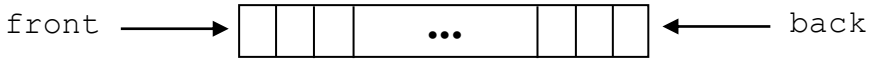
EEL670 - Linguagens de Programação
Departamento de Engenharia Eletrônica e de Computação - DEL-Poli/UFRJ
Prof.: Miguel Elias Mitre Campista
== Laboratório 11

---

1) Lista de exercícios: Resolva os exercícios abaixo como se pede.

a) Escreva uma classe template `ListaRestrita` que restringe as possíveis operações em um array a apenas as de inserção e remoção de elementos nas extremidades da estrutura. Isso quer dizer que a classe template `ListaRestrita` oferece apenas os métodos `push_front` e `push_back` para inserções no início e no final da estrutura, respectivamente, e os métodos `pop_front` e `pop_back` para remoção de elementos no início e no final da estrutura, respectivamente, como visto na figura abaixo.



A classe armazena os elementos em uma array privado alocado dinamicamente no construtor. Lembre-se de implementar um destrutor para liberação da memória. Uma cópia dos elementos no início e no fim da estrutura é retornada a partir dos métodos `front` e `back`.

O índice do elemento no fim da estrutura é armazenado no atributo privado `top`, assim como o tamanho máximo da estrutura, que é armazenado no atributo privado `maxsize`. Note que quando a lista estiver vazia, `top == -1` é verdadeiro; e quando ela estiver cheia, `top == (maxsize - 1)` é verdadeiro.

Faça uma função principal que contemple todos os métodos da classe template `ListaRestrita`.

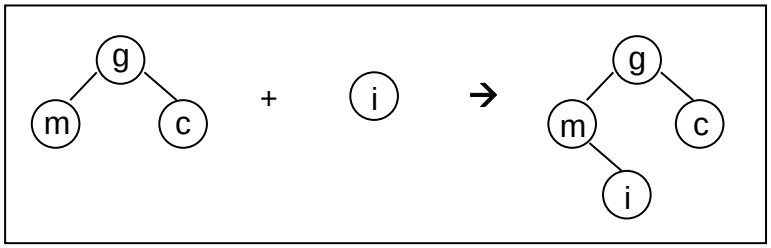
b) Continuando a Questão 1.a, reescreva o programa anterior usando herança da classe `vector`. Isso irá dispensar a necessidade de implementação do array. Além do mais, note que os métodos `front` e `back` podem ter problemas, caso a estrutura esteja vazia. Dispare uma exceção do tipo `EstruturaVaziaException` derivada da classe `exception`, caso isso ocorra. A exceção deverá ser tratada na função principal.

Faça uma função principal que contemple todos os métodos da classe template `ListaRestrita`.

2) Programa para entrega dia 23/12/2022: A entrega do programa será através do Google Classroom e consiste da devolução de todos os arquivos referentes ao código-fonte, um Makefile e um arquivo README que documente a utilização do programa. Todos os arquivos serão avaliados e devem ser entregues em um único arquivo compactado (zip ou rar).

Escreva um programa que implemente uma classe `Cadastro` para gerenciamento de pacientes hospitalares. A classe `Cadastro` reproduz parte das operações existentes na classe `Agenda`, como implementada na lista de exercícios do Laboratório 10. Porém, ao invés de utilizar uma estrutura `vector` para armazenamento, a classe `Cadastro` utiliza uma estrutura do tipo árvore binária, implementada como uma classe template, como em "`template <class T> class Arvore`". A classe `Cadastro` deve oferecer as seguintes operações:

- **Inserção**: A inserção é realizada através do método `insere` que emprega o operador +, como em "`arvore = arvore + paciente`", implementado na classe template `Arvore`. Cada elemento inserido deve ser armazenado na árvore privada. Para isso, o elemento é posicionado à esquerda do nó raiz atual da árvore, caso o nome do paciente seja maior em ordem alfabética que o do nó raiz. Caso o nome seja menor em ordem alfabética, este deve ser posicionado à direita do nó raiz atual. Este procedimento pode ser realizado de forma recursiva. A figura abaixo ilustra a inserção do caractere "i" na árvore binária de caracteres. Note que a primeira comparação foi entre a raiz atual "g" e o "i". A comparação demonstrou que o caractere "i" é maior em ordem alfabética que o "g" e, como tal, deve ser posicionado à esquerda de "g". Em seguida, a mesma comparação é realizada com o "m", raiz atual da sub-árvore à esquerda, onde se conclui que "i" é menor que "m". Portanto, o caractere "i" deve ser posicionado à direita do "m". Como não há mais nenhuma sub-árvore à direita do "m", o caractere "i" é finalmente inserido. Não há preocupações em manter a árvore balanceada.



