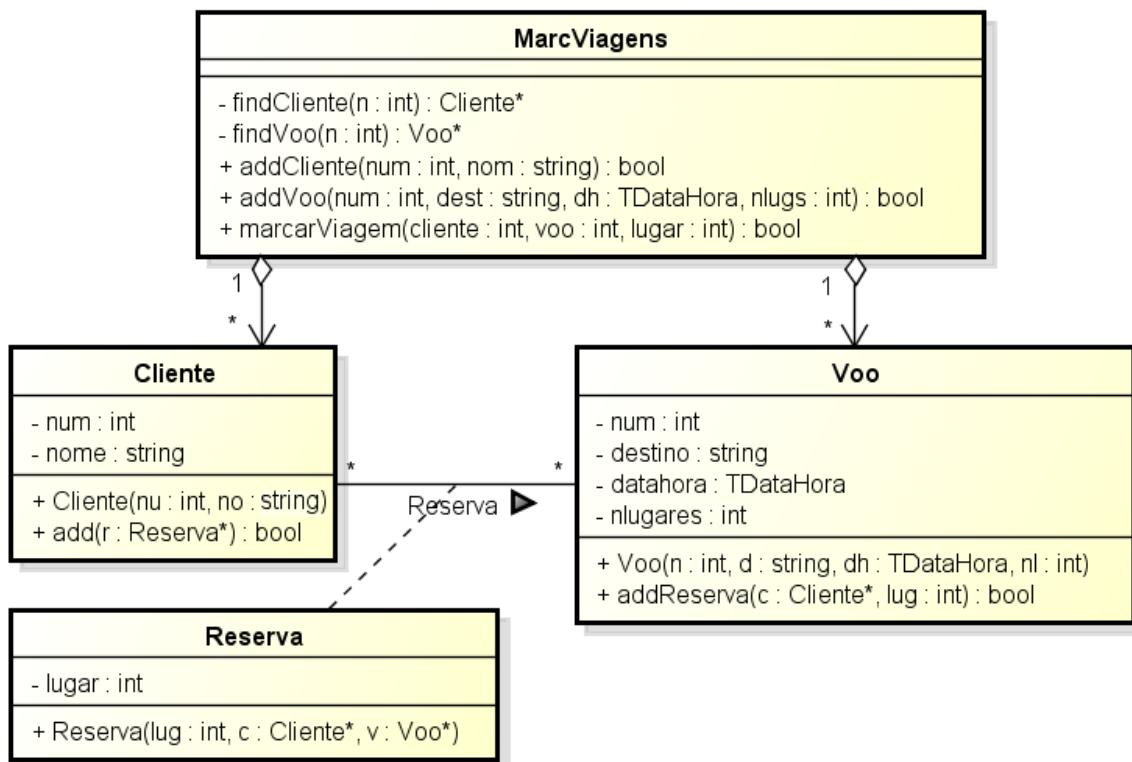


Notas importantes:

1. A duração da prova é de 2 horas.
2. Comece por preencher a zona reservada à sua identificação na folha de exame.

Grupo I
[15 valores]

Uma transportadora aérea necessita de uma pequena aplicação que permita aos seus clientes marcar a sua viagem num dos voos agendados. Nessa marcação deverá ficar reservado o lugar da aeronave que o cliente escolher. Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para o problema descrito.



a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e atributos descritos no diagrama, e tendo ainda em conta todas as seguintes considerações: [9.9 val.]

- Não defina quaisquer outros métodos ou atributos, a não ser, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis;
- A marcação da viagem só deve ser permitida para um lugar da aeronave que se encontre ainda livre;
- Para o tipo data/hora é usada a classe **TDataHora**, da qual se apresenta, no Anexo B deste enunciado, o protótipo de alguns dos seus métodos e operadores mais importantes;

- As coleções deverão ser implementadas com base nos templates de classes Colecao ou ColecaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;
 - Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;
 - Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.
- b) Acrescente ao problema os métodos que permitam apresentar na saída standard os voos que se encontram lotados (mostre apenas o número de cada voo). [2.1 val.]
- c) Acrescente ao problema os métodos que permitam cancelar uma reserva, sendo fornecidos os números do voo e do lugar. [3 val.]

Grupo II
[5 valores]

No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com o devido cuidado, responda às seguintes questões.

