EEL670 - Linguagens de Programação
Departamento de Engenharia Eletrônica e de Computação - DEL-Poli/UFRJ
Prof.: Miguel Elias Mitre Campista
== Laboratório 4

---

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

a) Escreva um programa para armazenar contatos em uma agenda. Para isso, programe a classe `Contato` que possui como atributos privados o nome, a profissão e a idade do contato. A classe `Contato` também oferece um construtor para inicialização de todos os seus atributos privados e métodos do tipo "get" e "set" para cada um deles.

Programe também a classe `Agenda`. Essa classe oferece um objeto da classe `vector` como atributo privado para armazenamento de objetos da classe `Contato`. A classe `Agenda` implementa adicionalmente três atributos privados, um para definir o tamanho máximo do `vector`, outro para definir o tamanho máximo dos nomes dos contatos e um último para lidar com persistência dos contatos. Para isso, implemente como atributo privado um objeto da classe `fstream`.

A inserção de contatos na agenda deve verificar o tamanho do nome, que pode ter no máximo 10 caracteres. Caso o nome seja maior que isso, uma mensagem deve ser exibida ao usuário e o nome deve ser truncado para caber no limite máximo definido. Para truncar os nomes, utilize o método `substr(.)` da classe `string`. Além da verificação do tamanho do nome, antes da inserção de um contato na agenda, é necessário verificar se o tamanho máximo do `vector` já foi atingido (agenda está cheia) e se o contato já existe. Logo, um contato só pode ser inserido na agenda caso nenhum outro com o mesmo nome exista e se a agenda não estiver cheia. Utilize os métodos `push_back(.)` e `size()` da classe `vector` e `compare(.)` da classe `string` na implementação do método de inserção. O primeiro deve ser usado para realizar inserção de contatos no `vector`, o segundo para verificar o número de contatos já inseridos e o último para saber se um nome já existe na agenda.

Para remoção dos contatos, utilize o método `erase(.)` da classe `vector`. Para isso, faça:

```
v.erase(v.begin() + idx);
```

onde `v` é `vector` de contatos e `idx` é o índice inteiro do contato.

Por fim, para implementar persistência, leia os contatos de um arquivo localizado no mesmo diretório do executável ao invocar o construtor da classe `Agenda` e substitua todos os contatos imediatamente antes de terminar o programa no arquivo. Os contatos podem ter seus atributos organizados no arquivo usando espaços ou vírgulas como separadores.

b) Escreva um programa que formate a exibição dos atributos de objetos da classe `Cadastro`. Essa classe possui um construtor que inicializa três atributos privados: `nome`, `endereço` e `telefone`. Além do construtor, a classe `Cadastro` oferece métodos do tipo "get" para cada um dos atributos privados. A formatação é feita através do emprego de uma função global chamada `formataCadastro` que recebe uma referência a um objeto da classe `Cadastro` e um ponteiro para a função que realizará a formatação. O protótipo da função global é o seguinte:

```
void formataCadastro(Cadastro &, void (*)(Cadastro &));
```

Note que diferentes funções podem ser usadas para formatação dos atributos da classe `Cadastro`, sem que a função `formataCadastro` precise ser alterada.

Programe os arquivos `cadastro.h` e `cadastro.cpp` para programação da classe `Cadastro`, o arquivo `global.h` para programação da função global `formataCadastro` e o arquivo `main.cpp` para programação da função principal do programa e das funções cujos ponteiros serão passados como parâmetro da `formataCadastro`.

2) **Programa para entrega dia 17/05/2024:** 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 explique brevemente a compilação e a utilização do programa. Todos os arquivos serão avaliados. Esta atividade é individual.

   O programa deverá apresentar dados referentes ao desempenho de times do futebol brasileiro. Para isso, um menu contendo as seguintes opções deve ser disponibilizado:

