UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I | 2017.1

Projeto de Programação III
Loja de Conveniência *v2.0*

Objetivo

O objetivo deste trabalho é introduzir melhorias ao programa de cadastro e venda de produtos para a loja de conveniência *QLeveTudo*, implementado no Projeto de Programação II. As melhorias a serem introduzidas dizem respeito à aplicação de conceitos STL, bibliotecas e tratamento de exceções. Este documento apresenta as novas melhorias a serem introduzidas na versão 2.0.

Melhorias na versão 2.0

Você deverá implementar as seguintes alterações no programa de cadastro e venda de produtos para a loja de conveniência *QLeveTudo*:

1) Substitua a TAD Lista, implementada por você, por um *container* do tipo `map` da STL. Realize as correções/adaptações necessárias para tirar o máximo proveito do uso do `map`.

2) A fim de permitir a reutilização de código em projetos futuros, organize o seu modelo de classes em uma biblioteca dinâmica de nome `qlevetudo.so` (Linux) ou `qlevetudo.dll` (Windows). Esta biblioteca deverá ser utilizada na construção dos programas que irão compor o sistema da *QLeveTudo*.

3) Utilizando a biblioteca criada no item anterior, implemente um programa auxiliar para permitir exportar **apenas dados comuns a todos os produtos** e que satisfaçam a um determinado conjunto de critérios.

```
./exportar -t <tipo> -f <fornecedor> --full <arquivo_saida>
```

   Na sintaxe acima, o arquivo executável `exportar` é invocado passando quatro possíveis argumentos via linha de comando:

   - `-t <tipo>`: este argumento **opcional** indica o tipo de produto a ser exportado (bebida, salgado, DVD, etc.). Por razões de simplicidade, deverá ser indicada apenas um tipo;
   - `-f <fornecedor>`: este argumento **opcional** indica que apenas produtos fornecidos pelo fornecedor informado devem ser exportados. Por razões de simplicidade, deverá ser indicado apenas um fornecedor.
   - `--full`: este argumento **opcional** indica que, para cada produto a ser exportado, deverá ser também exportado os dados que definem o tipo de produto selecionado. Caso não seja informado um tipo, o uso deste argumento deverá gerar um erro.

- `<arquivo_saida>`: este argumento **obrigatório** indica o nome do arquivo de saída, ou seja, do arquivo onde serão guardados os dados a serem exportados. Este arquivo deve ser gerado em formato CSV.

4) Realize o devido tratamento de exceções para as operações de manipulação de arquivos e para a entrada (leitura) de dados por meio da criação das classes de exceção necessárias e lançamento dos respectivos objetos quando for o caso.

5) Uma vez realizadas as tarefas de implementação, você deverá elaborar um relatório detalhando as modificações implementadas em sua versão 2.0. **A apresentação do relatório é obrigatória e a sua ausência implicará em nota zero nesta tarefa**.

Dica: Para o processamento de parâmetros em programas escritos em C/C++, recomenda-se o uso da função `getopt`. Para maiores informações sobre o uso dessa função, consulte:

- Documentação oficial: https://www.gnu.org/software/libc/manual/html_node/Getopt.html
- Boa descrição (em Português), ainda que voltada para o uso com a linguagem C: https://daemoniolabs.wordpress.com/2011/10/07/usando-com-as-funcoes-getopt-e-getopt_long-em-c/
- Um bom exemplo do uso na linguagem C++: http://code.runnable.com/UqvNM9NFP0gtAAAM/command-line-arguments-with-options-in-c++

Orientações gerais

Você deverá atentar para as seguintes observações gerais no desenvolvimento deste trabalho:

1) Apesar da completa compatibilidade entre as linguagens de programação C e C++, seu código fonte **não** deverá conter recursos da linguagem C nem ser resultante de mescla entre as duas linguagens, o que é uma má prática de programação. Dessa forma, deverão ser utilizados **estritamente** recursos da linguagem C++.

2) Durante a compilação do seu código fonte, você deverá habilitar a exibição de mensagens de aviso (*warnings*), pois elas podem dar indícios de que o programa potencialmente possui problemas em sua implementação que podem se manifestar durante a sua execução.

3) Aplique boas práticas de programação. Codifique o programa de maneira legível (com indentação de código fonte, nomes consistentes, etc.) e documente-o adequadamente na forma de comentários. Como anteriormente instruído, o código fonte deverá ser anotado para dar suporte à geração automática de documentação utilizando a ferramenta Doxygen (http://www.doxygen.org/).

