

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

- a) Escreva um programa de um Relógio. Para isso, crie uma classe chamada `Relogio` que possui cinco atributos privados: `time_t hora`, `struct tm *infoHora`, `int segundos`, `int minutos` e `int horas`. Os tipos `time_t` e `struct tm` estão definidos no arquivo `time.h` que deve ser incluído.

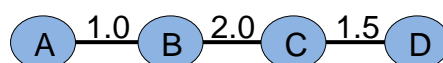
A classe `Relogio` ainda possui dois construtores sobrecarregados que inicializam os atributos `horas`, `minutos` e `segundos` da classe de formas diferentes. Enquanto um construtor é padrão e inicializa os atributos com o horário atual; o outro requer a passagem de três parâmetros inteiros para inicialização dos atributos `horas`, `minutos` e `segundos`. Implemente também o método sobrecarregado `resetHora`, que assim com o construtor, possui uma versão sem passagem de argumentos para reinicialização dos atributos `horas`, `minutos` e `segundos` com o horário atual e outra com passagem de parâmetros inteiros para reinicialização dos mesmos atributos.

O método `resetHora` sem argumentos usa funções definidas no arquivo `time.h`, e atributos da classe `Relogio`, como pode ser visto no trecho de código a seguir:

```
time (&hora);  
infoHora = localtime (&hora);
```

Saiba que a estrutura `infoHora` possui os campos `tm_sec`, `tm_min` e `tm_hour` que armazenam inteiros contendo, respectivamente o valor dos segundos, dos minutos e das horas atuais. Além dos métodos construtores e dos métodos `resetHora`, a classe `Relogio` possui ainda métodos do tipo “set” e “get” para os atributos `segundos`, `minutos` e `horas`, que devem realizar validação dos valores recebidos. Por exemplo, o atributo `hora` não pode ter valor negativo. Por fim, a classe `Relogio` implementa um método chamado `mostreHorario` que exibe na tela o horário do relógio.

- b) Escreva um programa para definição de caminhos em grafos simétricos. Um caminho é definido como uma sequência de arestas (ou enlaces) que ligam uma sequência de vértices distintos. A figura abaixo ilustra um caminho entre os nós A e D, composto por 4 vértice e 3 arestas. Note que cada aresta tem um peso, denotada pelo número em Real acima de cada linha. Dessa forma, pode-se representar uma aresta como sendo uma tupla contendo os dois vértices mais o peso. Por exemplo, a primeira aresta do caminho abaixo é composta por um primeiro vértice A, um segundo vértice B e peso igual a 1.0. Já o caminho completo entre A e D é composto por três arestas (A, B, 1.0), (B, C, 2.0) e (C, D, 1.5), com custo igual a 4.5, que é a soma de todos os pesos das arestas individuais.



Tendo em vista a definição de caminhos dada, escreva um programa que implemente um caminho como um objeto da classe `Caminho`. Tal classe possui como atributo

privado um `vector` de **ponteiros** para objetos da classe `Aresta`. A classe `Caminho` deve oferecer publicamente um método de inserção de arestas, um para cálculo do custo do caminho e outro para impressão na tela da sequência de vértices do caminho. Note que o método de inserção deve inserir arestas sempre ao final do caminho atual. Para isso, é necessário verificar se o primeiro vértice da aresta a ser inserida coincide com o último vértice do caminho. Caso não coincida, a aresta não pode ser inserida.

A classe `Aresta` possui dois ponteiros para objetos da classe `Vertice` como atributos privados, além do peso. Os vértices são objetos da classe `Vertice` inicializados através do construtor da classe. A classe `Vertice` ainda implementa três métodos `get`, um para cada atributo. O construtor prevê o valor 1.0 como peso padrão das arestas. Por fim, a classe `Vertice` possui apenas um atributo do tipo `string` para armazenar o rótulo do vértice, um construtor que inicializa o rótulo e um método do tipo `get` para recuperar o rótulo.

Considere que para inserir uma aresta no caminho, primeiro é preciso criar dois vértices, para depois criar a aresta. Somente depois da aresta criada, há a inserção dessa mesma aresta no caminho.

- 2) **Programa para entrega dia 27/08/2021:** A entrega do programa será através do Google Classroom e consiste da devolução de um arquivo zip ou rar contendo todos os arquivos referentes ao código-fonte, um Makefile e um arquivo README que documente a utilização do programa. Todos os arquivos serão avaliados.

