TÉCNICO LISBOA

Programação com Objectos Ano Lectivo 2014/2015, 1º Semestre **Exercício 03 (C++)**
Nome:
Número:
Data:
Válido se realizado o Exercício 03 (aula)

(1.0 val.) Modelação Básica, Construção de Objectos, Herança

Considere o conceito de animal, contendo as propriedades idade e nome. Considere ainda as especializações deste conceito: cão e gato. Os conceitos cão e gato têm, respectivamente, o peso e o número de vidas como propriedades adicionais. Considere que todos os animais dormem, mas que apenas os gatos trepam e que apenas os cães ladram. Implemente em C++ as classes que representam os conceitos: cada método dos anteriores deve ser implementado simplesmente com uma instrução que apresente uma cadeia de caracteres descritiva da acção. Implemente ainda as funções que permitem comparar instâncias para cada uma das classes (`operator==`), assim como as funções de linearização textual (`operator<<`). Os animais genéricos podem ser inicializados especificando-se apenas a idade (neste caso, o nome é vazio), mas podem ser também inicializados especificando-se ambas as propriedades (a acção para a idade é idêntica à anterior). Os cães e os gatos são sempre inicializados com explicitação das suas respectivas propriedades, i.e., idade, nome e peso ou número de vidas (conforme o caso). Construa uma aplicação (`main`) que ilustre a utilização das classes.

As assinaturas dos operadores indicados acima são (exemplo para uma classe Batata):

```
std::ostream &operator<<(std::ostream &o, const Batata &batata);
bool operator==(const Batata &b1, const Batata &b2);
```

Note-se que, quando o tipo do primeiro argumento é o da classe em questão, os operadores podem ser implementados como métodos da classe (os operadores que alteram o objecto devem estar dentro da classe, e.g., `operator=`, `operator+=`, etc.). Os outros operadores, i.e., cujo primeiro argumento não é do tipo da classe, podem ser incluídos no corpo da classe, desde que precedidos da palavra chave `friend` (esta palavra chave concede ao membro que assim declara acesso privilegiado à classe em causa). Exemplos para os casos acima:

```
classe Batata {
  //...
  friend std::ostream &operator<<(std::ostream &o, const Batata &batata);
  //...
  bool operator==(const Batata &batata); // a primeira batata é referida por "this"
  //...
};
```

`friend` é necessário nas situações em que a função não pode ser um método e deve (por alguma razão) ser incluída no corpo da classe (porque, por exemplo, tem com ela uma relação especialmente estreita). Tem, contudo, a consequência de permitir acesso sem restrições ao conteúdo da classe, razão pela qual deve ser utilizada com parcimónia.

Exemplo:

```
#include <iostream>

class Batata {
  int _casca;
public:
  Batata(int casca = 8) : _casca(casca) {}
  int casca() const { return _casca; }
  void casca(int casca) { _casca = casca; }
  bool operator==(const Batata &batata) { return _casca == batata.casca(); }
  friend std::ostream &operator<<(std::ostream &os, const Batata &batata) {
    os << batata.casca();
    return os;
  }
};
```