Universidade do Minho	Escola de Engenharia da Universidade do Minho **Mestrado Integrado em Eng. Electrónica Industrial e Computadores** # Complementos de Programação de Computadores	**2014/2015** **MIEEIC** **(1º Ano)** **2º Sem**
	Luís Paulo Reis	
	Aula Prática 8: Listas	

Objectivos:

Esta Folha de Exercícios destina-se a:

• Compreender a utilização de listas ligadas;

Os exercícios aqui propostos deverão ser realizados no mais simples ambiente de desenvolvimento possível para a linguagem C: editor de texto de programação ou editor DevC++ e ferramentas da GCC (GNU Compiler Collection) e afins.

Exercício 8

Faça download do ficheiro ***Prog2_Aula8.rar*** do blackboard e descomprima-o (contém os ficheiros ***crianca.h, jogo.h***, e ***Test.cpp***) Deverá realizar esta ficha respeitando a ordem das alíneas. Para a classe **Jogo** predefinida no ficheiros **Jogo.h**, considere realizar a implementação em ficheiro **.cpp** respectivo.

"*Pim, Pam, Pum, cada bola mata um, p'ra galinha e para o peru quem se livra és mesmo tu!*" Recorde este jogo de crianças, cujas regras são muito simples, como indicado a seguir: Uma das crianças começa a dizer a frase, e em cada palavra vai apontando para cada uma das crianças em jogo. Ao chegar ao fim da lista de crianças, volta ao início, ou seja a ela mesma. A criança que está a ser apontada, quando é dita a última palavra da frase, livra-se e sai do jogo. A contagem recomeça na próxima criança. Perde o jogo a criança que restar.

Use uma lista (pode usar a classe *list* da STL) para implementar este jogo. Os elementos da lista são objectos da classe **Crianca**, definida abaixo e implementada no ficheiro *criança.h*:

```
class Crianca
{
    string nome; int idade; public: Crianca(); Crianca(string nm, int id);
    Crianca(const Crianca &c1);
    bool operator == (const Crianca & c2) const;
    bool operator != (const Crianca & c2) const;
    Crianca & operator = (const Crianca & c2);
    int getIdade() const; string getNome() const; void escreve();
};
```

A classe **Jogo** também está previamente declarada, como a seguir:

```
class Jogo
{
    list<Crianca> criancas;
  public:
```

```
    Jogo();
    Jogo(list<Crianca>& lc2); void insereCrianca(Crianca &c1);
    const list<Crianca>& getCriancasJogo() const;
    string escreve();
    int nPalavras(string frase);
    Crianca& perdeJogo(string frase);
    list<Crianca>& inverte();
    list<Crianca> divide(int id);
    void setCriancasJogo(list<Crianca>& l2);
    bool operator==(Jogo& j2);
    list<Crianca> baralha(const list<Crianca>& lista);
};
```

a) Implemente o seguinte membro-função da classe **Jogo**:
`void Jogo::insereCrianca(Crianca & c1)`
Esta função adiciona a criança *c1* ao jogo. Implemente também o membro-função da classe Jogo:
`const list<Crianca>& Jogo::getCriancasJogo() const`
que retorna a lista de crianças correntemente a jogarem.

b) Implemente o seguinte membro-função da classe **Jogo**: `string Jogo::escreve() const`
Esta função retorna uma *string* com todas as crianças que estão em jogo num dado momento para que possa, por exemplo, ser exibida no ecrã.

c) Implemente o membro-função da calsse **Jogo**: `int Jogo::nPalavras(string frase)`
que determina o número de palavras existentes numa dada string *frase*.

d) Implemente o seguinte membro-função da classe **Jogo**:
`Crianca& Jogo::perdeJogo(string frase)`
Esta função determina qual a criança que perde o jogo quando a frase utilizada no jogo é *frase*.

e) Implemente o seguinte membro-função da classe **Jogo**:
`list<Crianca>& Jogo::inverte()`
Esta função inverte a ordem dos jogadores, na lista *criancas*, em relação à ordem original e retorna a referência para a lista de crianças.

f) Implemente o seguinte membro-função da classe **Jogo**:
`list<Crianca> Jogo::divide(int id)`
Esta função remove do jogo as crianças de idade superior ao valor *id* especificado como parâmetro, e retorna uma nova lista com as crianças que foram removidas.

g) Implemente o *overload* do operador abaixo para a classe **Jogo**:
`bool Jogo::operator==(Jogo& j2)`
Dois jogos são considerados iguais se possuem nas suas respectivas listas as mesmas crianças, na mesma ordem.

h) Implemente o membro-função abaixo para a classe **Jogo**:
`list<Object> Jogo::baralha(const list<Object> & lista)`
Esta função cria uma nova lista onde as crianças de *lista* são colocadas em uma posição determinada aleatoriamente (use a função *rand()* para gerar números aleatórios).