- a) Diga quais são os métodos construtores (definidos explicitamente ou não) que estão presentes em cada uma das classes do problema. [0.6 val.]
- b) Implemente o método nota() da classe EpNormal, de forma a devolver o valor da nota da época normal de avaliação, calculada através da média ponderada que leve em conta 30% da nota do miniteste, 20% do trabalho prático e 50% da prova de exame, isto para o caso desta última nota (a do exame) ser maior ou igual a 7 valores. Caso seja menor, então será a nota do exame a ser usada para nota final da época normal. [0.9 val.]
- c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.5 val.]
- d) Qual o resultado que seria visualizado na saída standard se o método print() da classe Epoca não fosse virtual? Apresente apenas a parte da visualização que resultaria diferente em relação ao output da alínea anterior. [0.8 val.]
- e) Diga que consequências é que teria, naquilo que é visualizado, o destrutor da classe Epoca não ser definido como virtual. [0.4 val.]
- f) Por que razão numa coleção híbrida são normalmente guardados os apontadores para os objetos que se pretende colecionar, em vez dos próprios objetos? [0.8 val.]

ANEXO A

```
#include<iostream>
using namespace std;

class Epoca {
    double exame;
public:
    Epoca(double e) {
        exame = e;
        cout << "Inicio de Epoca..." << endl;
    }
    virtual int nota() const { return (int)(exame + 0.5); }
    virtual bool aprovado() const { return nota() >= 10; }
    virtual void print() const {
        cout << (aprovado())?"aprovado com ":"reprovado com ") << nota() << " valores.\n";
    }
    virtual ~Epoca() { cout << "Fim de Epoca: " << endl; }
};

class EpRecurso : public Epoca {
public:
    EpRecurso(double e): Epoca(e){cout << "Inicio da Epoca de Recurso..." << endl;}
    void print() const { cout << "Epoca de Recurso: "; Epoca::print(); }
    ~EpRecurso() { cout << "Fim da Epoca de Recurso" << endl; }
};

class EpNormal: public Epoca{
    double miniteste, trabalho;
    static const double notaMinimaExame;
public:
    EpNormal(double mini, double trab, double exam) : Epoca(exam) {
        miniteste = mini; trabalho = trab;
        cout << "Inicio da Epoca Normal..." << endl;
    }
    int nota()const { //implementar na alinea b)
        ...
    }
    void print() const { cout << "Epoca Normal: "; Epoca::print(); }
    ~EpNormal() { cout << "Fim da Epoca Normal" << endl; }
};

const double EpNormal::notaMinimaExame = 7.0;

void main(){
    EpNormal enormal(10.0, 15.0, 14.0);
    EpRecurso erecurso(11.3);
    cout << "-1-" << endl;
    Epoca &en = enormal, &er = erecurso;
    cout << "-2-" << endl;
    en.print();
    er.print();
    cout << "Classificacao final: ";
    if (er.nota() > en.nota()) cout << er.nota() << endl;
    else cout << en.nota() << endl;
    cout << "-3-" << endl;
};
```

ANEXO B

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);
    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers{
public:
    bool operator()(const T &left, const T &right) const { return (*left < *right); }

};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);
    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Classe TDataHora

```
class TDataHora{
public: // construtor por defeito
    TDataHora();
    // cria o objeto com a data d/m/a e a hora h:min:s
    TDataHora(int d, int m, int a, int h, int min, double s);
    // cria o objeto com a data hora expressa em dt (string com o formato "x/x/xxxx h:m:s")
    TDataHora(const string & dt);
    // atribui ao objeto a data d/m/a e a hora h:min:s
    bool set(int d, int m, int a, int h, int min, double s);
    // atribui ao objeto a data hora expressa em dt (string com o formato "x/x/xxxx h:m:s")
    void set(const string &dt);
    // devolve uma string com a data hora do objeto expressa no formato "x/x/xx h:m:s"
    string toString() const;
    // verifica se a data hora do objeto é anterior a uma outra
    bool operator<(const TDataHora &outra) const;
    // verifica se a data hora do objeto é igual a uma outra
    bool operator==(const TDataHora &outra) const;
    // devolve um objeto do tipo TDataHora com a data e hora correntes retiradas do sistema
    static TDataHora hoje_agora();
};

};
```

Grupo I (15 valores)

- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e... [9.9 val.]

```
Class MarcViagens {  
    Colecao<Cliente> clientes; 3  
    Colecao<Voo> voos;  
  
    Cliente *findCliente(int n) const{ 2  
        Cliente c(n, "");  
        return clientes.find(c);  
    }  
    Voo *findVoo(int n) const{ 2  
        Voo v(n, "", TDataHora(), 0);  
        return voos.find(v);  
    }  
public:  
    bool addCliente(int num, string nom) { 2  
        Cliente c(num, nom);  
        return clientes.insert(c);  
    }  
    bool addVoo(int num, string dest, TDataHora dh, int nlugs) { 2  
        Voo v(num, dest, dh, nlugs);  
        return voos.insert(v);  
    }  
    bool marcarViagem(int cliente, int voo, int lugar) 4  
        Cliente *c = findCliente(cliente);  
        if (c == NULL) return false; //Cliente inexistente!  
        else {  
            Voo *v = findVoo(voo);  
            if (v == NULL) return false; //Voo inexistente!  
            else return v->addReserva(c, lugar);  
        }  
    }  
};  
  
class Reserva {  
    int lugar; 2  
    Cliente *cliente;  
    Voo *voo;  
public:  
    Reserva(int lug, Cliente *c, Voo *v) { 1  
        lugar = lug;  
        cliente = c;  
        voo = v;  
    }  
    bool operator<(const Reserva &outro) const {return lugar < outro.lugar;} 1  
};
```

```

class Cliente{
    int num;
    string nome;
    Colecao<Reserva*> reservas;      2
public:
    Cliente(int nu, string no) { num = nu; nome = no; }      1
    bool add(Reserva *r) { return reservas.insert(r); }      2
    bool operator<(const Cliente &outro) const { return num < outro.num; } 1
};

class Voo{
    int num;          2
    string destino;
    TDataHora datahora;
    int nlugares;
    Colecao<Reserva> reservas;
public:
    Voo(int n, string d, TDataHora dh, int nl){      1
        num = n; destino = d; datahora = dh; nlugares = nl;
    }
    bool addReserva(Cliente *c, int lug) {            4
        if (lug<1 || lug>nlugares) return false;
        Reserva r(lug, c, this);
        if (reservas.insert(r))
            return c->add(reservas.find(r));
        else return false;
    }
    bool operator<(const Voo &outro) const { return num < outro.num; } 1
};

```

b) Acrescente ao problema os métodos que permitam apresentar na saída standard. [2.1 val.]

```

void MarcViagens::mostrarVooLotados() const {           4
    Colecao<Voo>::iterator it;
    for (it = voos.begin(); it != voos.end(); it++)
        if (it->lotado()) cout<<"Voo " << it->getNumero() << " lotado." << endl;
}

bool Voo::lotado() const {                            2
    return reservas.size() == nlugares;
}

int Voo::getNumero() const {                         1
    return num;
}

```

c) Acrescente ao problema os métodos que permitam cancelar uma reserva... [3 val.]

```

bool MarcViagens::cancelarReserva(int voo, int lugar) { 3
    Voo *v = findVoo(voo);
    if (v == NULL) return false; //Voo inexistente!
    else return v->cancelarReserva(lugar);
}

```

```
bool Voo::cancelarReserva(int lugar) {  
    Reserva r(lugar, NULL, NULL);  
    Reserva *pr = reservas.find(r);  
    if (pr == NULL) return false;  
    else {  
        pr->getCliente()->cancelarReserva(pr);  
        reservas.erase(r);  
        return true;  
    }  
}
```

4

```
*Reserva::getCliente()const { return cliente; }
```

1

```
void Cliente::cancelarReserva(Reserva *r) { return reservas.erase(r); }
```

2

Grupo II (5 valores)

- a) Diga quais são os métodos construtores (definidos explicitamente ou não) que estão ... [0.6 val.]
*Nas classes Epoca e EpRecurso: construtor de cópia e construtor de conversão;
Na classe EpNormal: construtor de cópia e o construtor de 3 parâmetros EpNormal(double,double,double).*

- b) Implemente o método nota() da classe EpNormal de forma a devolver o valor da nota... [0.9 val.]

```
int nota()const {
    if (Epoca::nota() < notaMinimaExame) return Epoca::nota();
    else {
        double val = miniteste * 0.3 + trabalho * 0.2 + Epoca::nota()*0.5;
        return (int)(val + 0.5);
    }
}
```

- c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.5 val.]

```
Inicio de Epoca...
Inicio da Epoca Normal...
Inicio de Epoca...
Inicio da Epoca de Recurso...
-1-
-2-
Epoca Normal: aprovado com 13 valores.
Epoca de Recurso: aprovado com 11 valores.
Classificacao final: 13
-3-
Fim da Epoca de Recurso
Fim de Epoca:
Fim da Epoca Normal
Fim de Epoca:
```

- d) Apresente o resultado que será visualizado na saída standard após a execução do programa. [0.8 val.]

```
-2-
Epoca Normal: aprovado com 13 valores.
Epoca de Recurso: aprovado com 11 valores.
Classificacao final: 13
-3-
```

- e) Diga que consequências é que teria, naquilo que é visualizado, o destrutor [0.4 val.]
Nada se alterava.

