EEL670 - Linguagens de Programação
Departamento de Engenharia Eletrônica e de Computação - DEL-Poli/UFRJ
Prof.: Miguel Elias Mitre Campista
== Laboratório 9

---

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

a) Escreva uma classe `Agenda` para armazenar contatos. Cada contato é um objeto da classe `Contato` que possui atributos `nome` e `telefone`. Os contatos são armazenados em um `vector` de ponteiros para objeto da classe `Contato`. A classe `Agenda` deve oferecer um método para inserir contatos e outro para remoção. O método de inserção deve inserir o contato no `vector`, enquanto o de remoção deve tirar um contato específico do mesmo `vector`. A classe `Agenda` ainda deve implementar um construtor de cópia, o operador `+` sobrecarregado para concatenar agendas, o operador `-` sobrecarregado para retornar os contatos diferentes entre duas agendas (como subtração de conjuntos) e o operador `<<` sobrecarregado para impressão na tela do conteúdo da agenda.

A classe `Contato` deve oferecer um método para inicialização dos seus atributos (pode ser o próprio construtor) e métodos do tipo "get" para os atributos.

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

b) Continuando a Questão 1.a, reescreva o programa anterior para armazenar contatos de uma turma de colegas. Para tal, crie uma classe derivada da classe `Contato` chamada classe `Colega`. A classe `Colega` deve possuir os atributos privados `classe` e `turma`. Reescreva a classe `Agenda` para armazenar elementos da nova classe criada.

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

c) Reescreva ainda o programa anterior, armazenando os contatos da turma no `vector` da classe `Agenda`. Utilize, porém, um `vector` para ponteiros para objetos da classe `Contato`, como definidos na Questão 1.a.

Utilize conceitos de polimorfismo.

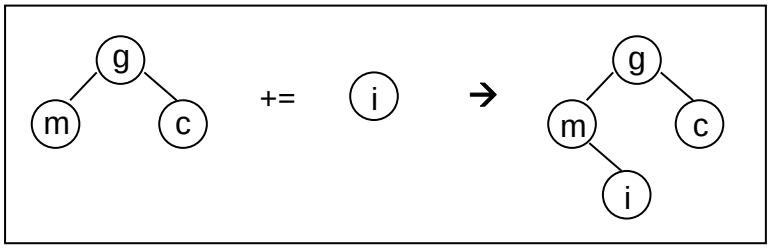
d) Reescreva mais uma vez o programa anterior, implementando a classe `Agenda` como uma classe derivada da classe `vector` especializada para `Contato *`. Dessa forma, os métodos "insere" e "apaga" deixam de ser necessários, já que os métodos `push_back` (ou `insert`) e `erase` da classe `vector` são herdados e, consequentemente, podem ser usados respectivamente na função principal. Note que os métodos `push_back` (ou `insert`) e `erase` poderão ser usados, mas algumas alterações serão necessárias tanto na função principal quanto na classe `Agenda`.

2) Programa para entrega dia 18/02/2022: 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 atual, o Laboratório 9. 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 += 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 um ponteiro para o nó inserido ou `NULL`, 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, tanto a operação de inserção quanto a de busca podem retornar `NULL`. Trate esses casos como uma exceção na função principal. Caso essas exceções ocorram, o programa deve imprimir uma mensagem específica usando o método `what()` 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.

PACIENTE EXAME - com o exame que vai fazer ou quer fazer ou ja fez
PACIENTE PLANO - com o plano de saúde

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 "agenda.h"
#include "contato.h"

using namespace std;

/* Programa do Laboratório 9:
   Programa de mais uma Agenda com polimorfismo
   Autor: Miguel Campista */

int main() {
    Agenda agenda;

    Contato c1 ("Miguel", "111222");
    agenda.insereContato (c1);
    Contato c2 ("Joao", "222333");
    agenda.insereContato (c2);