   1. Exibir a evolução da média dos gols realizados e sofridos de **três times nos últimos anos** em campeonatos nacionais/estaduais (por exemplo, Brasileirão, Copa do Brasil, Cariocão) individualmente e no total de todos os campeonatos disputados (soma dos campeonatos disputados por um dado time), usando a estratégia de média móvel dos últimos `N` anos. A média móvel ($\bar{I}$) deverá ser calculada como se segue:

$$\bar{I} = \frac{1}{N}\sum_{i=0}^{N-1} m_{a-i},$$

      onde m_a é o número de gols de um dado time no ano atual, m_{a-1} é o número de gols de um dado time no ano anterior ao atual, m_{a-2} é o número de gols de um dado time no ano anterior ao ano m_{a-1} e assim por diante. Calcule a média móvel para no mínimo os últimos três anos (ou seja, usando $N=3$) de estatísticas de gols tanto efetuados quanto sofridos para cada um dos times (campeonatos individuais e total). Assumir que o programa tem acesso a dados referentes aos gols em todos os campeonatos que os times participaram nos últimos $N+1$ anos e que N é no máximo igual a 7. Assumir também que os times disputaram ao menos dois campeonatos por ano;

   2. Exibir de forma agrupada os times que apresentaram melhoria de desempenho em relação aos gols efetuados e também em relação aos gols sofridos no último ano para cada um dos campeonatos disputados. Uma equipe teve melhoria de desempenho segundo a razão entre a média móvel calculada no ano atual ($\bar{I}$ calculada no item 1) e a média móvel calculada no ano anterior (para calcular a média móvel do ano anterior faça: $m_a = m_{a-1}$). Utilize como limiar de definição o valor de 10%, sendo que qualquer um dos times com aumento de gols efetuados superior a 10% ou com redução de 10% no número de gols sofridos teve melhora no desempenho. Mostre também os times com piora de desempenho, aqueles com redução no número de gols efetuados em pelo menos 10% ou aumento no número de gols sofridos no último ano de pelo menos 10% em comparação ao ano anterior. Por fim, mostre os times que estão estáveis, ou seja, que não atendem aos dois critérios anteriores.

   3. Repetir o item 2, sendo que a melhora ou piora de desempenho, ou estabilidade dos times devem considerar todos os campeonatos disputados no último ano em relação ao ano anterior;

   4. Exibir o time com maior saldo de gols (número de gols efetuados menos o número de gols sofridos) em cada um dos campeonatos durante todo o intervalo de tempo considerado;

5. Mostrar qual dos times teve a maior evolução no último ano em relação aos gols efetuados e gols sofridos. Use os resultados calculados para o item 2.

O programa deve prever a implementação de uma classe `Time` e uma classe `Liga`, na qual a classe `Liga` é composta por objetos da classe `Time`. Cada objeto da classe `Time` deve representar um time de futebol brasileiro e, como tal, deve gerenciar individualmente os seus gols efetuados e sofridos nos campeonatos durante o intervalo de tempo escolhido. Note que os dados da série histórica de gols podem ser arbitrários, ou seja, não necessariamente precisam representar a realidade. Os dados da série podem ser inicializados com os objetos da classe `Time`, sem necessidade de inserção ou remoção de dados mensais através de opção do menu ou leitura de arquivo. Considere que todos os times disputaram os mesmos campeonatos durante o intervalo de tempo escolhido.

Dica: Verifique a possibilidade do uso de um array ou de um objeto da classe `vector` para armazenamento dos objetos da classe `Time` pela classe `Liga` e de uma estrutura tridimensional em que uma dimensão é o campeonato, outra é o número de gols efetuados/sofridos e outra é o ano correspondente para armazenamento dos dados da série histórica pelos times.

== Resposta da Lista de Exercícios

1)

