1.

a) Contentores de sequência:

I. Crie um projeto com o ficheiro “main_sc.cpp”.
II. Defina um vector (STL) de “string”.
III. Preencha o vector (STL) de “string” usando o vector sve.
IV. Mostre o vector no ecrã usando a iteração “for each”.
V. Defina uma list (STL) de “string”.
VI. Preencha a list (STL) de “string” usando o vector sve.
VII. Mostre a list no ecrã usando iteradores.
VIII. Inverta a list.
IX. Ordene a list.
X. Crie uma segunda list (STL) de “string” só com o elemento “Sexto”.
XI. Junte esta segunda list à primeira.
XII. Mostre a list no ecrã usando a iteração “for each”.

b) Contentores adaptados:

I. Crie um projeto com o ficheiro “main_ca.cpp”.
II. Defina uma stack (STL) de “string”.
III. Preencha a stack (STL) de “string” usando o vector sve.
IV. Defina uma queue (STL) de “string”.
V. Preencha a queue (STL) de “string” usando o vector sve.
VI. Remova todos os elementos da stack, mostrando-os no ecrã (FILO).
VII. Remova todos os elementos da queue, mostrando-os no ecrã (FIFO).

c) Contentores associativos:

I. Crie um projeto com o ficheiro “main_ac.cpp”.
II. Defina um **set** (STL) de “string”.
III. Preencha o **set** (STL) de “string” usando o vector sve.
IV. Mostre o **set** no ecrã usando a iteração “for each”.
V. Defina um **map** (STL) com chave “char” e informação “string”.
VI. Mostre no ecrã a informação do **map** respeitante à 3ª chave
VII. Mostre o **map** no ecrã usando iteradores de ordem inversa.

d) Algoritmos STL Contentores de sequência:

I. Crie um projeto com o ficheiro "main_alg.cpp".

II. Defina uma **list** (STL) de "char".

III. Preencha a **list** (STL) de "char" usando o vector sve.

IV. Verifique se a **list** tem o 3º elemento usando o algoritmo genérico **find**.

V. Verifique qual o máximo da **list** usando o algoritmo genérico **max_element**.

2. Considere a classe **InFront** que guarda várias sequências de objetos num vetor, identificadas pelo índice do vetor, sendo cada sequência de objetos não duplicados guardada numa deque. Nesta classe o processo de juntar objetos consiste em aceder à respetiva sequência sendo o novo objeto sempre inserido no início da sequência. Se o objeto já existe é removido da sequência e inserido no início desta. Esta estratégia em frente baseia-se no fato dos elementos recentemente acedidos terem maior probabilidade de serem acedidos novamente.

A classe para além dos construtores, destrutor deve disponibilizar métodos para:

a) Inserir objetos numa sequência

b) Retirar os n últimos elementos de uma sequência

c) Sobrecarga do operador << para visualizar todas as sequências de objetos.

3. Pretende-se simular um armazém no qual opera um empilhador que tem por missão arrumar caixotes de qualquer tipo. Estes caixotes podem ser sobrepostos até um número máximo de 5 caixotes. O empilhador só coloca um caixote no chão se o não puder colocar em cima de outro caixote. O empilhador só retira um caixote do chão quando não existirem caixotes sobrepostos.

a) Defina a classe **Warehouse**

Adicione à classe os métodos:

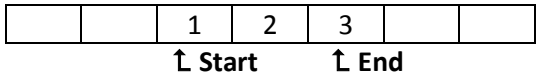
b) Empilha

c) Desempilha

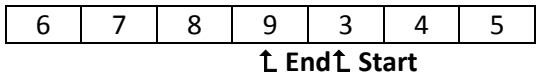
d) Sobrecarga do operador << que lista todos os caixotes para o écran

4. Um RingBuffer é uma estrutura de dados variante da fila de espera de dimensão fixa. Tal como a fila de espera funciona segundo a disciplina FIFO, no entanto quando o RingBuffer está cheio e é inserido um novo objeto, o objeto que se encontra no início é substituído pelo novo objeto. Isto torna o RingBuffer ideal para buffer de entrada de teclado ou streaming de dados, como o som em tempo real ou de vídeo, onde os dados antigos são menos importantes e são substituídos por novos. As figuras exemplificam o funcionamento do RingBuffer.

RingBuffer com 3 elementos



RingBuffer cheio com os dois primeiros elementos substituídos:



a) Usando a classe deque implemente a classe **RingBuffer** os construtores e o destrutor.

b) Faça o método insert.

c) Elabore o método remove tendo em atenção que este retira sempre do RingBuffer o elemento mais antigo.

d) Visualize no écran a inserção de n elementos no RingBuffer

5. O objetivo da classe **Dictionary** é guardar palavras com a respetiva lista de sinónimos. Relativamente aos sinónimos estes deverão ser guardados por ordem alfabética da sua terminação, de modo a auxiliar a busca de palavras sinónimas com igual terminação.

a) Com base nestas indicações defina a classe **Dictionary**

Desenvolva métodos que realizem sobre o dicionário as operações:

b) Inserir palavra/ [sinónimo | lista de sinónimos]

c) Pesquisar sinónimos de uma palavra

d) Eliminar palavra e respetiva lista de sinónimos

e) Listar o dicionário por ordem alfabética das suas palavras e para cada uma a respetiva lista de sinónimos

6. Escreva a versão iterativa e recursiva dos algoritmos de ordenação:

- **a)** InsertionSort
- **b)** MergeSort
- **c)** QuickSort