    Agenda agendaCopia (agenda);

    cout << "Depois das inserções:" << endl;
    cout << agenda << endl;

    cout << "E a cópia, como ficou?" << endl;
    cout << agendaCopia << endl;

    agenda.apagaContato ("Miguel");
    cout << "Depois de apagar o contato Miguel:" << endl;
    cout << agenda << endl;

    cout << "Qual a diferença da cópia para a original?" << endl;
    Agenda agendaDif = (agendaCopia - agenda);
    cout << agendaDif << endl;

    cout << "Vamos concatenar tudo de novo em uma nova:" << endl;
    Agenda agendaNova = (agenda + agendaDif);
    cout << agendaNova << endl;

    return 0;
}

/*****************************************************************************/
/***************************** Arquivo contato.h *****************************/
#include <iostream>
#include <string>

using namespace std;

#ifndef CONTATO_H
#define CONTATO_H

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

            string getNome ();
            string getTelefone ();

            void print ();

    private:
            string nome, telefone;
};

#endif
```

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

Contato::Contato (string n, string t): nome (n), telefone (t) {}

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

string Contato::getTelefone () { return telefone; }

void Contato::print () {
    cout << getNome () << ": " << getTelefone () << endl;
}

/*****************************************************************************/
/***************************** Arquivo agenda.h ******************************/
#include <iostream>
#include <string>
#include <vector>

#include "contato.h"

using namespace std;

#ifndef AGENDA_H
#define AGENDA_H

class Agenda {
    friend ostream &operator<< (ostream &, Agenda &);

    public:
            Agenda ();

            Agenda (const Agenda &);

            /* Optei inserir contatos diretamente porque assim a agenda fica
            mais genérica. Ou seja, ela continua valendo, independente de
            quais atributos a classe Contato possui.*/
            void insereContato (Contato &);

            /* Optei apagar contatos baseando a busca pelo nome. Estou
            assumindo que todo contato tem pelo menos o método getNome. */
            void apagaContato (string);

            Agenda operator- (const Agenda &);

            Agenda operator+ (const Agenda &);

    private:
            vector <Contato *> vContatos;
};

#endif

/*****************************************************************************/
/***************************** Arquivo agenda.cpp ****************************/

#include "agenda.h"

ostream &operator<< (ostream &out, Agenda &a) {
    for (int i = 0; i < a.vContatos.size (); i++) {
            ((a.vContatos).at (i))->print ();
    }
    return out;
}

Agenda::Agenda () {}

Agenda::Agenda (const Agenda &a) {
    vContatos = a.vContatos;
}

void Agenda::insereContato (Contato &c) {
    vContatos.push_back (&c);
}
```

```
void Agenda::apagaContato (string n) {
    vector <Contato *>::iterator it = vContatos.begin ();

    for (; it != vContatos.end (); it++) {
            if (!((*it)->getNome ()).compare (n)) {
                    vContatos.erase (it);
                    break;
            }
    }
}

Agenda Agenda::operator- (const Agenda &a) {
    Agenda agenda;
    bool achou;

    for (int i = 0; i < this->vContatos.size (); i++) {
            achou = false;

            for (int j = 0; j < a.vContatos.size (); j++) {
                    string nome = (this->vContatos).at (i)->getNome ();

                    if (!nome.compare (a.vContatos.at (j)->getNome ())) {
                            achou = true; break;
                    }
            }

            if (!achou) agenda.insereContato (*(this->vContatos.at (i)));
    }

    return agenda;
}

Agenda Agenda::operator+ (const Agenda &a) {
    Agenda agenda;

    for (int i = 0; i < this->vContatos.size (); i++) {
            Contato *c = this->vContatos.at (i);
            agenda.insereContato (*c);
    }

    bool achou;

    for (int i = 0; i < a.vContatos.size (); i++) {
            achou = false;

            for (int j = 0; j < this->vContatos.size (); j++) {
                    string nome = (a.vContatos).at (i)->getNome ();

                    if (!nome.compare (this->vContatos.at (j)->getNome ())) {
                            achou = true; break;
                    }
            }

            if (!achou) agenda.insereContato (*a.vContatos.at (i));
    }

    return agenda;

}

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

b)

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

#include "agendacolega.h"
#include "colega.h"

using namespace std;

/* Programa do Laboratório 9:
   Programa de mais uma Agenda com polimorfismo
   Autor: Miguel Campista */
```

```
int main() {
    Agenda agenda;

    Colega c1 ("Miguel", "111222", "classeA", "turmaA");
    agenda.insereContato (c1);
    Colega c2 ("Joao", "222333", "classeB", "turmaB");
    agenda.insereContato (c2);

    Agenda agendaCopia (agenda);

    cout << "Depois das inserções:" << endl;
    cout << agenda << endl;

    cout << "E a cópia, como ficou?" << endl;
    cout << agendaCopia << endl;

    agenda.apagaContato ("Miguel");
    cout << "Depois de apagar o contato Miguel:" << endl;
    cout << agenda << endl;

