INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: BUSCA BINÁRIA E HASHTABLE

Aluno (a):			
Matrícula:		**Data:**	

Prática 06

Parte 0: Preparação

Passo 1: Crie um novo projeto chamado **Pratica6**.

Parte 1: Implementando Lista Ordenada e Busca Binária

Passo 1: Utilize o arquivo **ordenada.cpp** que acompanha a prática 6 ao projeto.

> A função `main()` nesse arquivo realiza uma série inserções na lista, valida e exibe o seu conteúdo. Em seguida faz uma série de buscas tanto de forma sequencial como binária, exibindo o índice do elemento (caso tenha sido encontrado).

Passo 2: Implemente os métodos que faltam na classe `ListaOrdenada`, seguindo as orientações abaixo:

> Método `insere()`: esse método deve inserir o elemento na lista de forma a mantê-la ordenada. Dica de implementação: varra a lista de trás para frete, movendo para a direita os elementos maiores que o elemento a ser inserido. Quando achar um item da lista menor ou igual ao elemento, parar e inserir na posição seguinte.
>
> Método `buscaSequencial()`: esse método busca o elemento na lista do começo até o final, retornando o índice quando é encontrado ou -1 caso contrário. Para a busca ser mais otimizada, podemos tirar vantagem da ordenação: no momento que for encontrando um elemento maior que a chave que buscamos, podemos encerrar a busca sem sucesso (-1).
>
> Método `buscaBinaria()` [privado]: esse método realiza a busca do elemento na lista empregando a técnica de busca binária descrita no material de aula.

Passo 3: Rode e teste a aplicação.

> Verifique se a lista está válida e se o resultado das buscas está correto. Pode haver divergência no resultado das buscas quando há repetição de valores na lista.

Passo 4: Adicione e implemente um método para remover um item da lista.

> Modifique a função `main()` de forma a testar a remoção: valide e exiba a lista depois da remoção; busque os elementos removidos para verificar se foram de fato.

Passo 5: (Desafio) Faça testes comparando o tempo de busca usando a busca sequencial e a busca binária.

> Inicialize a lista com milhares de elementos e faça uma série de buscas de cada tipo (também na ordem de milhares) para ver o tempo que cada um toma.

Parte 2: Implementando Tabela de Espalhamento (Hashtable)

Passo 1: Adicione o arquivo **hashtable.cpp** que acompanha a prática 6 ao projeto.

Nesse arquivo está implementada uma tabela de espalhamento que trabalha com entradas na forma (chave, valor) ou (key, value). No construtor dessa classe é indicado o tamanho da tabela e um valor default que é retornado quando a busca por uma chave falha.

Do ponto de vista de implementação, a tabela usa uma estrutura ligada (isto é, com ponteiros) interna para armazenar os pares (chave, valor). Por exemplo, ao inserir na tabela, busca-se primeiro a posição correta na tabela (através da função de hash), a qual aponta para um lista encadeada contendo os elementos.

A função `main()` instancia uma tabela de espalhamento associando nomes a valores reais (notas) e realiza uma série inserções e buscas nessa tabela.

Passo 2: Implemente os métodos que faltam na classe `Hashtable`, seguindo as orientações abaixo:

Método `insert(key, value)`: esse método deve usar a função `hash()` privada para gerar um número inteiro a partir da chave (`key`) e, com esse número, gerar um índice na tabela (considerando a capacidade). Uma vez com esse índice, deve ser chamado o método privado `insert(node, key, value)` passando o nó da tabela, a chave e o valor. Esse método realiza uma inserção na lista encadeada daquela entrada na tabela.

Método `remove(key)`: semelhante ao `insert(key, value)`, primeiro é gerado um índice para a tabela, que é usado para obter o nó que será repassado junto com a chave para o método privado `remove(node, key)`. Esse último método retorna o valor que havia associado a essa chave, caso exista, ou o valor default de erro, caso contrário.

Método `search(key)`: como nos outros, primeiro obtêm-se o hash/índice da chave, o qual indica o nó que é repassado para `search(node, key)`. Esse último método retorna o valor associado a essa chave, caso exista, ou o valor default de erro, caso contrário.

Passo 3: Rode e teste a aplicação.

Verifique se a tabela é válida e se o resultado das buscas está correto.

Passo 4: Melhore a função de espalhamento de forma a diminuir a ocorrência de colisões.

Por exemplo, experimente substituir a operação de multiplicação por uma soma. Faça testes para ver se os elementos são mais bem distribuídos na tabela.