CI2 Parte prática. Duração: 1h45m

Submeter a resolução da prova num ficheiro zip (incluir os ficheiros *.cpp, *.h). Não necessita incluir o ficheiro tests.cpp

Suponha um armazém (***Warehouse***) que vai recebendo sucessivamente caixas (***Box***) de dois tipos: estreitas e altas (***ThinTallBox***) e caixas largas e baixas (***WideFlatBox***), contendo Objetos (***Objects)*** no seu interior. As caixas são colocadas numa fila de processamento por ordem de entrada. Todos os dias a fila de caixas é processada, sendo despachadas (retirando-as da fila) todas as caixas que tenham atingido o seu tempo limite em armazém, ou estejam cheias. As caixas que não forem despachadas são simplesmente processadas, com a adição de um novo Objeto da fila de objetos disponíveis (ou a adição de um objeto "dummy" caso não exista nenhum objeto que caiba na caixa disponível) e são colocadas no final da fila respetiva de caixas para serem novamente processadas no dia seguinte.

Uma caixa (***Box***) permite armazenar objetos (***Objects***) numa determinada ordem. Suponha a existência de dois tipos de caixas: Caixas estreitas e altas (***ThinTallBox***) exigem que o acesso aos objetos de baixo se faça retirando os objetos de cima (funciona como uma pilha). Caixas largas e baixas (***WideFlatBox***) não permitem colocar itens em cima uns dos outros, mas facilitam o acesso a todos os objetos da caixa. As caixas são criadas com um número de dias para envio (days_to_send) aleatório entre 2 e 5;

Cada um destes conceitos encontra-se parcialmente declarado a seguir. Uma caixa (***Box***) tem uma determinada capacidade (***capacity***) e um tempo máximo (em dias) até ser enviada (***days_to_send)***. De modo a saber, em tempo constante, o espaço ocupado pelos respetivos objetos, o membro-dado ***content_size*** contém o espaço ocupado pelos objetos que estão na caixa. Cada objeto (***Object***) tem um nome único que o identifica (***name***) e um tamanho (***size***).

```
class Warehouse {
    queue<ThinTallBox> boxes;
    queue<Object> objects;
public:
    Warehouse();
    queue<ThinTallBox> getBoxes();
    queue<Object> getObjects();
    void addObject(Object o1);
    int InsertObjectsInQueue(vector <Object>
        objects);
    Object RemoveObjectQueue(int maxSize);
    int addPackOfBoxes(vector <ThinTallBox> boV);
    vector<ThinTallBox> DailyProcessBoxes();
};

class Object {
    string name;
    double size;
public:
    Object() {};
    Object(string name, double size);
    string getName() const;
    double getSize() const;
    bool operator ==(const Object& obj) const;
    bool operator <(const Object& obj) const;
    friend ostream & operator << (ostream & out,
const Object & o1);
};
```

```
class Box {
    const double capacity;
    double content_size;
    int days_to_send;
public:
    Box(double capacity);
    virtual void insert(Object object) = 0;
    virtual void remove(Object object) = 0;
    const double getCapacity() const;
    double getContentSize() const;
    void setContentSize(double contentSize);
    bool empty() const;
    virtual const Object& next() const = 0;
    virtual void sortObjects() = 0;
    Object removeNext();
};

class ThinTallBox : public Box {
    stack<Object> objects;
public:
    ThinTallBox(double capacity);
    ThinTallBox(double capacity,
      stack<Object> objects, double content_size);
    void insert(Object object);
    void remove(Object object);
    const Object& next() const;
    void sortObjects();
};
class WideFlatBox : public Box {
    vector<Object> objects;
public:
    WideFlatBox(double capacity);
    void insert(Object object);
    void remove(Object object);
    const Object& next() const;
    void sortObjects();
};
```

a) [**2.5 valores**] Implemente na classe ***WideFlatBox*** o membro-função

```
void WideFlatBox::remove(Object object)
```

que remove o objeto ***object*** da caixa. O espaço ocupado deve ser atualizado. No caso de o objeto não se encontrar na caixa, deve ser lançada a exceção ***InaccessibleObjectException***, já definida.

b) [**2.5 valores**] Implemente na classe ***ThinTallBox*** o membro-função

```
void ThinTallBox::insert(Object object)
```

que insere na caixa o objeto ***object***, se este couber. O espaço ocupado pelos objetos deve ser atualizado. No caso de não caber deve ser lançada a exceção ***ObjectDoesNotFitException***, já definida.

c) [**2.5 valores**] Implemente na classe ***Box*** o membro-função

```
Object Box::removeNext()
```

que retira da caixa o próximo objeto (dado pelo membro-função ***next***), retornando-o. No caso da caixa estar vazia, deve ser lançada a exceção ***InaccessibleObjectException***, já definida.

d) [**2.5 valores**] Numa caixa estreita e alta, é conveniente que os objetos mais pesados fiquem em baixo. Implemente na classe ***ThinTallBox*** o membro-função

```
void ThinTallBox::sortObjects()
```

que reordena os objetos que estão na caixa. *(Sugestão: pode usar uma caixa larga e baixa como auxiliar…)*

e) [**2.5 valores**] Suponha que dispõe de um vetor preenchido com objetos. Introduza os objetos, ordenados por ordem decrescente de tamanho na fila de objetos do armazém. A função deve retornar o número de objetos inseridos.

```
int Warehouse::InsertObjectsInQueue(vector <Object> objects)
```

f) [**2.0 valores**] Na classe **Warehouse**, implemente uma função que selecione da fila de objetos o primeiro Objeto com tamanho máximo maxSize, retirando-o da Fila e retornando-o. No final da execução da função, a única alteração à fila deve ser a remoção do objeto considerado. Se não existir um objeto com dimensão menor ou igual a maxSize, deve ser retornado um novo objeto criado com nome "dummy" e dimensão maxSize.

```
Object Warehouse::RemoveObjectQueue(int maxSize)
```

g) [**2.0 valores**] Na classe Warehouse, implemente uma função que adiciona um conjunto de caixas estreitas ao armazém. A função deve retornar o número de caixas total com que o armazém fica.

```
int Warehouse::addPackOfBoxes(vector <ThinTallBox> boV)
```

h) [**3.5 valores**] Implemente a função que processa a fila de caixas estreitas.

```
vector<ThinTallBox> Warehouse::DailyProcessBoxes()
```

A função deve verificar para cada caixa, se a mesma está cheia, ou se é possível colocar mais um objeto (existente ou dummy), verificando também se a data de envio expirou (chegou a 0). Se a caixa está cheia, deve ser despachada, isto é, eliminada da fila de caixas. Se a data de envio expirou, a caixa deve ser processada e despachada (eliminada da fila de caixas). Caso contrário, a caixa deve ser processada e movida para o final da fila, decrementando o seu prazo de envio (dado que se passou um dia). Processar uma caixa equivale a adicionar a esta o primeiro objeto da fila de objetos que ainda cabe na caixa (que deve ser um dummy" de tamanho 0, se não existir nenhum objeto de tamanho adequado). A função deve retornar o vetor de caixas despachadas.

Submeter a resolução da prova num ficheiro zip (inclui ficheiros *.cpp e *.h). Não necessita incluir o ficheiro Tests.cpp