7ª aula prática - Pilhas e Filas

Faça download do ficheiro ***aeda1819_fp07.zip*** 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*).

- Deverá realizar esta ficha respeitando a ordem das alíneas

Enunciado

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()` // retorna o elemento no topo da pilha
   - `void pop()` // remove elemento
   - `void push(const T & val)` // insere elemento
   - `T &findMin()` // retorna valor do 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.
   Nota: os testes unitários podem ser demorados.

2. Pretende-se desenvolver um simulador de um balcão de embrulhos. O balcão inclui uma fila de clientes a ser atendidos (*clientes*). Os clientes vão chegando em instantes determinados aleatoriamente e ficam à espera na fila para ser atendidos. O tempo de atendimento de um cliente varia com o número de presentes que este tem para embrulhar. O membro-dado *tempo_atual* indica o instante atual (relógio) da simulação. O membro-dado *prox_chegada* indica o instante em que ocorre a chegada do próximo cliente à fila (as chegadas são geradas aleatoriamente). O membro-dado *prox_saida* indica o instante em que será finalizado o atendimento do primeiro cliente da fila.

   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() const;
};
```

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

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

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 retorna o número de presentes do cliente (*numPresentes*).

b) Implemente o construtor da classe **Balcao**. Este deve inicializar o tempo de simulação a zero (*tempo_atual*), 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 getTempoAtual() const  // retorna tempo atual.`
- `int getProxChegada() const  // retorna tempo chegada próximo cliente`
- `int getClienteAtendidos() const  // retorna nº de clientes atendidos`
- `int getTempoEmbrulho() const  // retorna` *`tempo_embrulho`*
- `int getProxSaida() const  // retorna` *`prox_saida`*
- `Cliente & getProxCliente()  // retorna` o cliente seguinte da fila *clientes*. Se a fila estiver vazia, deve lançar a exceção `FilaVazia()`. Esta exceção contém o método string *getMsg() const* que devolve "Fila Vazia".

c) Implemente o membro-função:

```
void Balcao::chegada()
```

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

- 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
- Atualizar o tempo de saída do próximo cliente (*prox_saida*) se necessário (isto é, no caso de a fila estar vazia antes desta chegada). A próxima saída é igual ao *tempo_atual* + *numPresentes* do cliente criado * *tempo_embrulho*
- Escrever no monitor o instante atual e a informação sobre o cliente que chegou à fila (ver os testes unitários para saber o formato e conteúdo da informação)

d) Implemente o membro-função:

```
void Balcao::saida()
```

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

- Tratar convenientemente a exceção, no caso de a fila estar vazia (sugestão: use a função *getProxCliente()* para obter o cliente a sair)
- Retirar o primeiro cliente da fila
- Atualizar o tempo de saída do próximo cliente (*prox_saida*)
- Escrever no monitor o instante atual e a informação sobre o cliente que saiu da fila (ver os testes unitários para saber o formato e conteúdo da informação)

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*. Atualiza também o valor de *tempo_atual* de acordo.

Nota: o teste unitário nao falha, verifique na consola os valores.

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

Nota: o teste unitário nao falha, verifique na consola os valores.