Escreva um programa que leia linhas de texto de um arquivo `*.txt` de entrada (pode igualmente ser um csv) contendo as arestas e respectivos pesos de um grafo simétrico. Este arquivo pode estar organizado como se segue:

```
A B 2,2
B C 1,0
C D 3,1
A C 1,2
```

Note que quatro arestas e quatro vértices são definidos no arquivo acima, sendo a primeira aresta composta pelos vértices A e B com peso 2,2. **No programa a ser entregue, considere minimamente 20 vértices e 20 arestas. Note que nem todos os vértices precisam ter arestas associadas e que alguns vértices podem participar de mais arestas que outros.**

O programa deve implementar um grafo simétrico como uma classe. Da mesma forma, as arestas e os vértices também são implementados como classes. A classe `Grafo` possui um método de inserção de arestas, enquanto a classe `aresta` possui um método construtor que define os seus vértices de origem e de destino. Os vértices e arestas podem ser criados conforme o arquivo de entrada é lido. Após a criação completa do grafo, o programa deve oferecer as seguintes opções através de um menu:

1. Imprimir na tela o número de vértices e de enlaces no grafo.
2. Imprimir na tela a lista de todos os vértices existentes no grafo.
3. Exibir na tela a sequência de arestas atravessadas e o custo total do caminho com o menor custo entre dois vértices escolhidos pelo usuário. Utilize o algoritmo de Dijkstra (https://pt.wikipedia.org/wiki/Algoritmo_de_Dijkstra) para o cálculo do caminho de menor custo. Caso haja empate, ou seja, mais de um caminho de menor custo exista entre o par de vértices solicitado, o programa pode escolher apenas um desses caminhos para exibir a sequência de arestas atravessadas.

4. Imprimir na tela o diâmetro do grafo. O programa deve apresentar todas as arestas pertencentes ao menor caminho com o maior custo existente, considerando todos os pares de vértices. Assim como na opção 3, em caso haja empate, o programa pode escolher um dos caminhos encontrados para exibir a sequência de arestas atravessadas.
5. Imprimir na tela o vértice com a maior centralidade de grau (vértice com o maior número de arestas).
6. O vértice com a maior centralidade de intermediação (vértice que faz parte da maior quantidade de caminhos de menor custo considerando todos os pares de vértices do grafo).

Observação 1: Ao invés de letras no arquivo de entrada, use nomes de colegas. Ao invés de usar um peso aleatório, use o número de disciplinas cursadas em conjunto. Tente dar significado ao arquivo de entrada.

Observação 2: Ao invés de um menu, os alunos podem optar por passar os argumentos necessários usando `argc` e `argv`.

Observação 3: Pesquisem na Internet o problema da centralidade em grafos. Uma referência inicial pode ser a wikipedia (<https://pt.wikipedia.org/wiki/Centralidade>).

== Respostas da Lista de Exercícios

1)

a)

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

#include "relogio.h"

/* Programa do Laboratório 5:
   Programa de um relógio digital
   Autor: Miguel Campista */

using namespace std;

int main() {
    Relogio pontual; // Inicializado com horário atual
    Relogio atrasado (8, 10, 30); // Inicializado com horário qualquer

    // Mostra os horários ajustados por cada construtor sobrecarregado
    cout << "pontual: "; pontual.mostraHorario ();
    cout << "atrasado: "; atrasado.mostraHorario ();

    // Acerta o relógio atrasado, mas desconfigura o relógio pontual
    pontual.resetHora(1, 2, 59);
    atrasado.resetHora();

    // Mostra os horários ajustados pelo reset
    cout << endl;
    cout << "pontual: "; pontual.mostraHorario ();
    cout << "atrasado: "; atrasado.mostraHorario ();

    return 0;
}
/* *****
***** Arquivo relógio.h *****
#include <iostream>
#include <iomanip>
#include <string>
#include <time.h>

using namespace std;

#ifndef RELOGIO_H
#define RELOGIO_H

class Relogio {
public:
    Relogio ();
    Relogio (int, int, int);

    void resetHora ();
    void resetHora (int, int, int);

    void setSegundos (int);
    void setMinutos (int);
    void setHoras (int);

    int getSegundos ();
    int getMinutos ();
    int getHoras ();

    void mostraHorario ();

private:
    time_t hora;
    struct tm *infoHora;
    int segundos, minutos, horas;
};
```

```

#endif

/*****
/***** Arquivo relógio.cpp *****/

#include "relogio.h"

Relógio::Relógio () {
    resetHora ();
}

Relógio::Relógio (int h, int m, int s) {
    resetHora (h, m, s);
}

void Relógio::resetHora () {
    time (&hora);
    infoHora = localtime (&hora);

    setHoras (infoHora->tm_hour);
    setMinutos (infoHora->tm_min);
    setSegundos (infoHora->tm_sec);
}

void Relógio::resetHora (int h, int m, int s) {
    setHoras (h); setMinutos (m); setSegundos (s);
}

void Relógio::setSegundos (int s) {
    segundos = ((s < 0) || (s > 59)) ? 0 : s;
}
void Relógio::setMinutos (int m) {
    minutos = ((m < 0) || (m > 59)) ? 0 : m;
}
void Relógio::setHoras (int h) {
    horas = ((h < 0) || (h > 23)) ? 0 : h;
}

int Relógio::getSegundos () {
    return segundos;
}
int Relógio::getMinutos () {
    return minutos;
}
int Relógio::getHoras () {
    return horas;
}

void Relógio::mostreHorario () {
    cout << setfill ('0') << setw (2) << getHoras ()
        << ":" << setw(2) << getMinutos ()
        << ":" << setw(2) << getSegundos () << endl;
}
/*****/

```

b)