  A inserção deve retornar a árvore resultante ou a própria árvore usada para inserção, sem alterações, caso a operação de inserção falhe. Uma falha ocorre caso um paciente com o mesmo nome já exista no cadastro.

- **Busca**: A busca deve ser realizada através do método `busca` que utiliza o operador `[]`, como em "`arvore["nome"]`". A busca retorna um ponteiro para o elemento encontrado ou `NULL`, caso contrário. A busca é realizada a partir do nome do paciente, uma vez que não é permitido inserir dois pacientes com o mesmo nome.

- **Impressão**: A impressão de todos os elementos da árvore pode ser obtida através de método `imprime`. Todos os pacientes da árvore e seus atributos, por sua vez, devem ser impressos a partir do operador `<<`, como em "`cout << arvore`". Para ajudar, é interessante implementar também o operador `<<` sobrecarregado para objetos da classe `Paciente`, como em "`cout << paciente`".

Note que a operação de busca pode retornar `NULL`. Trate esse caso como uma exceção na função principal. Caso essa exceção ocorra, o programa deve imprimir uma mensagem específica usando o método `what()` de uma classe especializada derivada da classe `exception`.

Ainda, assim como na lista de exercícios realizada, o cadastro deve ser capaz de ser utilizado para armazenamento de pacientes diversos, com diferentes tipos de atributos privados específicos. Dessa forma, o cadastro tem que ser genérico o suficiente para ser utilizado para qualquer tipo de paciente. Crie, portanto, uma classe `Paciente` base e classes derivadas específicas. **Utilize o conceito de polimorfismo.**

Observação 1: *Crie um menu que permita a execução de todas as ações por intermédio da interação com o usuário. Alternativamente, é permitido que as opções sejam passadas para o executável através de* `argc` *e* `argv`*.*

Observação 2: *Não é necessário inserir tipos diferentes de pacientes no mesmo cadastro. Dessa forma, ao criar um cadastro, este poderá ser usado para armazenar pacientes de uma única classe, derivada ou não da classe* `Paciente`*. É importante, porém, reforçar que o cadastro deve funcionar independentemente da classe, derivada ou não da classe* `Paciente`*, escolhida. Para tanto, o polimorfismo deve ser utilizado.*

== Respostas da Lista de Exercícios

1)

a)

```
/*****************************************************************************/
/***************************** Programa Principal ****************************/
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "lista-restrita.h"

using namespace std;

/* Programa do Laboratório 10:
   Programa de uma class template Lista
   Autor: Miguel Campista */


int main() {
    ListaRestrita <int> lista (5);
    srand (time (0));

    /* Inserções ao final da estrutura */
    int numero = rand () % 10;
    while (lista.push_back (numero)) {
           cout << "Inseri: " << numero << endl;
           numero = rand () % 10;
    }

    cout << endl;

    cout << "Elemento no início: " << lista.front()
         << "\nElemento no final: " << lista.back()
         << endl;

    cout << endl;

    while (lista.pop_back (numero)) {
           cout << "Removi: " << numero << endl;
    }

    cout << "\n*** Agora ao contrário\n" << endl;

    /* Inserções no início da estrutura */
    numero = rand () % 10;
    while (lista.push_front (numero)) {
           cout << "Inseri: " << numero << endl;
           numero = rand () % 10;
    }

    cout << endl;

    cout << "Elemento no início: " << lista.front()
         << "\nElemento no final: " << lista.back()
         << endl;

    cout << endl;

    while (lista.pop_front (numero)) {
           cout << "Removi: " << numero << endl;
    }

    return 0;
}

/*****************************************************************************/
/************************ Arquivo lista-restrita.h ***************************/
#include <iostream>

using namespace std;
```

```
#ifndef LISTA_RESTRITA_H
#define LISTA_RESTRITA_H

template <class T>
class ListaRestrita {
    public:
            ListaRestrita (int sz):
                    maxsize (sz), top (-1), ptr (new T [sz]) {}

            ~ListaRestrita () { delete [] ptr; }

            bool push_back (T v) {
                    if (!estaCheia ()) {
                            ptr [++top] = v;
                            return true;
                    }
                    return false;
            }

            bool push_front (T v) {
                    if (!estaCheia ()) {
                            int idx = ++top;
                            while (idx > 0)
                                    ptr [idx] = ptr [--idx];
                            ptr [0] = v;
                            return true;
                    }
                    return false;
            }

            bool pop_back (T& v) {
                    if (!estaVazia ()) {
                            v = ptr [top--];
                            return true;
                    }
                    return false;
            }

            bool pop_front (T& v) {
                    if (!estaVazia ()) {
                            int idx = 1;
                            v = ptr [0];
                            while (idx <= top) {
                                    ptr [idx - 1] = ptr [idx++];
                    }
                    top--;
                    return true;
                    }
                    return false;
            }

            T front () { return ptr [0]; }
            T back () { return ptr [top]; }

            bool estaCheia () { return top == maxsize - 1; }
            bool estaVazia () { return top == - 1; }

    private:
            T *ptr;
            int top, maxsize;
};

#endif

/*****************************************************************************/
```