    cout << "Qual a diferença da cópia para a original?" << endl;
    Agenda agendaDif = (agendaCopia - agenda);
    cout << agendaDif << endl;

    cout << "Vamos concatenar tudo de novo em uma nova:" << endl;
    Agenda agendaNova = (agenda + agendaDif);
    cout << agendaNova << endl;

    return 0;
}

/*****************************************************************************/
/***************************** Arquivo contato.h *****************************/

#include <iostream>
#include <string>

using namespace std;

#ifndef CONTATO_H
#define CONTATO_H

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

            string getNome ();
            string getTelefone ();

            void print ();

    private:
            string nome, telefone;
};

#endif

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

Contato::Contato (string n, string t): nome (n), telefone (t) {}

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

string Contato::getTelefone () { return telefone; }

void Contato::print () {
    cout << getNome () << ": " << getTelefone () << endl;
}

/*****************************************************************************/
/***************************** Arquivo colega.h ******************************/
#include <iostream>
#include <string>
```

```
#include "contato.h"

using namespace std;

#ifndef COLEGA_H
#define COLEGA_H

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

            string getClasse ();
            string getTurma ();

            void print ();

    private:
            string classe, turma;
};

#endif
/*****************************************************************************/
/*************************** Arquivo colega.cpp ******************************/
#include "colega.h"

Colega::Colega (string n, string t, string c, string tu):
            Contato (n, t), classe (c), turma (tu) {}

string Colega::getClasse () { return classe; }

string Colega::getTurma () { return turma; }

void Colega::print () {
    Contato::print ();

    cout << "(" << getClasse ()
             << ", " << getTurma ()
             << ")" << endl;
}

/*****************************************************************************/
/************************** Arquivo agendacolega.h ***************************/
#include <iostream>
#include <string>

#include <vector>
#include "colega.h"

using namespace std;

#ifndef AGENDACOLEGA_H
#define AGENDACOLEGA_H

class Agenda {
    friend ostream &operator<< (ostream &, Agenda &);

    public:
            Agenda ();

            Agenda (const Agenda &);

            /* Optei inserir contatos diretamente porque assim a agenda fica
            mais genérica. Ou seja, ela continua valendo, independente de
            quais atributos a classe Contato possui.*/
            void insereContato (Colega &);

            /* Optei apagar contatos baseando a busca pelo nome. Estou
            assumindo que todo contato tem pelo menos o método getNome. */
            void apagaContato (string);

            Agenda operator- (const Agenda &);

            Agenda operator+ (const Agenda &);

    private:
            vector <Colega *> vContatos;
};
```

```
#endif
/*****************************************************************************/
/************************ Arquivo agendacolega.cpp ***************************/
#include "agendacolega.h"

ostream &operator<< (ostream &out, Agenda &a) {
    for (int i = 0; i < a.vContatos.size (); i++) {
            ((a.vContatos).at (i))->print ();
    }

    return out;
}

Agenda::Agenda () {}

Agenda::Agenda (const Agenda &a) {
    vContatos = a.vContatos;
}

void Agenda::insereContato (Colega &c) {
    vContatos.push_back (&c);
}

void Agenda::apagaContato (string n) {
    vector <Colega *>::iterator it = vContatos.begin ();

    for (; it != vContatos.end (); it++) {
            if (!((*it)->getNome ()).compare (n)) {
                    vContatos.erase (it);
                    break;
            }
    }
}

Agenda Agenda::operator- (const Agenda &a) {
    Agenda agenda;
    bool achou;
    for (int i = 0; i < this->vContatos.size (); i++) {
            achou = false;

            for (int j = 0; j < a.vContatos.size (); j++) {
                    string nome = (this->vContatos).at (i)->getNome ();

                    if (!nome.compare (a.vContatos.at (j)->getNome ())) {
                            achou = true; break;
                    }
            }

            if (!achou) agenda.insereContato (*(this->vContatos.at (i)));
    }

    return agenda;
}

Agenda Agenda::operator+ (const Agenda &a) {
    Agenda agenda;

    for (int i = 0; i < this->vContatos.size (); i++) {
            Colega *c = this->vContatos.at (i);
            agenda.insereContato (*c);
    }

    bool achou;

    for (int i = 0; i < a.vContatos.size (); i++) {
            achou = false;

            for (int j = 0; j < this->vContatos.size (); j++) {
                    string nome = (a.vContatos).at (i)->getNome ();

                    if (!nome.compare (this->vContatos.at (j)->getNome ())) {
                    achou = true; break;
                    }
            }

            if (!achou) agenda.insereContato (*a.vContatos.at (i));
```

```
    }

    return agenda;
}
/*****************************************************************************/
```

