Aula 3

Objectivos: Utilização de herança simples.
Polimorfismo de inclusão e ligação dinâmica de métodos.

Problema A1

Continuando o problema A1 da aula anterior, pretende-se agora adaptar as classes feitas para a gestão de funcionários de uma empresa, em particular.

A1.1. Construa e teste uma nova classe `FUNCIONARIO` derivada da classe `PESSOA`, acrescentando os métodos e atributos necessários para aceder e guardar o número de funcionário e o número de meses desde o inicio do trabalho na empresa.
A estrutura da classe pode ser a seguinte:

```
class FUNCIONARIO: public PESSOA // herança simples
{
public:

  // novo construtor
  // definir número de funcionário
  // definir número de meses de empresa
  // escrever a informação do funcionário

protected:

  int p_num_func; // número de funcionário
  int p_num_meses_empresa; // número de meses de empresa
};
```

O método de escrita da informação sobre o funcionário deve manter a mesma assinatura do existente na classe `PESSOA`.

(**Sugestão:** Verifique qual a diferença entre ter o método de escrita de informação na classe `PESSOA` com e sem a definição de `virtual` (ligação dinâmica).)

A1.2. Reutilize o programa de gestão de PESSOAS feito na aula anterior incluindo esta nova funcionalidade (minimize as alterações necessárias).

A1.3. Acrescente o método (no caso, função real): `salario`, que devolve o valor do salário mensal (em Euros) de cada funcionário. Defina esse método na classe `FUNCIONARIO` devolvendo o salário zero.

Defina as classes derivadas de `FUNCIONARIO`: `ELEMENTO_DIRECCAO`, `ENGENHEIRO_PRODUCAO` e `FUNCIONARIO_FABRIL`, considerando que os salários para cada tipo de funcionário são calculados respectivamente pelas fórmulas:

$$SalElemDir = 3000 + 300 * numero\_de\_anos\_na\_empresa$$
$$SalEngProd = 1500 + 150 * numero\_de\_anos\_na\_empresa$$
$$SalFuncFab = 500 + 50 * numero\_de\_anos\_na\_empresa$$

Problema A2

Altere o problema A2 da aula anterior generalizando-o para uma colecção de peças de museu. Para além de quadros o museu pode também ter esculturas, serigrafias, estando todas estas peças organizadas por autor.

Problema B1

B1.1. Esta tarefa baseia-se na classe `CLivro` (livro) considerada na aula 1. Implemente uma lista ligada (`CList`) de ponteiros para livros. Construa métodos que permitem adicionar, consultar e apagar livros numa determinada posição, verificar se a lista está vazia, determinar o cumprimento da lista e visualizar todos os livros contidos na lista. A lista não deve ter restrições no seu comprimento.

B1.2. Deriva da classe `CList` uma classe `CSortedList` que permita manter uma lista de ponteiros para livros ordenados de acordo com o ano de publicação. Implemente os métodos relevantes recorrendo quando necessário aos métodos semelhantes definidos na classe `CList`.

B1.3. Em todos os construtores/destrutores das classes `CList` e `CSortedList` imprima no écran os seus nomes. Por exemplo:

```
cout << " CList::~CList ()" << endl;
```

B1.4. Faça um programa que exemplifique o uso das classes `CList` e `CSortedList`.

B1.5. Corra o programa e repare na ordem em que os diferentes construtores/destrutores são chamados.

Problema B2

Imagine que tem uma classe com a estrutura seguinte:

```
class CSimple
{
public:
  CSimple  (int nSize) : m_nSize(nSize)  { };
  ~CSimple()  { } ;

private:
  int m_nSize;
};
```

A função main é implementada de maneira seguinte:

```
int main(int argc, char* argv[])
{
  {
```

```
    CSimple s1;                    //1
    CSimple s2(2);                 //2
    CSimple* ps = new CSimple(3);  //3
    delete ps;                     //4
  }                                //5
  return 0;
}
```

Tente responder as perguntas seguintes sem recorrer ao uso de computador.

a) O código acima está correcto?

b) Que funções são chamadas em cada uma das linhas numeradas?