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 7: Pilhas e Filas	

Objectivos:

Esta Folha de Exercícios destina-se a:

• Compreender as estruturas de dados: pilhas e filas;

• Analisar a complexidade de algoritmos simples;

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 7

Faça download do ficheiro ***progr_PihasFilas.tar*** do blackboard e descomprima-o (contém os ficheiros ***StackExt.h***, ***Balcao.h***, ***Balcao.cpp*** e ***Test.cpp*** bem como alguns ficheiros de suporte aos testes – *cycle.h* e *performance.h*).

Note que os testes unitários deste projeto estão comentados. Retire os comentários à medida que vai implementando os testes. Deverá realizar esta ficha respeitando a ordem das alíneas.

Deve fazer a implementação da classe *StackExt* no ficheiro ***StackExt.h*** e as restantes no ficheiro ***balcao.cpp***.

1. Escreva a classe **StackExt**, que implementa uma estrutura do tipo pilha (*stack*) com um novo método: `findMin`. O método `findMin` retorna o valor do menor elemento da pilha e deve ser realizado em tempo constante, O(1). Deve implementar os seguintes métodos:

```
bool empty() const // verifica se a pilha está vazia

T top() const // Obtém o próximo elemento

void pop() // Remove elemento

void push(const T & val) // insere elemento

T findMin() const // retorna valor menor elemento
```

Sugestão: Use a classe *stack* da biblioteca STL. Use duas stacks: uma para guardar todos os valores, e outra para guardar os valores mínimos à medida que estes vão surgindo.

2. Pretende-se desenvolver um simulador de um balcão de embrulhos. Crie uma classe **Cliente** com os seguintes membros (construtor vazio deve gerar aleatoriamente um número de presentes entre 1 e 5).

```
class Cliente {
    int numPresentes;
  public:
    Cliente();
    int getNumPresentes();
};
```

Crie uma outra classe **Balcao** que simula o andamento de uma fila de clientes.

```
class Balcao {
    queue<Cliente> clientes;
    // tempo que demora a embrulhar um presente
    const int tempo_embrulho;
    // tempo em que irá ocorrer a proxima chegada/saida cliente fila
    int prox_chegada, prox_saida;
    int tempo_actual; // tempo actual da simulacao
    int clientes_atendidos;
  public:
    Balcao(int te=2); // te = tempo_embrulho
    void proximoEvento();
    int getTempoActual();
    int getProxChegada();
    void chegada();
    void saida();
    friend ostream & operator << (ostream & out, const Balcao & b1);
    int getTempoEmbrulho() const;
    int getProxSaida();
    int getClientesAtendidos();
    Cliente & getProxCliente();
};
```

a) Implemente o construtor vazio da classe **Cliente** que deverá gerar aleatoriamente um número de presentes entre 1 e 5. Implemente o membro-função público `getNumPresentes()` que devolve o número de presentes do cliente (*numPresentes*).

b) Implemente o construtor da classe **Balcao**. Este deve inicializar o tempo de simulação a zero (*tempo_actual*), gerar aleatoriamente o tempo de chegada do próximo cliente (*prox_chegada*) com um valor entre 1 e 20 e a próxima saída (*prox_saida*) a zero. Implemente os membros-função públicos da classe **Balcao**:

- `int getTempoActual()` que devolve o tempo actual.
- `int getProxChegada()` que devolve o tempo de chegada do próximo cliente.
- `int getClienteAtendidos()` que devolve o número de clientes atendidos.
- `int getTempoEmbrulho()` que devolve o *tempo_embrulho*.
- `int getProxSaida()` que devolve o *prox_saida*.

- `Cliente & getProxCliente()` que devolve o cliente seguinte da fila *clientes*. Se a fila estiver vazia, deve lançar a excepção `FilaVazia()` que contém o método *getMsg()* que devolve "Fila Vazia".

c) Implemente o membro-função:

- `string Balcao::chegada()`

  Esta função simula a chegada de um novo cliente à fila e deverá:

  - Criar um novo cliente e inseri-lo na fila.
  - Gerar aleatoriamente o tempo de chegada do próximo cliente (*prox_chegada*) com um valor entre 1 e 20.
  - Actualizar o tempo de saída do próximo cliente (*prox_saida*) se necessário (no caso de a fila estar vazia). A próxima saída é igual ao *tempo_actual* + *numPresentes* do cliente criado * *tempo_embrulho*.
  - Escrever no monitor o instante actual e a informação sobre o cliente que chegou à fila.

d) Implemente o membro-função:

- `void Balcao::saida()`

  Esta função simula a saída de um novo cliente da fila e deverá:

  - Tratar convenientemente a excepção, no caso de a fila estar vazia (sugestão, use a função *getProxCliente()* para obter o cliente a sair).
  - Retirar o primeiro cliente da fila.
  - Actualizar o tempo de saída do próximo cliente (*prox_saida*) caso a fila não fique vazia. Se ficar, a *prox_saida* deverá ficar com valor zero.
  - Escrever no monitor o instante actual e a informação sobre o cliente que saiu da fila.

e) Implemente o membro-função:

- `void Balcao::proximoEvento()`

  Esta função invoca o próximo evento (chegada ou saída), de acordo com os valores de *prox_chegada* e *prox_saida*. Actualiza também o valor de *tempo_actual* de acordo.

f) Implemente o operador de escrita (`operator <<`). Esta função deve imprimir no monitor o número de clientes atendidos e número de clientes em espera.