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

Trabalho 1
Classes, Objetos, Métodos Construtores, Método Destrutor e Sobrecarga de Operadores

Objetivo

O objetivo deste exercício é colocar em prática conceitos iniciais do paradigma de Programação Orientada a Objetos (POO) na linguagem de programação C++, em particular a implementação de classes, objetos, métodos construtores, método destrutor e sobrecarga de operadores.

Orientações gerais

Você deverá observar as seguintes observações gerais na implementação deste exercício:

1) 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.
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 e separação entre arquivos cabeçalho (`.h`) e corpo (`.cpp`).
6) A fim de auxiliar a compilação do seu projeto, construa **obrigatoriamente** um `Makefile` que faça uso da estrutura de diretórios já discutida em aula.

Programa 1

Implemente um programa em C++ que atenda aos seguintes critérios:

a) Contenha uma classe que representa um funcionário, registrando seu nome (pesquise o uso da classe *string*[1]), salário e data de admissão;
b) Contenha também uma classe que representa uma empresa, registrando seu nome, CNPJ e lista de funcionários;
c) O programa deverá permitir criar uma empresa;
d) O programa deverá permitir que se adicione funcionários (um por vez) a uma empresa e não deve permitir adicionar um funcionário que já tenha sido anteriormente adicionado, sendo neste caso exibida uma mensagem de erro. Os funcionários de uma empresa poderão ser mantidos em um *array* (você já conhece), *vector*[2] *ou list*[3] (pesquisar sobre a STL do C++), ou ainda alguma estrutura de dados que você já implementou em EDB1 e que considere conveniente.
   Dica: utilize a sobrecarga dos operadores de igualdade (==) e de extração de *stream* (>>).
e) O programa deverá permitir listar os dados de todos os funcionários de uma empresa, sobrecarregando-se o operador de inserção em *stream* (<<);
f) O programa deverá permitir que seja dado um aumento de *X***%** a todos os funcionários de uma determinada empresa;
g) O programa deverá permitir listar os dados de todos os funcionários de uma empresa em período de experiência, ou seja, contratados há menos de 90 dias considerando a data corrente;
h) O programa deverá permitir listar a média de funcionários por empresa. O cálculo deve ser realizado a partir de atributos estáticos das próprias classes (Empresa e Funcionário).

Dica 1: Observe que as funcionalidades exigidas nesta questão podem (**e devem**) ser resolvidas com a alteração das classes (por exemplo, criando novos métodos) e não manipulando de forma mirabolante os objetos no programa principal. Não complique o que é simples!

Dica 2: A modelagem das classes é livre, ou seja, inclua os atributos que você considerar importantes para resolver a questão. Mas não se limite, pois quanto mais elaborado for o seu modelo de classes, melhor será a avaliação de seu trabalho.

[1] http://www.cplusplus.com/reference/string/
http://www.cplusplus.com/reference/string/string/
[2] http://www.cplusplus.com/reference/vector/vector/
[3] http://www.cplusplus.com/reference/list/list/

Programa 2

Utilizando classes, implemente em C++ um jogo de dados hipotético, no qual cada jogador (de vários) deve lançar dois dados em sua vez. A soma dos valores dos dados deve ser acumulada para cada jogador. O objetivo é ficar o mais próximo e abaixo do valor *N* a ser estabelecido como configuração geral do jogo e visível a todos os jogadores. Ao obter um valor agregado superior a *N*, o jogador é considerado excluído da rodada. A cada vez de jogar, o jogador pode optar por jogar os dados ou parar (e não jogar mais na rodada). Uma rodada é finalizada quando: (1) resta apenas um jogador, uma vez que os outros foram excluídos; (2) quando não há mais jogadores a jogar, ou seja, todos os jogadores "ativos" decidiram parar, ou; (3) quando um dos jogadores atinge exatamente o valor **N**. Vence a rodada o jogador que permanecer na rodada (ou seja, não for excluído) e obtiver o número de pontos agregados mais próximo de *N*. Ao final de cada rodada, o programa deverá listar os pontos agregados obtidos por cada jogar e declarar o campeão da rodada.

Autoria e política de colaboração

Este trabalho deverá ser realizado **individualmente** ou **em dupla**. 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.

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, através da opção *Tarefas* na Turma Virtual do SIGAA, de acordo como prazo da tarefa cadastrada na turma.

Avaliação

O trabalho será avaliado sob os seguintes critérios: (i) utilização correta dos conteúdos vistos anteriormente e nas aulas presenciais da disciplina; (ii) a corretude da execução dos programas implementados, que devem apresentar saída em conformidade com a especificação e as entradas de dados fornecidas, e; (iii) a aplicação correta de boas práticas de programação, incluindo legibilidade, organização e documentação de código fonte. A presença de mensagens de aviso (*warnings*) ou de erros de compilação e/ou de execução, a modularização inapropriada e a ausência de documentação são faltas que serão penalizadas.