```

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

#include "vertice.h"
#include "aresta.h"
#include "caminho.h"

/* Programa do Laboratório 5:
   Programa de definição de um caminho
   Autor: Miguel Campista */

using namespace std;

int main() {
    Vertice v1 ("A"), v2 ("B"), v3 ("C"), v4 ("D"), v5 ("E");
    Aresta a1 (&v1, &v2), a2 (&v2, &v3, 2.5), a3 (&v3, &v4, 1.5), aerr (&v5, &v3);
}

```

```

    Caminho caminho;

    caminho.inserirEnlace(&a1);
    caminho.inserirEnlace(&a2);
    caminho.inserirEnlace(&a3);
    // Tenta inserir um enlace que não dá continuidade ao caminho
    caminho.inserirEnlace(&aerr);

    // Imprime o caminho e o peso correspondente
    cout << endl;
    caminho.imprimeCaminho();

    return 0;
}

/*****
/***** Arquivo vertice.h *****/
#include <iostream>
#include <vector>

using namespace std;

#ifndef VERTICE_H
#define VERTICE_H

class Vertice {
public:
    Vertice (string);

    string getRotulo ();

private:
    string rotulo;
};

#endif

/*****
/***** Arquivo vertice.cpp *****/
#include "vertice.h"

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

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

/*****
/***** Arquivo enlace.h *****/
#include <iostream>

#include "vertice.h"

using namespace std;

#ifndef ARESTA_H
#define ARESTA_H

class Aresta {
public:
    Aresta (Vertice *, Vertice *, double = 1.0);

    Vertice *getPrimeiroVertice ();
    Vertice *getSegundoVertice ();

    double getPeso ();

private:
    Vertice *primeiroVertice, *segundoVertice;
    double peso;
};

#endif

```

```

/*****
**** Arquivo enlace.cpp ****
*****/
#include "aresta.h"

Aresta::Aresta (Vertice *prim, Vertice *seg, double p) {
    primeiroVertice = prim; segundoVertice = seg; peso = p;
}
Vertice *Aresta::getPrimeiroVertice () {
    return primeiroVertice;
}
Vertice *Aresta::getSegundoVertice () {
    return segundoVertice;
}
double Aresta::getPeso () {
    return peso;
}

/*****
**** Arquivo caminho.h ****
*****/
#include <iostream>
#include <vector>

#include "aresta.h"

using namespace std;

#ifndef CAMINHO_H
#define CAMINHO_H

class Caminho {
public:
    void insereEnlace (Aresta *);

    void imprimeCaminho ();

    double calculaDistancia ();

private:
    vector <Aresta *> v;
};

#endif

/*****
**** Arquivo caminho.cpp ****
*****/
#include "caminho.h"

void Caminho::insereEnlace (Aresta *a) {
    if (!v.size()) {
        v.push_back (a);
        cout << "Aresta (" << a->getPrimeiroVertice()->getRotulo()
                << ", " << a->getSegundoVertice()->getRotulo()
                << ") inserida no caminho..." << endl;
    } else {
        if (a->getPrimeiroVertice() == v.at(v.size() - 1)->getSegundoVertice()) {
            v.push_back (a);
            cout << "Aresta (" << a->getPrimeiroVertice()->getRotulo()
                    << ", " << a->getSegundoVertice()->getRotulo()
                    << ") inserida no caminho..." << endl;
        } else
            cout << "Aresta (" << a->getPrimeiroVertice()->getRotulo()
                    << ", " << a->getSegundoVertice()->getRotulo()
                    << ") NÃO pode ser inserida no caminho..." << endl;
    }
}

void Caminho::imprimeCaminho () {
    cout << "Caminho: ";

    for (unsigned i = 0; i < v.size(); i++)
        cout << v.at(i)->getPrimeiroVertice ()->getRotulo () << " -- ";

    cout << v.at(v.size() - 1)->getSegundoVertice ()->getRotulo () << endl;
    cout << "Distância: " << calculaDistancia () << endl;
}

```

```
double Caminho::calculaDistancia () {
    double distancia = 0;

    for (unsigned i = 0; i < v.size(); i++)
        distancia += v.at(i)->getPeso ();

    return distancia;
}

/*****/
```