c)

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

#include "agenda.h"
#include "colegapoli.h"

using namespace std;

/* Programa do Laboratório 9:
   Programa de mais uma Agenda com polimorfismo
   Autor: Miguel Campista */

int main() {
    Agenda agenda;

    Colega c1 ("Miguel", "111222", "classeA", "turmaA");
    agenda.insereContato (c1);
    Colega c2 ("Joao", "222333", "classeB", "turmaB");
    agenda.insereContato (c2);

    Agenda agendaCopia (agenda);

    cout << "Depois das inserções:" << endl;
    cout << agenda << endl;

    cout << "E a cópia, como ficou?" << endl;
    cout << agendaCopia << endl;

    agenda.apagaContato ("Miguel");
    cout << "Depois de apagar o contato Miguel:" << endl;
    cout << agenda << endl;

    cout << "Qual a diferença da cópia para a original?" << endl;
    Agenda agendaDif = (agendaCopia - agenda);
    cout << agendaDif << endl;

    cout << "Vamos concatenar tudo de novo em uma nova:" << endl;
    Agenda agendaNova = (agenda + agendaDif);
    cout << agendaNova << endl;

    return 0;
}

/*****************************************************************************/
/************************** Arquivo contatopoli.h ****************************/
#include <iostream>
#include <string>

using namespace std;

#ifndef CONTATOPOLI_H
#define CONTATOPOLI_H

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

            string getNome ();
            string getTelefone ();

            virtual void print ();

    private:
            string nome, telefone;
};