a)

```
/*****************************************************************************/
/*************************** Programa Principal ******************************/

#include <iostream>

#include "agenda.h"

/* Programa do Laboratório 4:
   Mais uma agenda...
   Autor: Miguel Campista */

using namespace std;

int main() {
    Agenda agenda;

    agenda.insereContato("ana", "estudante", 20);
    agenda.insereContato("aristoteles", "estudante", 21);
    // A agenda estará cheia aqui da primeira vez que o programa for executado
    agenda.insereContato("nao entra", "professor", 30);

    agenda.mostraTodos ();

    // Teste de uma remoção
    agenda.removeContato("miguel");

    // Vou tentar inserir um contato que já exista...
    agenda.insereContato ("aristoteles", "estudante", 21);

    // Vou inserir um contato diferente porque agora tem espaço
    agenda.insereContato("fatima", "jornalista", 50);

    agenda.mostraTodos ();

    // Vou editar a idade do Aristoles e a profissao dele...
    agenda.editaIdadeContato("aristoteles", 1000);
    agenda.editaProfissaoContato("aristoteles", "cientista");

    agenda.mostraTodos ();

    /* Os dados serão escritos no arquivo.
    Por enquanto, este método precisará ser
    invocado explicitamente... */
    agenda.escreveArquivo();

    return 0;
}
/*****************************************************************************/
/****************************** Arquivo contato.h ****************************/

#include <iostream>
#include <string>

using namespace std;

class Contato {
    public:
            Contato (string, string, int);

            void setNome (string);
            void setProfissao (string);
            void setIdade (int);

            string getNome ();
            string getProfissao ();
            int getIdade ();
```

```
    private:
            string nome, profissao;
            int idade;
};
/*****************************************************************************/
/****************************** Arquivo contato.cpp **************************/

#include "contato.h"

Contato::Contato (string n, string p, int i) {
    setNome (n); setProfissao (p); setIdade (i);
}

void Contato::setNome (string n) {
    nome = n;
}

void Contato::setProfissao (string p) {
    profissao = p;
}

void Contato::setIdade (int i) {
    idade = i;
}

string Contato::getNome () {
    return nome;
}

string Contato::getProfissao () {
    return profissao;
}

int Contato::getIdade () {
    return idade;
}
/*****************************************************************************/
/****************************** Arquivo agenda.h *****************************/

#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <fstream>

#include "contato.h"

using namespace std;

class Agenda {
    public:
            Agenda (int = 3);

            void insereContato (string, string, int);
            void removeContato (string);

            void editaIdadeContato (string, int);
            void editaProfissaoContato (string, string);

            bool existeContato (string);

            void mostraTodos ();

            void lerArquivo ();
            void escreveArquivo ();

    private:
            vector <Contato> v;
            unsigned tamMaxAgenda;
            int tamMaxNome;
            fstream file;
            string nomeArquivo;

            string verificaNome (string);
};
/*****************************************************************************/
```

```
/***************************** Arquivo agenda.cpp *************************/
#include "agenda.h"

Agenda::Agenda (int t) {
    tamMaxAgenda = t;
    tamMaxNome = 10;
    nomeArquivo = "arquivocontatos.txt";

    lerArquivo ();
}

void Agenda::insereContato (string n, string p, int i) {
    static int contaTentativas = 1;

    if (v.size() < tamMaxAgenda) {
            if (existeContato (n)) {
                    cout << "[CONTATO EXISTENTE] " << contaTentativas
                            << "a tentativa de insercao (" << n
                            << ") NAO foi bem sucedida...\n" << endl;
            } else {
                    Contato contato (verificaNome (n), p, i);

                    v.push_back (contato);

                    cout << "[CONTATO INSERIDO] " << contaTentativas
                            << "a tentativa de insercao (" << n
                            << ") foi bem sucedida...\n" << endl;
            }
    } else {
            cout << "[AGENDA CHEIA] " << contaTentativas
                    << "a tentativa de insercao (" << n
                    << ") NAO foi bem sucedida...\n" << endl;
    }
    contaTentativas++;
}

void Agenda::removeContato (string n) {
    for (unsigned i = 0; i < v.size(); i++) {
            /* Usa o nome como critério de busca e,
            em seguida usa o método da classe vector
            para remover o contato */
            if (!v.at(i).getNome ().compare(n.substr (0, tamMaxNome))) {
                    v.erase (v.begin() + i);
                    cout << "[CONTATO REMOVIDO] " << n << " removido!" << endl;
            }
    }
}

bool Agenda::existeContato (string n) {
    for (unsigned i = 0; i < v.size(); i++) {
            /* Usa o nome como critério de busca e,
            em seguida usa o método da classe vector
            para verificar se o contato existe */
            if (!v.at(i).getNome ().compare(n.substr (0, tamMaxNome)))
                    return true;
    }

    return false;
}


void Agenda::editaIdadeContato (string n, int i_nova) {
    for (unsigned i = 0; i < v.size(); i++) {
            /* Usa o nome como critério de busca e,
            em seguida usa o método da classe vector
            para editar o contato */
            if (!v.at(i).getNome ().compare(n.substr (0, tamMaxNome))) {
                    v.at (i).setIdade (i_nova);
                    cout << "[CONTATO EDITADO] idade do contato " << n << " editado!"
<< endl;
            }
    }
}

void Agenda::editaProfissaoContato (string n, string p_nova) {
    for (unsigned i = 0; i < v.size(); i++) {
            /* Usa o nome como critério de busca e,
```

```
                em seguida usa o método da classe vector
                para editar o contato */
                if (!v.at(i).getNome ().compare(n.substr (0, tamMaxNome))) {
                        v.at (i).setProfissao (p_nova);
                        cout << "[CONTATO EDITADO] profissao do contato " << n << "
editado!" << endl;
                }
        }
    }

    void Agenda::mostraTodos () {
        cout << endl;
        cout << left << setw(15) << "Nome:"
                                << setw(15) << "Profissao:"
                                << setw(5) << "Idade:" << endl;

        for (unsigned i = 0; i < v.size(); i++)
                cout << setw(15) << v.at(i).getNome ()
                        << setw(15) << v.at(i).getProfissao ()
                        << setw(5) << v.at(i).getIdade () << endl;

        cout << endl;
    }

    string Agenda::verificaNome (string n) {
        if (n.length() > tamMaxNome) {
                cout << "[CUIDADO] Nome muito comprido!" << endl;
                cout << "Nome truncado a partir do " << tamMaxNome
                     << "o caracter \"" << n [tamMaxNome]
                        << "\": " << n.substr(0, tamMaxNome)  << endl;
        }

        return n.substr(0, tamMaxNome);
    }

    void Agenda::lerArquivo () {
        string n, p;
        int i;

        file.open(nomeArquivo, fstream::in);

        if (!file.is_open()) {
                cout << "Arquivo nao existe." << endl;
                return;
        }

        while (file.good()) {
                file >> n >> p >> i;
                insereContato (n, p, i);
        }

        file.close();
    }

    void Agenda::escreveArquivo () {
        file.open(nomeArquivo, fstream::out);

        for (unsigned i = 0; i < v.size(); i++)
                file << v.at(i).getNome ()
                        << " " << v.at(i).getProfissao ()
                        << " " << v.at(i).getIdade () << endl;

        file.close();
    }
        /*****************************************************************************/
```

