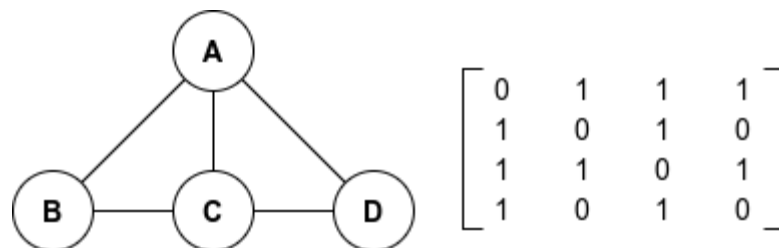


1) **Lista de exercícios:** Resolva os exercícios abaixo como se pede.

- a) Escreva um programa para implementação de um vértice de um grafo. A classe `Vertice` deve ter um rótulo (nome) **constante** como um atributo privado. Além disso, a classe deve oferecer um construtor e método do tipo `get` para retorno do rótulo. A possibilidade de instanciação de objetos **constantes** da classe `Vertice` deve ser prevista. Dessa forma, qualquer objeto, constante ou não, deve ser capaz de chamar um `get` para retorno do rótulo.
- b) Escreva um programa que implemente uma classe Grafo simétrico inspirada em um grafo Erdős-Rényi, $G(n, p)$, onde n é o número de vértices e p é a probabilidade de existência de arestas entre qualquer par de vértices. A implementação, porém, deve prever que o grafo é iniciado sem vértices e que estes são inseridos um a um. Note que sempre que um novo vértice for inserido, as arestas já existentes devem ser preservadas. Isso quer dizer que apenas as arestas que incluam o vértice inserido devem ser adicionadas ao grafo conforme a probabilidade p .

A classe Grafo mantém um vector de objetos da classe `Vertice` e uma matriz de adjacências como atributos privados. A matriz de adjacências é uma matriz de inteiros, implementada a partir de um ponteiro para array de ponteiros (`int **matriz`). Essa matriz possui 1 (um) quando existe uma aresta entre os vértices i e j e 0 (zero), caso contrário. Além do vector e da matriz de adjacências, a classe Grafo possui também um atributo privado do tipo `double` para armazenar a probabilidade p de existência de arestas e o número de arestas.

A figura abaixo ilustra um grafo contendo quatro vértices e a sua matriz de adjacências, onde o vértice A possui índice 0 na matriz, o vértice B possui índice 1 e assim por diante.



A classe `Vertice` é a mesma da Questão 1.a, enquanto a classe `Aresta` não precisa ser criada. A classe `Grafo` oferece um construtor que inicializa os atributos p com argumento passado para o construtor, o número de arestas com zero e o ponteiro para a matriz de adjacências com `NULL`. Além do construtor, a classe oferece um método público para fazer inserção de vértices no grafo e consequente atualização da matriz de adjacências. Para isso, lembre-se de sempre realocar memória dinamicamente para a matriz usando a função `malloc` do C. Além disso, não deixe memória vaziar ao realocar memória. A definição se existe aresta ou não deve obedecer a probabilidade p . A classe `Grafo` oferece ainda um método para impressão na tela da matriz de adjacências e um método para verificar o grau de conectividade da rede. Este último retorna o número de arestas do grafo sobre o número máximo de arestas que este mesmo grafo poderia ter (combinação do número de vértices 2 a 2). Note que arestas do tipo (i, i) não existem.

A classe `Grafo` deve implementar um destrutor e outros métodos privados auxiliares que possam ser julgados necessários.

== Respostas da Lista de Exercícios

1)

a)

```
/* *****
***** Programa Principal *****
#include <iostream>
#include <string>

#include "vertice.h"

/* Programa do Laboratório 6:
   Programa de um Vertice usando método e atributo constantes
   Autor: Miguel Campista */

using namespace std;

int main() {
    Vertice vertice ("A");
    const Vertice constvertice ("B");

    cout << "Vertice: " << vertice.getRotulo() << endl;
    cout << "Const Vertice: " << constvertice.getRotulo() << endl;

    return 0;
}
/* *****
***** Arquivo vertice.h *****
#include <iostream>
#include <string>

using namespace std;

#ifndef VERTICE_H
#define VERTICE_H

class Vertice {
public:
    Vertice (string = "semrotulo");

    string getRotulo ();
    string getRotulo () const;

private:
    const string rotulo;
};
#endif
/* *****
***** Arquivo vertice.cpp *****
#include "vertice.h"

Vertice::Vertice (string s): rotulo(s) {}

string Vertice::getRotulo () {
    return rotulo;
}

string Vertice::getRotulo () const {
    return rotulo;
}
/* *****
```

b)

