

CADERNO DE EXERCÍCIOS 03

7. Utilize a biblioteca STL para especificar uma classe cuja principal responsabilidade é a de disponibilizar alguns métodos capazes de gerir uma sequência de números inteiros, armazenados em memória na forma de lista simplesmente ligada (NOTA: no **Anexo A**, desta folha de exercícios, encontra alguma documentação relativa à classe *template 'list'* da *Standard Template Library*, ou STL). Esta classe gestora, "*Gestor_Num*", além dos seus construtores, destrutor e métodos de acesso, deve disponibilizar um conjunto de funcionalidades que permitam também:
- a) Inserir um novo número inteiro na lista;
 - b) Remover um número da lista;
 - c) Listar, no ecrã, todos os números da lista;
 - d) Obter a posição absoluta de um determinado número, na lista.
8. Considere o exercício 6 da folha de exercícios anterior, em que uma fábrica de automóveis pretende desenvolver uma aplicação para fazer a gestão dos seus funcionários.
- 8.1. Recorrendo à classe *template 'list'*, crie uma classe gestora "*Gestor_Func*" composta por uma lista ligada de apontadores para funcionários, que podem ser operários ou administrativos, consoante a necessidade.
- 8.2. A classe "*Gestor_Func*", além dos seus construtores, destrutor e métodos de acesso, deve possuir também os métodos abaixo listados para gerir a lista dos funcionários:
- a) Um método para adicionar um funcionário à lista (AddFunc);
 - b) Um método para remover um funcionário da lista (RemoveFunc);
 - c) Listar, no ecrã, a informação de todos os funcionários (ShowFunc);
 - d) Sobrecarga dos operadores >> e << ;
 - e) Ler os dados de N funcionários, para a lista, a partir de um ficheiro (ReadFile).
 - f) Gravar os dados dos funcionários para um ficheiro (SaveFile);
- 8.3. Desenvolva um programa que:
- a) Carregue, num objeto "*GestorFunc*", os dados dos funcionários, lidos a partir do ficheiro "funcionarios.txt";
 - b) insira numa lista ligada, dois novos objetos do tipo Funcionario: (1 operário e 1 administrativo) com os seguintes atributos:
O:Zeca Estacionancio;1/1/1985;Rua A n9;25050;Pintura;1100;15;nao;
A:Adrusila Lopes;6/6/1996;Rua E n6;25051;RecHumanos;900;0;
 - c) mostre no ecrã os dados de todos os funcionários da lista;
 - d) remova o funcionário Joca Gaio;
 - e) grave num ficheiro a informação de todos os funcionários (um por linha) com os seus vários campos separados por ';' .
9. Considere o exercício 8 deste caderno de exercícios. Utilizando o *container Map*, adicione um método na classe gestora que, com base numa lista de funcionários já preenchida, crie uma tabela associativa entre números mecanográficos e vencimentos. Teste este método no *main* mostrando no ecrã o vencimento de um funcionário escolhido pelo utilizador.

NOTA: pode consultar alguma informação sobre este *container* a partir dos seguintes links:
documentação: <http://www.cplusplus.com/reference/map/map/>
exemplos: <http://thispointer.com/stdmap-tutorial-part-1-usage-detail-with-examples/>

ANEXO A

A Standard Template Library (STL) é considerada como um dos elementos distintivos mais importantes do C++ e tem crescido muito nos últimos anos. Esta biblioteca fornece classes e funções de propósito geral padronizadas (templates) que implementam muitos algoritmos e estruturas de dados frequentemente utilizados. Por exemplo, inclui suporte para vetores, listas, filas e pilhas. Também define várias rotinas de acesso e manipulação. Como a STL é construída a partir de classes modelo, os seus algoritmos e estruturas de dados podem ser aplicados a quase qualquer tipo de dados.

No núcleo do STL estão quatro itens fundamentais:

- Containers (de listas, por exemplo)
- Iterators
- Algorithms
- Function Objects

Alguns pontos sobre o *container* de listas STL:

- implementa uma estrutura de dados clássica, na forma de lista;
- suporta uma lista linear bidireccional dinâmica;
- ao contrário de uma matriz C++, os objetos que a lista STL contém não podem ser acedidos diretamente (ou seja, através de um índice);
- é definido como uma classe modelo (template), o que significa que pode ser personalizado para armazenar objetos de qualquer tipo;
- responde como uma lista não ordenada (ou seja, a ordem da lista não é mantida); no entanto, existem funções disponíveis para ordenar a lista.

O *container* de listas biblioteca STL disponibiliza vários métodos para interagir com os elementos de uma lista ligada. Entre os métodos cujo uso é mais comum, encontram-se:

size	<pre>size_type size() const;</pre> <p>Devolve o número de itens (elementos) armazenados na lista. O tipo <code>size_type</code> é um unsigned integer.</p> <pre>// Loop as long as there are still elements in the list. while (list1.size() > 0) { ... }</pre>
empty	<pre>bool empty() const;</pre> <p>Devolve um valor verdadeiro se o número de elementos na lista é zero, e falso caso contrário.</p> <pre>if (list1.empty()) { ... }</pre>
push_back push_front	<pre>void push_back(const T& x); void push_front(const T& x);</pre> <p>Adiciona o elemento <code>x</code> no final (ou início) da lista. (<code>T</code> é o tipo de dados dos elementos da lista).</p> <pre>list<int> nums; nums.push_back (3); nums.push_back (7);</pre>

