Lista 5 (EXERCÍCIOS)

1) Considere o seguinte código:

```
#include <iostream>

using namespace std;

class Ferramenta
{
	// Preencher
};

// Implemente classe Tesoura
// Implemente a classe Papel
// Implemente a classe Pedra

int main()
{
	// Exemplo de main
	// Adicione seu próprio código para teste e altere o código se
necessário
	Tesoura s1;
	Papel p1;
	Pedra r1(2);
	cout << s1.lutar(p1) << p1.lutar(s1) << endl;
	cout << p1.lutar(r1) << r1.lutar(p1) << endl;
	cout << r1.lutar(s1) << s1.lutar(r1) << endl;
	return 0;
}
```

Implemente uma classe Ferramenta. Ela deve ter um atributo "força" e um atributo "tipo". A escolha do tipo desses atributos (int, char, enum, etc.) e seu modificador de acesso (private, protected, public) fica a seu critério.

A classe ferramenta deverá definir seus atributos em seu construtor, de modo que nunca exista um objeto Ferramenta com atributos indefinidos.

Crie 3 classes chamadas Pedra, Papel e Tesoura, que herdem de Ferramenta. Cada uma dessas classes deverá atribuir apropriadamente o tipo da Ferramenta.

Implemente um construtor para essas classes de tal maneira que a força da ferramenta pode ser especificada e, caso não seja, tenha o valor padrão 1.

Essas classes devem ter um método `bool lutar(const Ferramenta &)`, que compara as forças da seguinte maneira:

A força da Pedra é dobrada (temporariamente) ao lutar contra Tesoura e reduzida pela metade (temporariamente) ao lutar contra papel.

Similarmente, Papel tem vantagem contra Pedra e Tesoura contra Papel.

O atributo "força" não deve ser alterado. A função apenas retorna verdadeiro se o objeto vencer em força e falso caso contrário.

2) Escreva um programa que defina uma classe Poligono com um construtor que recebe valor para Comprimento e Altura. Defina duas subclasses Triangulo e Retangulo, que calculam a área por uma função `double area()`. No main, defina duas variáveis Triangulo e Retangulo e então chame a função area() desses objetos.

3) Implemente uma classe Animal, com um atributo "força" e um atributo "vida", o construtor de Animal deverá setar estes atributos de maneira que nunca exista um animal com atributos indefinidos. Implemente na classe Animal um método `void lutar(Animal &)`, que calcula e aplica dois valores de dano de maneira aleatória, um para o animal atacante e outro para o animal atacado, usando as fórmulas: dano = valorAleatorio(0,força); e vida = vida - dano. Quando dano for igual a força, o ataque é considerado crítico e tem um modificador de 1.5x. Quando dano for zero o ataque é considerado uma falha crítica e o animal causa 1 ponto de dano a si mesmo. Implemente também a classe Leão com força/vida 50/100 e a classe Poodle com força/vida 0.5/10. Utilizando as classes acima crie um programa que responda a charada milenar: **500 poodles matam um leão?** Em 20 execuções do seu programa, quantas vezes o leão morre e quantas vezes ele ganha?

4) Considere a código:

```
class Um : public Dois {
      public:
      Um()  {
            Tres tres;
            cout << "alguma coisa" << endl;
      }
      ~Um() {
            cout << "outra alguma coisa" << endl;
      }
      private:
      Quatro _quatro;
};
```

Produza a saída “um dois tres quatro cinco seis sete”. Não é permitido alterar o código da Classe Um com exceção de suas saídas (os dois cout). É permitido criar outras classes. Dica: não é necessário criar mais do que três outras classes.

5) Como são resolvidos os conflitos de nomes de funções entre superclasses e subclasses?

6) Qual a diferença entre *overload* (sobrecarga) e *override* (sobrescrita) em C++?

7) Considere o problema do xadrez das listas passadas. De que maneira herança pode ser aplicada nesse cenário e por quais motivos? Quais classes herdam de quais classes? As heranças são public, protected ou private?