Estruturas de Dados em C++

Ano/Semestre: 2019/2

Prof. Márcio Miguel Gomes - marciomg@unisinos.br

Trabalho do Grau A

Apresentação: Os trabalhos serão apresentados pelos grupos diretamente ao professor na aula do dia **24/09/2019**. Não haverá prorrogação da data de apresentação. 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 para a atividade aberta no **Moodle até às 19h30min do dia 24/09/2019**. Envios após esse horário terão desconto progressivos de 15% sobre a nota a cada 24h 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:

Um servidor ftp registra as requisições feitas pelas aplicações clientes em um arquivo de log contendo os dados: *timestamp, ip, port, command, mime_type, file_size, reply_code e reply_msg*. O arquivo está no formato TSV (*tab separated values*). A coluna *timestamp* está no formato *POSIX time*. Dados vazios são representados por um hífen.

Classes:

Implementar as classes descritas abaixo.

```
class Registro
{
private:
   DataHora dataHora;
   string origemIP;
   int origemPorta;
   string comando;
   string mimeType;
   int fileSize;
   int replyCode;
   string replyMsg;
public:
   Registro(string linha);
   // métodos adicionais
}
```

```
class DataHora
{
private:
   time_t timestamp;
public:
   DataHora(string timestamp);
   string getDataHora();
   // sobrecarga de operadores
   // métodos adicionais
}
```

```
class Sistema
{
private:
   vector<Registro*> logs;
public:
   Sistema(string arqLog);
   // métodos adicionais
}
```

A classe *Registro* deve receber no seu construtor uma linha do arquivo de log, realizar a interpretação dos dados e alimentar os atributos do objeto. O atributo *dataHora* deve ser um objeto da classe *DataHora*, que recebe no seu construtor um *timestamp* e o armazena em um atributo privado. A classe *DataHora* deve realizar operações de comparação entre duas datas usando os operadores <, <=, >, >= e == (ou seja, utilizando sobrecarga de operadores). Os demais atributos devem respeitar os tipos de dados definidos nas classes.

O programa deve ser desenvolvido em uma classe chamada *Sistema*, ou seja, o *main* instancia um objeto da classe *Sistema*, chama a sua execução e o finaliza ao encerrar o programa. O objeto *Sistema* deve receber em seu construtor o nome do arquivo de log a ser processado. Em seguida, deve abrir o arquivo de log, percorrê-lo do início ao fim e instanciar um objeto *Registro* para cada linha do arquivo (cada linha representa um registro do log). Os objetos *Registro* devem ser armazenados em um *vector* chamado *logs* (o vector deve, obrigatoriamente, ser de ponteiros para objetos *Registro*). Todo o trabalho de filtragem, visualização e exportação de dados deve ser feito sobre os objetos do *vector logs*. Adicione novos atributos e métodos nas classes *Registro*, *Sistema* e *DataHora* se necessário.

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. **Adicionar filtro**. Mostre um sub-menu listando os atributos que aceitam filtragem. O usuário deve selecionar uma opção e em seguida informar o critério de seleção.
   - Data e Hora: informar um intervalo de tempo, ou seja, valor inicial e final no formato ISO 8601
   - IP: informar o endereço IP do solicitante
   - Porta: informar um intervalo de portas, ou seja, valor inicial e final
   - Comando: informar o comando ftp, aceitando apenas valores válidos
   - Mime Type: informar um texto e filtrar por substring case insensitive
   - File Size: informar um intervalo de tamanhos, ou seja, valor inicial e final
   - Reply Code: informar um único código e filtrar por valor exato
   - Reply Message: informar um texto e filtrar por substring case insensitive
   - Deve ser possível adicionar mais de um filtro para a exibição dos registros de logs
2. **Limpar filtros**. Limpa a lista de filtros previamente adicionados
3. **Visualizar filtros**. Mostra na tela todos os filtros previamente adicionados
4. **Visualizar dados**. Mostra na tela os registros que satisfaçam todos os filtros definidos. O campo *timestamp* deve ser apresentado no formato ISO 8601.
5. **Exportar dados**. Salva em arquivo os registros que satisfaçam todos os filtros definidos. O programa deve solicitar ao usuário o nome do arquivo que conterá os registros. O campo *timestamp* deve ser apresentado no formato ISO 8601.

Formatos:

- Os dados devem ser apresentados com uma linha de cabeçalho (identificando o nome do atributo) e um rodapé (mostrando os filtros utilizados e a quantidade de registros selecionados)
- O campo "Data Hora" deve obedecer o padrão ISO 8601 (YYYY-MM-DDThh:mm:ss). Fonte: https://www.w3.org/TR/NOTE-datetime
- A visualização em tela deve utilizar colunas de largura fixa
- A exportação deve gerar um arquivo TSV (texto tabulado) em que cada atributo é separado por TAB.

Observações:

- Todas as opções do menu devem ser implementadas. A não-implementação de alguma opção acarretará em um desconto na nota final do grupo.
- Os **nomes** de classes, atributos e métodos especificados acima **devem ser mantidos** na implementação do código (ou seja, não os renomeie). Novos métodos e atributos devem ser nomeados de acordo com a sua respectiva função.

Critérios de avaliação:

- 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**";
- Devem ser utilizados, no mínimo, os seguintes comandos e conceitos: orientação a objetos, vectors, iteradores, manipulação de arquivos, sobrecarga de operadores e alocação dinâmica;
- Todas as funcionalidades do programa contidas nesta definição devem ser implementadas;
- Todos os componentes do grupo devem explicar uma parte do trabalho durante a apresentação;