4) Busque desenvolver o seu programa com qualidade, garantindo que ele funcione de forma correta e eficiente. Pense também nas possíveis entradas que poderão ser utilizadas para testar apropriadamente o seu programa e trate adequadamente possíveis entradas consideradas inválidas.

5) Lembre-se de aplicar boas práticas de modularização, em termos da implementação de diferentes funções/métodos e separação entre arquivos cabeçalho (`.h`) e corpo (`.cpp`).
6) A fim de auxiliar a compilação de seu projeto, você deverá **obrigatoriamente** construir um `Makefile` que faça uso da estrutura de diretórios apresentada anteriormente em aula.
7) Garanta o uso consistente de alocação dinâmica de memória. Para auxilia-lo nesta tarefa, você pode utilizar o Valgrind (http://valgrind.org/) como ferramenta de apoio para verificar se existem problemas de gerenciamento de memória.
8) A sua abstração sobre o contexto do trabalho faz parte da avaliação e será refletida no modelo de classes a ser implementado. Fique livre para adicionar funcionalidades e características que julgue necessárias ou interessantes.

Autoria e política de colaboração

Este trabalho deverá ser realizado individualmente ou em equipe de **no máximo** dois integrantes. O trabalho em cooperação entre estudantes da turma é estimulado, sendo admissível a discussão de ideias e estratégias. Contudo, tal interação não deve ser entendida como permissão para utilização de (parte de) código fonte de colegas, o que pode caracterizar situação de plágio. Trabalhos copiados em todo ou em parte de outros colegas ou da Internet serão sumariamente rejeitados e receberão nota zero.

Independentemente de o trabalho ser realizado individualmente ou em equipe, você deverá utilizar o sistema de controle de versões Git no desenvolvimento. Ao final, todos os arquivos de código fonte do repositório Git local deverão estar unificados em um repositório remoto Git hospedado em algum serviço da Internet, a exemplo do GitHub, Bitbucket, Gitlab ou outro de sua preferência. A fim de garantir a boa manutenção de seu repositório, configure corretamente o arquivo `.gitignore` em seu repositório Git. A atividade de cada aluno deverá ser registrada através de *commits* sobre o repositório Git, que será examinado pelos professores durante a avaliação do trabalho.

Entrega

Você deverá submeter um único arquivo compactado no formato **`.zip`** contendo todos os códigos fonte resultantes da implementação deste exercício, sem erros de compilação e devidamente testados e documentados, **até as 23h59 do dia 29 de junho de 2017** através da opção *Tarefas* na Turma Virtual do SIGAA. Juntamente com os códigos fonte, o arquivo compactado deverá também conter o relatório escrito, preferencialmente em formato PDF. Por fim, você deverá ainda informar, no campo *Comentários* do formulário de submissão da tarefa, o endereço do repositório Git utilizado.

Avaliação

A avaliação deste trabalho será feita principalmente sobre os seguintes critérios: (1) utilização correta dos conteúdos vistos anteriormente e nas aulas presenciais da disciplina; (2) a corretude da execução do programa implementado, que deve apresentar saída em conformidade com a especificação e as entradas de dados fornecidas; (3) a aplicação correta de boas práticas de programação, incluindo

legibilidade, organização e documentação de código fonte; (4) qualidade do relatório técnico produzido, e; (5) a utilização correta do repositório Git, no qual deverá estar registrado **todo** o histórico da implementação por meio de *commits*. O trabalho possuirá nota máxima de 4,00 (quatro) pontos para a 3ª unidade da disciplina, distribuídos de acordo com a seguinte composição:

Item avaliado	**Nota máxima**
Modularização adequada	0,50
Melhorias da versão 2.0	
- Uso do *container* `map` da STL	0,50
- Criação e utilização de biblioteca	0,50
- Programa para exportação de dados	1,00
- Tratamento de exceções	0,50
Qualidade do relatório escrito	1,00
Total	**4,00**

Por sua vez, o não cumprimento de algum dos critérios anteriormente especificados poderá resultar nos seguintes decréscimos, calculados sobre a nota total obtida até então:

Falta	**Decréscimo**
Programa apresenta erros de compilação, não executa ou apresenta saída incorreta	–70%
Falta de comentários no código fonte e/ou de documentação gerada com Doxygen	–10%
Implementação na linguagem C ou resultante de mistura entre as linguagens C e C++	–30%
Programa compila com mensagens de aviso (*warnings*)	–50%
Plagiarismo	–100%
Uso inadequado do `Makefile` e/ou erros na verificação de memória	-30%
`Makefile` que apaga arquivos ao usar o alvo `clean`	**-100**%
Uso incorreto de controle de versão com Git	-30%