#endif
```

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

Contato::Contato (string n, string t): nome (n), telefone (t) {}

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

string Contato::getTelefone () { return telefone; }

void Contato::print () {
    cout << getNome () << ": " << getTelefone () << endl;
}

/*****************************************************************************/
/*************************** Arquivo colegapoli.h ****************************/
#include <iostream>
#include <string>

#include "contatopoli.h"

using namespace std;

#ifndef COLEGAPOLI_H
#define COLEGAPOLI_H

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

            string getClasse ();
            string getTurma ();

            virtual void print ();

    private:
            string classe, turma;
};

#endif

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

Colega::Colega (string n, string t, string c, string tu):
            Contato (n, t), classe (c), turma (tu) {}

string Colega::getClasse () { return classe; }

string Colega::getTurma () { return turma; }

void Colega::print () {
    Contato::print ();

    cout << "(" << getClasse ()
             << ", " << getTurma ()
             << ")" << endl;
}

/*****************************************************************************/
/****************************** Arquivo agenda.h *****************************/
#include <iostream>
#include <string>
#include <vector>

#include "contatopoli.h"

using namespace std;

#ifndef AGENDA_H
#define AGENDA_H

class Agenda {
    friend ostream &operator<< (ostream &, Agenda &);
```

```
    public:
            Agenda ();

            Agenda (const Agenda &);

            /* Optei inserir contatos diretamente porque assim a agenda fica
            mais genérica. Ou seja, ela continua valendo, independente de
            quais atributos a classe Contato possui.*/
            void insereContato (Contato &);

            /* Optei apagar contatos baseando a busca pelo nome. Estou
            assumindo que todo contato tem pelo menos o método getNome. */
            void apagaContato (string);

            Agenda operator- (const Agenda &);

            Agenda operator+ (const Agenda &);

    private:
            vector <Contato *> vContatos;
};

#endif

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

ostream &operator<< (ostream &out, Agenda &a) {
    for (int i = 0; i < a.vContatos.size (); i++) {
            ((a.vContatos).at (i))->print ();
    }
    return out;
}

Agenda::Agenda () {}

Agenda::Agenda (const Agenda &a) {
    vContatos = a.vContatos;
}

void Agenda::insereContato (Contato &c) {
    vContatos.push_back (&c);
}

void Agenda::apagaContato (string n) {
    vector <Contato *>::iterator it = vContatos.begin ();

    for (; it != vContatos.end (); it++) {
            if (!((*it)->getNome ()).compare (n)) {
                    vContatos.erase (it);
                    break;
            }
    }
}

Agenda Agenda::operator- (const Agenda &a) {
    Agenda agenda;
    bool achou;

    for (int i = 0; i < this->vContatos.size (); i++) {
            achou = false;

            for (int j = 0; j < a.vContatos.size (); j++) {
                    string nome = (this->vContatos).at (i)->getNome ();

                    if (!nome.compare (a.vContatos.at (j)->getNome ())) {
                            achou = true; break;
                    }
            }

            if (!achou) agenda.insereContato (*(this->vContatos.at (i)));
    }

    return agenda;
}
```

```
Agenda Agenda::operator+ (const Agenda &a) {
    Agenda agenda;

    for (int i = 0; i < this->vContatos.size (); i++) {
            Contato *c = this->vContatos.at (i);
            agenda.insereContato (*c);
    }

    bool achou;

    for (int i = 0; i < a.vContatos.size (); i++) {
            achou = false;

            for (int j = 0; j < this->vContatos.size (); j++) {
                    string nome = (a.vContatos).at (i)->getNome ();

                    if (!nome.compare (this->vContatos.at (j)->getNome ())) {
                            achou = true; break;
                    }
            }

            if (!achou) agenda.insereContato (*a.vContatos.at (i));
    }

    return agenda;

}
/*****************************************************************************/
```

d)

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

#include "agenda.h"
#include "colegapoli.h"

using namespace std;

/* Programa do Laboratório 9:
   Programa de mais uma Agenda com polimorfismo
   Autor: Miguel Campista */

int main() {
    Agenda agenda;

    Colega c1 ("Miguel", "111222", "classeA", "turmaA");
    agenda.push_back (&c1);
    Colega c2 ("Joao", "222333", "classeB", "turmaB");
    agenda.push_back (&c2);

    Agenda agendaCopia (agenda);

    cout << "Depois das inserções:" << endl;
    cout << agenda << endl;

    cout << "E a cópia, como ficou?" << endl;
    cout << agendaCopia << endl;

