

== Laboratório 5 ==

1. 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 vetor 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 vetor, enquanto o de remoção deve tirar um contato específico do mesmo vetor. 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`.

2. 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`.

3. Reescreva ainda o programa anterior, armazenando os contatos da turma no vetor da classe `Agenda`. Utilize, porém, um vetor para ponteiros para objetos da classe `Contato`, como definidos na questão 1. Utilize conceitos de polimorfismo.

== Respostas ==

1.

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

#include <iostream>
#include "agenda.h"
#include "contato.h"

using namespace std;

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 agenda.h *****/

#ifndef AGENDA_H
#define AGENDA_H

#include <iostream>
#include <string>
#include <vector>
#include "contato.h"

using namespace std;

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

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

#ifndef CONTATO_H
#define CONTATO_H

#include <iostream>
#include <string>

using namespace std;

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;
}

/*****/

```

2.

```

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

#include <iostream>
#include "agendacolega.h"
#include "colega.h"

using namespace std;

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 agendacolega.h *****/

#ifndef AGENDACOLEGA_H
#define AGENDACOLEGA_H

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

using namespace std;

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 = false;
    for (int i = 0; i < this->vContatos.size (); i++) {
        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 = false;
    for (int i = 0; i < a.vContatos.size (); i++) {
        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.inserereContato (*a.vContatos.at (i));
    }

    return agenda;
}

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

#ifndef CONTATO_H
#define CONTATO_H

#include <iostream>
#include <string>

using namespace std;

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 *****/

#ifndef COLEGA_H
#define COLEGA_H

#include <iostream>
#include <string>
#include "contato.h"

using namespace std;

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;
}

/*****/

```

3.

```

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

#include <iostream>
#include "agenda.h"
#include "colegapoli.h"

using namespace std;

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 agenda.h *****/

#ifndef AGENDA_H
#define AGENDA_H

#include <iostream>
#include <string>

```

```

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

using namespace std;

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:
    /* Mesmo usando vector de Contatos, podemos inserir objetos
    da classe Colega graças ao polimorfismo. A classe Agenda
    utilizada é a mesma da questão 1. */
    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 = false;
    for (int i = 0; i < this->vContatos.size (); i++) {
        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 = false;
    for (int i = 0; i < a.vContatos.size (); i++) {
        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;
}

/*****
*****/
/***** Arquivo contatopoli.h *****/

#ifndef CONTATOPOLI_H
#define CONTATOPOLI_H

#include <iostream>
#include <string>

using namespace std;

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 *****/

#ifndef COLEGAPOLI_H
#define COLEGAPOLI_H

#include <iostream>
#include <string>
#include "contatopoli.h"

using namespace std;

```

```

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 "cologapoli.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;
}

/*****/

```