Data: 03/10/2017 – 2º Semestre de 2017.
Prof. Dr. Jean M. Simão - DAINF/UTFPR Curitiba-Central.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ - Campus de Curitiba Central – **Dep. Acadêmico de Informática** (DAINF). Disciplina: **Técnicas de Programação – CSE20**.
Prova sobre linguagem C++ / Orientação a Objetos – 1ª Parcial

Nome do Aluno: _____ **Turma**: S71
Curso: _____ Horário de Começo: _____ **Horário de Fim**: _____

Leia toda a prova, pois os enunciados estão completados uns nos outros.
Utilize os bons princípios de projeto e programação orientada a objetos.

(Questão - 1) (1.1) Em um programa C++ (para *console*), crie uma classe *Personagem* abstrata com um método virtual puro chamado *mover*. Entretanto, esta classe pode ter outros métodos. Ainda, esta classe terá um atributo privado *string* chamado *nome.* **(1.2)** Essa classe também permitirá três classes derivadas dela, chamadas de *Jogador*, *Protetor* e *Inimigo*, sendo que estas duas também são abstratas via redefinição (em cada qual) do método *mover* como virtual puro. Ainda, essas três classes terão um atributo protegido, respectivamente *vidas* (inteiro), *energia* (real) e *ativo* (boleano). **(1.3)** Por fim, em cada classe criar construtoras, destrutoras e gets/sets apropriados (este basta um exemplo...), bem como usar corretamente elementos constantes (*const*) nos seus constituintes.

(Questão - 2) (2.1) Crie uma classe *Curupira* (com inteiro chamado *idade*) e uma classe *Fada* (com um atributo string chamado *posto*) derivadas de *Protetor* e uma classe *BoiTata* derivada de *Inimigo* (com um atributo real chamado *poder_fogo*). **(2.2)** As classes *Curupira*, *Fada* e *BoiTata* devem redefinir o método virtual pertinente (virtual puro de classe base) que informará em tela o *nome* e a frase "movendo um 'passo' ", tal qual terão ao menos um método a mais, respectivamente void *proteger_todos*, *proteger_jogador* e *enviar_fogo*" cujas implementações terão um comentário dizendo "a fazer". **(2.3)** Outrossim, o operador de incremente ++ será sobrecarregado para a classe *BoiTata* para permitir incrementar *poder_fogo*. **(2.4)** Ainda, a classe *Curupira* permitirá apontamentos para qualquer instância de derivados de *Personagem* por meio de um *vector* STL apropriado para tal. Por fim, tanto a classe *BoiTata quanto Fada* permitirá apontamento para a instância (objeto) da classe *Jogador*.

(Questão - 3) (3.1) Crie uma classe de pilha para ponteiros de objetos de *Personagem*. Esta classe *Pilha_Personagens* deve ser programada com qualquer técnica que o desenvolvedor julgar pertinente (e.g., vetor normal, componente STL ou encadeamento). Na prática, a classe *Pilha_Personagens* servirá para ponteiros de objetos de subclasses de *Personagem* (apontados cada qual como *Personagem* bem entendido). Ainda, a classe *Pilha_Personagens* deverá ter um método *void empilhar (Personagem* ps) que permitirá até 10 elementos empilhados*, bem como outro método *void desempilhar()*. **(3.2)** Por fim, em cada eventual objeto de *Pilha_Personagens*, o desempilhamento chamará poliformicamente o método *mover* do objeto cujo apontamento esteja registrado nesta pilha em questão.

(Questão - 4) (4.1) Crie uma classe *Jogo* sendo que a instância ou o objeto dela se chamará *Brasileirinho*. No *Brasileirinho* poderá se armazenar ponteiros de objetos das classes *Jogador*, *Curupira, Fada* e *BoiTata* por meio de um método *void incluir_personagem (Personagem* pe)*. Assim, visando polimorfismo, o objeto *Brasileirinho* armazenará os apontamentos de objetos dessas classes como apontamentos de objetos da classe *Personagem*. Isto se dará por meio de um objeto agregado do tipo *Pilha_Personagens*. **(4.2)** Por fim, a classe *Jogo* terá um método *void listar()* que chamará método *desempilhar* apropriado deste objeto agregado.

(Questão - 5) (5.1) Primeiramente, elabore um diagrama de classes em UML das classes solicitadas nas questões precedentes, pelo menos com seus nomes e seus relacionamentos. **(5.2)** Depois de elaborado o diagrama e escrito o código, faça um crítica apontando ao menos um ponto que poderia ser melhorado do ponto de vista dos princípio da coesão e desacoplamento que rege o correto encapsulamento e modularidade.