b)

```
/*****************************************************************************/
/***************************** Programa Principal ****************************/
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "lista-restrita.h"
```

```
#include "estrutura-vazia-exception.h"

using namespace std;

/* Programa do Laboratório 10:
   Programa de uma class template Lista
   Autor: Miguel Campista */


int main() {
    ListaRestrita <int> lista (5);
    srand (time (0));

    /* Inserções ao final da estrutura */
    int numero = rand () % 10;
    while (lista.push_back (numero)) {
           cout << "Inseri: " << numero << endl;
           numero = rand () % 10;
    }

    cout << endl;

    try {
           cout << "Elemento no início: " << lista.front()
                   << "\nElemento no final: " << lista.back()
                   << endl;
    } catch (EstruturaVaziaException &except) {
           cout << except.what() << endl;
    }

    cout << endl;

    while (lista.pop_back (numero)) {
           cout << "Removi: " << numero << endl;
    }

    cout << "\n*** Agora ao contrário\n" << endl;

    /* Inserções no início da estrutura */
    numero = rand () % 10;
    while (lista.push_front (numero)) {
           cout << "Inseri: " << numero << endl;
           numero = rand () % 10;
    }

    cout << endl;

    try {
           cout << "Elemento no início: " << lista.front()
                   << "\nElemento no final: " << lista.back()
                   << endl;
    } catch (EstruturaVaziaException &except) {
           cout << except.what() << endl;
    }

    cout << endl;

    while (lista.pop_front (numero)) {
           cout << "Removi: " << numero << endl;
    }

    /* Só para provocar a exceção... */
    try {
           cout << "Elemento no início: " << lista.front()
                   << "\nElemento no final: " << lista.back()
                   << endl;
    } catch (EstruturaVaziaException &except) {
           cout << "\n***" << except.what() << endl;
    }

    return 0;
}

/*************************************************************************/
/************************ Arquivo lista-restrita.h ************************/
#include <iostream>
#include <vector>
```

```
#include "estrutura-vazia-exception.h"

using namespace std;

#ifndef LISTA_RESTRITA_H
#define LISTA_RESTRITA_H

template <class T>
class ListaRestrita : public vector <T> {
    public:
            ListaRestrita (int sz):
                    maxsize (sz), top (-1), vector <T> (sz) {}

            bool push_back (T v) {
                    if (!estaCheia ()) {
                            vector <T>::at (++top) = v;
                            return true;
                    }
                    return false;
            }

            bool push_front (T v) {
                    if (!estaCheia ()) {
                            int idx = ++top;
                            while (idx > 0)
                                    vector <T>::at (idx) = vector <T>::at (--idx);
                            vector <T>::at (0) = v;
                            return true;
                    }
                    return false;
            }

            bool pop_back (T& v) {
                    if (!estaVazia ()) {
                            v = vector <T>::at (top--);
                            return true;
                    }
                    return false;
            }

            bool pop_front (T& v) {
                    if (!estaVazia ()) {
                            int idx = 1;
                            v = vector <T>::at (0);
                            while (idx <= top) {
                                    vector <T>::at (idx - 1) = vector <T>::at (idx++);
                    }
                    top--;
                    return true;
                    }
                    return false;
            }

            T front () {
                    if (estaVazia())
                            throw EstruturaVaziaException ();
                    return vector <T>::at (0);
            }

            T back () {
                    if (estaVazia())
                            throw EstruturaVaziaException ();
                    return vector <T>::at (top);
            }

            bool estaCheia () { return top == vector <T>::size() - 1; }
            bool estaVazia () { return top == - 1; }

    private:
            int top, maxsize;
};

#endif

/*************************************************************************/
/****************** Arquivo estrutura-vazia-exception.h ******************/
```

```
#include <iostream>
#include <stdexcept>

using namespace std;

#ifndef ESTRUTURA_VAZIA_EXCEPTION_H
#define ESTRUTURA_VAZIA_EXCEPTION_H

class EstruturaVaziaException : public exception {
    public:
            virtual const char * what () const throw ();
};

#endif

/*************************************************************************/
/****************** Arquivo estrutura-vazia-exception.cpp ******************/
#include "estrutura-vazia-exception.h"

const char * EstruturaVaziaException::what () const throw () {
    return "Estrutura Vazia Exception!";
}
/*************************************************************************/
```