b)

```
    /*****************************************************************************/
    /************************** Programa Principal ***************************/

    #include <iostream>
    #include <locale>
    #include <string>

    #include "cadastro.h"
```

```
#include "global.h"

/* Programa do Laboratório 4:
   Experimentando ponteiros para funções...
   Autor: Miguel Campista */

using namespace std;

void formataMaiusculas (Cadastro &c) {
    for (unsigned i = 0; i < c.getNome().length(); i++)
            cout << static_cast<char>(toupper(c.getNome()[i]));

    cout << endl;

    for (unsigned i = 0; i < c.getEnd().length(); i++)
            cout << static_cast<char>(toupper(c.getEnd()[i]));

    cout << endl;

    for (unsigned i = 0; i < c.getTel().length(); i++)
            cout << static_cast<char>(toupper(c.getTel()[i]));

    cout << endl;
}

void formataCSV (Cadastro &c) {
    cout << c.getNome() << ',' << c.getEnd() << ',' << c.getTel() << endl;
}

int main() {
    Cadastro cadastro ("Aluno", "Av. Brigadeiro Trompowski", "2122223333");

    formataCadastro (cadastro, formataMaiusculas);

    cout << endl;

    formataCadastro (cadastro, formataCSV);

    return 0;
}
/*****************************************************************************/
/****************************** Arquivo global.h *****************************/

void formataCadastro (Cadastro &c, void (*formata)(Cadastro &c)) {
    (*formata) (c);
}
/*****************************************************************************/
/***************************** Arquivo cadastro.h ****************************/

#include <iostream>
#include <string>

using namespace std;

class Cadastro {
    public:
            Cadastro (string, string, string);

            string getNome ();
            string getEnd ();
            string getTel ();

    private:
            string nome, end, tel;
};
/*****************************************************************************/
/***************************** Arquivo cadastro.cpp **************************/

#include "cadastro.h"

Cadastro::Cadastro (string n, string e, string t) {
    nome = n; end = e; tel = t;
}

string Cadastro::getNome () {
    return nome;
}
```

```
string Cadastro::getEnd () {
    return end;
}
string Cadastro::getTel () {
    return tel;
}
/*****************************************************************************/
```