	<pre>nums.push_front(10); // 10 3 7</pre>
front back	<pre>T& front(); const T& front() const; T& back(); const T& back() const;</pre> <p>Devolve a referência do primeiro (ou último) elemento da lista (válido apenas se a lista não estiver vazia). Esta referência pode ser utilizada para aceder ao primeiro (ou último) elemento da lista.</p> <pre>list<int> nums; nums.push_back(33); nums.push_back(44); cout << nums.front() << endl; // 33 cout << nums.back() << endl; // 44</pre>
begin	<pre>iterator begin();</pre> <p>Devolve um <code>iterator</code> que referencia o início da lista.</p>
end	<pre>iterator end();</pre> <p>Devolve um <code>iterator</code> que referencia a posição imediatamente a seguir ao último elemento da lista.</p>
insert	<pre>iterator insert(iterator position, const T& x);</pre> <p>Inserir o elemento <code>x</code> (o tipo <code>T</code> é o tipo de dados dos elementos da lista) na lista, na posição especificada pelo <code>iterator</code> (antes do elemento, se existir, que estava anteriormente na posição do iterador). O valor de retorno é um <code>iterator</code> que aponta para a posição do elemento inserido.</p> <pre>nums_iter = find(nums.begin(), nums.end(), 15); if (nums_iter != nums.end()) { nums_iter = nums.insert(nums_iter, -22); cout << "Inserted element " << (*nums_iter) << endl; }</pre>
erase	<pre>iterator erase(iterator position); iterator erase(iterator first, iterator last);</pre> <p>Apaga (remove) um elemento ou um intervalo de elementos de uma lista. No caso de um intervalo, esta operação exclui os elementos desde a primeira posição do iterador até, mas não incluindo, a segunda posição do iterador. O iterador retornado, aponta para o elemento imediatamente após o último que foi apagado. Para uma maneira alternativa de apagar todos os elementos, veja a descrição do <code>clear()</code> mais abaixo.</p> <pre>nums.erase(nums.begin(), nums.end()); // Remove all elements; ... nums_iter = find(nums.begin(), nums.end(), 3); // Search the list. // If we found the element, erase it from the list. if (nums_iter != nums.end()) nums.erase(nums_iter);</pre>
clear	<pre>void clear();</pre> <p>Apaga todos os elementos de uma lista.</p> <pre>nums.clear(); // Remove all elements;</pre>

pop_front pop_back	<pre>void pop_front(); void pop_back();</pre> <p>Apaga o primeiro (ou ultimo) elemento de uma lista. Estas operações são ilegais de a lista estiver vazia.</p> <pre>while (list1.size() > 0) { ... list1.pop_front(); }</pre>
remove	<pre>void remove (const T& value);</pre> <p>Apaga, da lista, todos os elementos que são iguais a um valor. O operador de igualdade (==) para o tipo T (de elementos da lista) deve ser definido.</p> <pre>nums.remove(15);</pre>
sort	<pre>void sort();</pre> <p>Ordena os elementos da lista em ordem crescente. O operador de comparação < ("menor que") deve ser definido para o tipo de elemento da lista. Note que o algoritmo de ordenação STL NÃO funciona para listas; é por isso que uma função de classificação deve ser fornecida.</p> <pre>nums.sort();</pre>
reverse	<pre>void reverse();</pre> <p>Inverte a ordem dos elementos de uma lista.</p> <pre>nums.reverse();</pre>

Um exemplo simples de uso de uma lista STL

```
// An example using STL list.
// This program pushes 4 integer values to an STL list.
// It then prints out each of these 4 values before
// deleting them from the list
#include <iostream>
#include <list>           // list class library
using namespace std;

int main()
{
    // Now create a "list" object, specifying its content as "int".
    // The "list" class does not have the same "random access"
    // capability
    // as the "vector" class, but it is possible to add elements at
    // the end of the list and take them off the front.
    list<int> list1;

    // Add some values at the end of the list, which is initially
    // empty.
    // The member function "push back" adds an item at the end of
    // the list.
    int value1 = 10;
    int value2 = -3;
    list1.push back (value1);
    list1.push back (value2);
    list1.push back (5);
    list1.push back (1);

    // Output the list values, by repeatedly getting the item from
    // the "front" of the list, outputting it, and removing it
    // from the front of the list.
    cout << endl << "List values:" << endl;
    // Loop as long as there are still elements in the list.
    while (list1.size() > 0)
    {
        // Get the value of the "front" list item.
        int value = list1.front();
        // Output the value.
        cout << value << endl;
        // Remove the item from the front of the list ("pop front"
        // member function).
        list1.pop front();
    }

    return(0);
}
```

Note que para usar a lista, deve ser feito o include à biblioteca STL:

```
#include <list>
```

No exemplo a seguir, demonstra-se o uso de algumas das funções do *container* de listas STL. Este programa cria uma lista de inteiros aleatórios e, em seguida, coloca a lista em forma ordenada:

```
// Create a random integer list and sort the list
#include <iostream>
#include <list>
using namespace std;

int main()
{
    list<int> lst;
    int i;

    //create a list of random integers
    for(i = 0; i < 10; i++)
        lst.push back(rand());

    cout << "Original list: " << endl;
    // create an iterator object and make it point to the
    // beginning of the list
    list<int>::iterator p = lst.begin();
    while(p != lst.end()){
        cout << *p << " ";
        p++;
    }

    cout << endl << endl;

    // sort the list
    lst.sort();

    cout << "Sorted contents:\n";
    p = lst.begin();
    while(p != lst.end()){
        cout << *p << " ";
        p++;
    }
    cout << endl << endl;
    return (0);
}
```

Aqui está um exemplo da saída do programa:

```
Original list:
41 18467 6334 26500 19169 15724 11478 29358 26962 24464

Sorted contents:
41 6334 11478 15724 18467 19169 24464 26500 26962 29358
```