Estruturas de Dados em C++

Ano/Semestre: 2019/2

Prof. Márcio Miguel Gomes - marciomg@unisinos.br

Trabalho do Grau B

Apresentação: Os trabalhos serão apresentados pelos grupos diretamente ao professor na aula do dia **26/11/2019**. Não haverá prorrogação de data. O tempo máximo para apresentação por grupo será de 20 minutos. A ordem de apresentação será definida no dia da apresentação.

Instruções para envio do trabalho: Enviar o projeto com todos os arquivos de código-fonte do programa para a atividade aberta no **Moodle até às 19h30min do dia 26/11/2019**. Envios após esse horário terão desconto de 1,5 pontos a cada dia de atraso. Apenas um integrante do grupo deve enviar os arquivos informando o nome de todos os componentes do grupo.

Grupos: no máximo 3 integrantes.

Contextualização

O arquivo "toy_sale.csv" possui o registro de venda de réplicas de veículos. É um arquivo texto com colunas separadas por ponto-vírgula.

Classes

Registro: classe que representa um objeto equivalente a uma linha do arquivo de dados. Deve receber no seu construtor uma linha do arquivo, realizar a *desserialização* dos dados e alimentar os atributos do objeto.

HashTable: classe capaz de armazenar e manipular objetos *vector* STL usando chaves genéricas.

Sistema: classe que gerencia a criação e liberação de todos os objetos, fluxos de dados e interações com o usuário.

Programa

O programa deve ser desenvolvido em uma classe chamada **Sistema**. O objeto **Sistema** deve receber em seu construtor o nome do arquivo a ser processado. Em seguida, deve abrir o arquivo, percorrê-lo do início ao fim e instanciar um objeto **Registro** para cada linha do arquivo (cada linha representa um item vendido). Os objetos **Registro** devem ser armazenados em um **vector** da STL chamado "**dados**". O destrutor da classe "**Sistema**" deve liberar todos recursos alocados dinamicamente.

Após carregar todos os registros no vector de dados, deve ser criada uma **HashTable** para o campo **PRODUCTLINE** contendo exatamente um índice para cada categoria e outra **HashTable** para o campo **CITY** contendo entre 7 e 15 índices. As **chaves** das *hashtables* devem ser **genéricas**, e os elementos devem ser um ponteiro para o objeto **Registro** associado à chave. As funções de *hash* e a quantidade de índices de cada *hashtable* devem garantir uma distribuição o mais uniforme possível para os registros. Os registros de cada índice das *hashtables* devem ser ordenados de forma crescente pelo valor das chaves.

Estruturas de Dados em C++

Ano/Semestre: 2019/2

Prof. Márcio Miguel Gomes - marciomg@unisinos.br

Menu

Crie um menu para interação com o usuário que implemente as seguintes funcionalidades:

1. **Localizar por "ORDERNUMBER"**: Mostrar na tela todos os dados de um ORDERNUMBER fornecido pelo usuário. Os dados ORDERNUMBER, ORDERDATE, STATUS, CUSTOMERNAME, CITY, COUNTRY, CONTACTLASTNAME, CONTACTFIRSTNAME e DEALSIZE devem ser exibidos uma única vez, uma informação por linha. Já os dados QUANTITYORDERED, PRICEEACH, PRODUCTLINE e PRODUCTCODE devem aparecer logo abaixo em formato de tabela. Utilizar localização por pesquisa binária sobre o *vector* de objetos Registro.
2. **Localizar por "PRODUCTLINE"** e **"COUNTRY"**: Mostrar na tela os campos ORDERNUMBER, ORDERDATE, STATUS, CITY e CUSTOMERNAME a partir de um PRODUCTLINE e COUNTRY informados pelo usuário. Para isso, deve ser utilizada a *hashtable* de PRODUCTLINE para acessar os produtos, e em seguida, fazer uma procura sequencial por COUNTRY. Não deve mostrar dados duplicados.
3. **Exportar por "CITY"**: Salvar em um arquivo todos os dados dos registros que possuam o campo CITY igual ao valor informado pelo usuário. Acessar os dados através da *hashtable* CITY.
4. **Histogramas**: Gerar na tela um histograma (contagem agrupada) listando os índices de cada *hashtable* e quantos registros estão associados a cada índice. Serão 2 histogramas, um para PRODUCTLINE e outro para CITY.

Critérios de avaliação:

- Implementação de **todas** as funcionalidades solicitadas;
- O código-fonte entregue deve ser **compilável** e **executável**;
- O programa deve ser todo **orientado a objetos**;
- O código-fonte deve estar **corretamente identado** e **comentado**;
- Os **nomes** dos atributos, métodos, classes, parâmetros e variáveis utilizados devem ser **autoexplicativos**, utilizando a notação "**camel case**";
- Todos os componentes do grupo devem explicar uma parte do trabalho durante a apresentação;