6ª aula prática - Pilhas e Filas

Faça download do ficheiro ***aeda2021_p06.zip*** e descomprima-o (contém a pasta ***lib***, a pasta ***Tests*** com os ficheiros ***stackExt.h***, ***counter.h***, ***counter.cpp, exceptions.h*** e ***tests.cpp*** bem como alguns ficheiros de suporte aos testes – *cycle.h* e *performance.h*, e os ficheiros ***CMakeLists*** e ***main.cpp***).

- Deverá realizar a 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 **Client** com os seguintes membros (construtor vazio deve gerar aleatoriamente um número de presentes entre 1 e 5).

```
class Client {
    unsigned numGifts;
public:
    Client();
    unsigned getNumGifts() const;
};
```

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

```
class Counter {
   queue<Client> clients;
   const unsigned wrappingTime;   // tempo que demora a embrulhar um presente
   unsigned nextEnter, nextLeave; // tempo em que ocorre a proxima
                                     chegada/saida cliente fila
   unsigned actualTime;           // tempo atual da simulacao
   unsigned numAttendedClients;
public:
    Counter(unsigned wt=2);          // wt = tempo_embrulho
    unsigned getActualTime() const;
    unsigned getNextEnter() const;
    unsigned getNumAttendedClients() const;
    unsigned getWrappingTime() const;
    unsigned getNextLeave() const;
    Client & getNextClient();
    void enter();
    void leave();
    void nextEvent();
    friend ostream & operator << (ostream & out, const Counter & c1);
};
```

a) Implemente o construtor vazio da classe **Client** que deverá gerar aleatoriamente um número de presentes entre 1 e 5. Implemente o membro-função público `getNumGifts()` que retorna o número de presentes do cliente (*numGifts*).

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

- `unsigned getActualTime() const // retorna tempo atual.`
- `unsigned getNextEnter() const // retorna tempo chegada próximo cliente`
- `unsigned getNumAttendedClients() const // retorna n°clientes atendidos`
- `unsigned getWrappingTime() const // retorna` *wrappingTime*
- `unsigned getNextLeave() const // retorna` *nextLeave*
- `Cliente & getNextClient()` `// retorna o cliente seguinte da fila` *clientes*. Se a fila estiver vazia, deve lançar a exceção **EmptyQueue**. Esta exceção contém o método string *getMsg() const* que devolve a string "Empty Queue".

c) Implemente o membro-função:

```
void Counter::enter()
```

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 (*nextEnter*) com um valor entre 1 e 20
- Atualizar o tempo de saída do próximo cliente (*nextLeave*) se necessário (isto é, no caso de a fila estar vazia antes desta chegada). A próxima saída é igual ao *actualTime* + *numGifts* do cliente criado * *wrappingTime*
- 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 Counter::leave()
```

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 *getNextClient()* para obter o cliente a sair)
- Retirar o primeiro cliente da fila
- Atualizar o tempo de saída do próximo cliente (*nextLeave*)
- 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 Counter::nextEvent()`

Esta função invoca o próximo evento (chegada ou saída), de acordo com os valores de *nextEnter* e *nextLeave*. Atualiza também o valor de *actualTime* de acordo.

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