    Agenda::iterator it = agenda.begin ();
    for (; it != agenda.end (); it++) {
            if (!((*it)->getNome ()).compare ("Miguel")) {
                    agenda.erase (it);
                    break;
            }
    }
    cout << "Depois de apagar o contato Miguel:" << endl;
    cout << agenda << endl;

    cout << "Qual a diferença da cópia para a original?" << endl;
    Agenda agendaDif = (agendaCopia - agenda);
    cout << agendaDif << endl;

    cout << "Vamos concatenar tudo de novo em uma nova:" << endl;
    Agenda agendaNova = (agenda + agendaDif);
```

```
    cout << agendaNova << endl;

    return 0;
}

/*****************************************************************************/
/*************************** Arquivo contatopoli.h ***************************/
#include <iostream>
#include <string>

using namespace std;

#ifndef CONTATOPOLI_H
#define CONTATOPOLI_H

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

            string getNome ();
            string getTelefone ();

            virtual void print ();

    private:
            string nome, telefone;
};

#endif

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

Contato::Contato (string n, string t): nome (n), telefone (t) {}

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

string Contato::getTelefone () { return telefone; }

void Contato::print () {
    cout << getNome () << ": " << getTelefone () << endl;
}

/*****************************************************************************/
/*************************** Arquivo colegapoli.h ****************************/
#include <iostream>
#include <string>

#include "contatopoli.h"

using namespace std;

#ifndef COLEGAPOLI_H
#define COLEGAPOLI_H

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

            string getClasse ();
            string getTurma ();

            virtual void print ();

    private:
            string classe, turma;
};

#endif

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

Colega::Colega (string n, string t, string c, string tu):
           Contato (n, t), classe (c), turma (tu) {}
```

```
string Colega::getClasse () { return classe; }

string Colega::getTurma () { return turma; }

void Colega::print () {
    Contato::print ();

    cout << "(" << getClasse ()
            << ", " << getTurma ()
            << ")" << endl;
}

/*****************************************************************************/
/************************* Arquivo agenda.h ********************************/
#include <iostream>
#include <string>
#include <vector>

#include "contatopoli.h"

using namespace std;

#ifndef AGENDA_H
#define AGENDA_H

class Agenda : public vector <Contato *> {
    friend ostream &operator<< (ostream &, Agenda &);

    public:
            Agenda ();

            Agenda (const Agenda &);

            Agenda operator- (const Agenda &);

            Agenda operator+ (const Agenda &);
};

#endif

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

ostream &operator<< (ostream &out, Agenda &a) {
    for (int i = 0; i < a.size (); i++) {
            (a.at (i))->print ();
    }

    return out;
}

Agenda::Agenda () {}

Agenda::Agenda (const Agenda &a) {
    for (int i = 0; i < a.size (); i++)
            this->push_back (a.at (i));
}

Agenda Agenda::operator- (const Agenda &a) {
    Agenda agenda;
    bool achou;

    for (int i = 0; i < this->size (); i++) {
            achou = false;

                    for (int j = 0; j < a.size (); j++) {
                            string nome = this->at (i)->getNome ();

                            if (!nome.compare (a.at (j)->getNome ())) {
                                    achou = true; break;
                            }
                    }

            if (!achou) agenda.push_back (this->at (i));
    }
```

```
    return agenda;
}

Agenda Agenda::operator+ (const Agenda &a) {
    Agenda agenda;

    for (int i = 0; i < this->size (); i++) {
            Contato *c = this->at (i);
            agenda.push_back (c);
    }

    bool achou;

    for (int i = 0; i < a.size (); i++) {
            achou = false;

            for (int j = 0; j < this->size (); j++) {
                    string nome = a.at (i)->getNome ();

                    if (!nome.compare (this->at (j)->getNome ())) {
                            achou = true; break;
                    }
            }

            if (!achou) agenda.push_back (a.at (i));
    }

    return agenda;
}
/*****************************************************************************/
```