esta questão, considere que a função `Inserir` irá operar sobre a seguintes estrutura que define uma *Tabela de Dispersão*:

```
Estrutura Tabela{
     int tamanho;
     int qtdElementos;
     Item itens[MAX_TAM];
     Chave chaves[MAX_TAM];

}
```

Obs.1: Considere que o tamanho dos vetores `itens` e `chaves` é igual a `MAX_TAM`.
Obs.2: Considere que instâncias das estruturas `Item` e `Chave` podem ser comparadas usando os operadores de comparação padrão (`==`, `!=`, `>`, `<`, etc).
Obs.3: Assuma que as seguintes funções já estão implementadas e podem ser usadas: `int PreHash( Chave )` e `int Hash( preHash, tamanho )`.
Obs.4: Para esta questão, não é necessário se preocupar com redimensionamento dinâmico.

Questão 5: (2,0 pts) Implemente uma função que recebe como entrada um arquivo de texto contendo apenas palavras e que registra em quais linhas cada palavra ocorre e que conta quantas vezes cada palavra ocorre no texto todo. Nesta questão, o arquivo de entrada só pode ser percorrido uma vez, sendo proibido percorrê-lo mais de uma vez. Após processar o arquivo de entrada, sua função deverá imprimir cada palavra encontrada no arquivo, em quais linhas ela ocorreu e quantas vezes aquela palavra ocorreu no texto todo, como mostrado no exemplo a seguir:

```
palavra – [1, 6, 10, 23, 38] - 10
arquivo – [1, 3] - 2
texto - [4] - 1
```

Implemente sua função seguindo a seguinte assinatura:

```
void ProcessarArquivo( Arquivo arquivo );
```

Para esta questão, é obrigatório usar a estrutura Tabela de Dispersão definida na questão anterior. As definições das estruturas `Chave` e `Item` fazem parte da sua solução. Portanto, deixe bem claro em sua resposta como você as definiu.

Obs.1: Considere que o tamanho dos vetores `itens` e `chaves` é igual a `MAX_TAM` e que este tamanho é suficiente para guardar todas as palavras contidas no arquivo de entrada.
Obs.2: Considere que palavras e instâncias das estruturas `Item` e `Chave` podem ser comparadas usando os operadores de comparação padrão (`==`, `!=`, `>`, `<`, etc).
Obs.3: Assuma que as seguintes funções já estão implementadas e podem ser usadas: `int PreHash( Chave )` e `int Hash( preHash, tamanho )`.
Obs.4: Para esta questão, não é necessário seguir a sintaxe de C/C++ para manipular o arquivo. Pode usar versões simplificadas das funções de manipulação de arquivos como “AbrirArquivo”, “LerLinha”, etc, desde que fique claro o que tal função faz. Também não é necessário se preocupar se o arquivo existe ou não. Pode assumir que o arquivo recebido como entrada sempre existirá.