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

```

#include "vertice.h"
#include "grafo.h"

/* Programa do Laboratório 6:
   Programa de matriz de adjacências de um grafo
   Autor: Miguel Campista */

using namespace std;

int main() {
    Grafo grafo (0.25);
    Vertice v1("L"), v2("I"), v3 ("N"), v4("G");

    grafo.inserereVertice(v1);
    grafo.imprimeMatriz();

    grafo.inserereVertice(v2);
    grafo.imprimeMatriz();

    grafo.inserereVertice(v3);
    grafo.imprimeMatriz();

    grafo.inserereVertice(v4);
    grafo.imprimeMatriz();

    cout << "\nCONECTIVIDADE: " << grafo.getConectividade () << endl;

    return 0;
}
/*****
***** Arquivo vertice.h *****/
#include <iostream>
#include <string>

using namespace std;

#ifndef VERTICE_H
#define VERTICE_H

class Vertice {
public:
    Vertice (string = "semrotulo");

    string getRotulo();
    string getRotulo() const;

private:
    const string rotulo;
};

#endif

/*****
***** Arquivo vertice.cpp *****/

#include "vertice.h"

Vertice::Vertice (string r): rotulo (r) {}

string Vertice::getRotulo() {
    return rotulo;
}

string Vertice::getRotulo () const {
    return rotulo;
}

/*****
***** Arquivo grafo.h *****/

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <iomanip>

#include "vertice.h"

```

```

using namespace std;

#ifndef GRAFO_H
#define GRAFO_H

class Grafo {
public:
    Grafo (double);
    ~Grafo ();

    void insereVertice (Vertice &);

    double getConectividade ();

    void imprimeMatriz ();

private:
    int **matriz;
    vector <Vertice> vertices;
    double probabilidadeAresta;
    int numArestas;

    void zerarMatriz ();
    void popularMatriz ();

    int existeAresta(int, int);
};

#endif

/*****
**** Arquivo grafo.cpp *****/

#include "grafo.h"

Grafo::Grafo (double p) : matriz (NULL), probabilidadeAresta (p), numArestas (0) {
    srand (time (0));
}

Grafo::~Grafo () {
    zerarMatriz();
}

void Grafo::insereVertice (Vertice &v) {
    vertices.push_back(v);
    popularMatriz ();
}

void Grafo::zerarMatriz () {
    for (int i = 0; i < vertices.size() - 1; i++) {
        free (matriz[i]);
    }
    free (matriz);
}

void Grafo::popularMatriz () {
    numArestas = 0;

    int **matrizNova = (int **)malloc(vertices.size()*sizeof(int *));

    for (int i = 0; i < vertices.size(); i++)
        matrizNova [i] = (int *)malloc (vertices.size()*sizeof(int));

    for (int i = 0; i < vertices.size(); i++) {
        for (int j = 0; j < vertices.size(); j++) {
            unsigned ultimoVertice = vertices.size() - 1;
            if ((i == ultimoVertice) || (j == ultimoVertice))
                matrizNova[i][j] = existeAresta(i, j);
            else
                matrizNova [i][j] = matriz [i][j];

            numArestas += matrizNova [i][j];
        }
    }
}

```

```

        if (vertices.size() > 1) zerarMatriz ();

        matriz = matrizNova;
    }

    int Grafo::existeAresta(int i, int j) {
        if ((i == j) || ((static_cast<double>(rand () % 101))/100 > probabilidadeAresta))
            return 0;
        else return 1;
    }

    double Grafo::getConectividade () {
        return static_cast<double>(numArestas)/(vertices.size()*(vertices.size() - 1));
    }

    void Grafo::imprimeMatriz () {
        cout << setw(3) << " ";
        for (int i = 0; i < vertices.size(); i++) {
            cout << left << setw (3) << vertices.at(i).getRotulo();
        }
        cout << endl;
        for (int i = 0; i < vertices.size(); i++) {
            cout << setw(3) << vertices.at(i).getRotulo();
            for (int j = 0; j < vertices.size(); j++) {
                cout << left << setw (3) << matriz[i][j];
            }
            cout << endl;
        }
    }
}

/*****

```