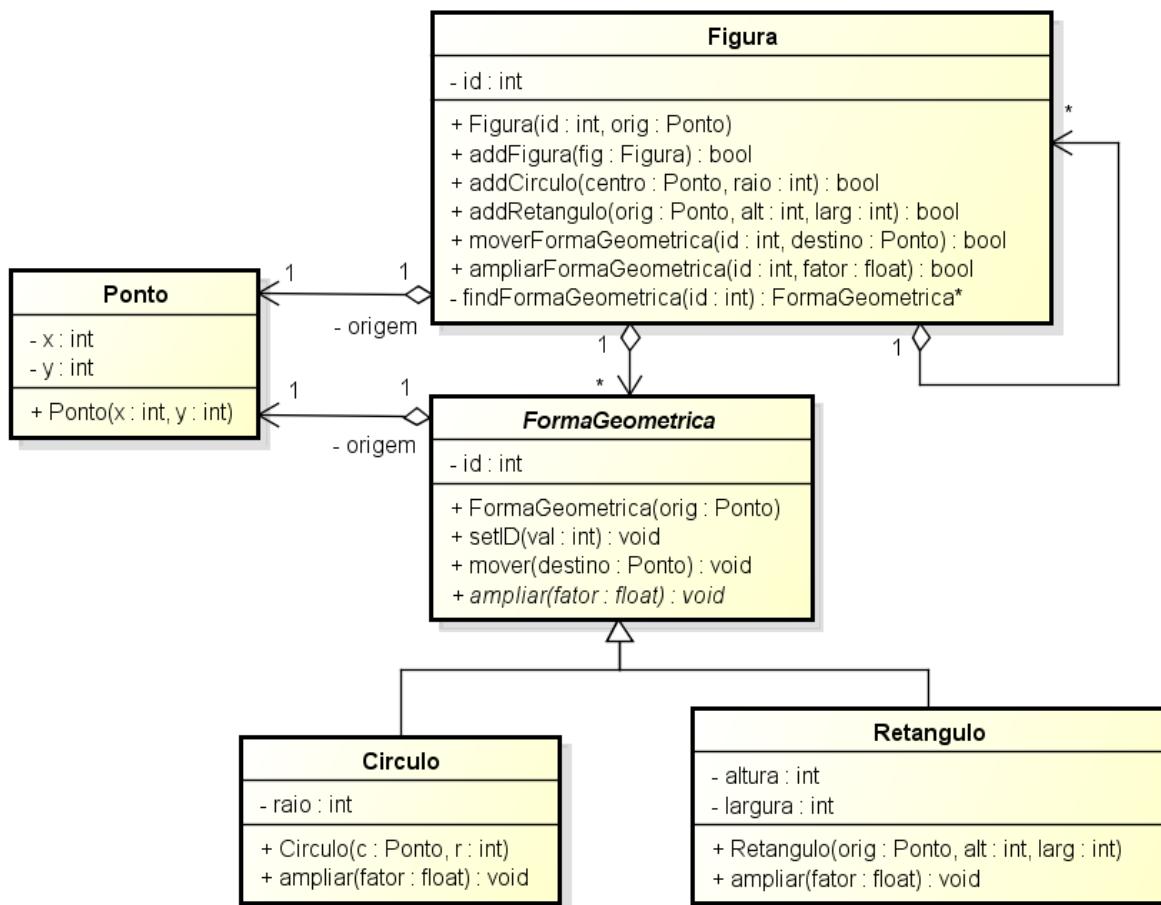
- f) Por que razão numa coleção híbrida são normalmente guardados os apontadores para [0.8 val.]
*Só assim, tirando-se partido da conversão ascendente (upcast), se consegue misturar na coleção elementos de diferentes tipos.
Explicando melhor: na criação da estrutura de dados que define uma coleção híbrida, começa-se por declarar que a mesma coleciona apontadores para uma dada classe base, para depois guardarem-se nela apontadores para objetos de diferentes classes de si derivadas.*

Notas importantes:

1. A duração da prova é de 2 horas.
2. Comece por preencher a zona reservada à sua identificação na folha de exame.

Grupo I
[16 valores]

Uma figura pode ser formada por formas geométricas simples ou, até mesmo, por outras figuras. Cada figura é também caracterizada, tal como as formas geométricas simples, pela posição que ocupa no plano (ponto origem) e por um código inteiro que a identifica. Para além das operações responsáveis pela criação dos elementos gráficos, deverá ser possível mover e ampliar as formas geométricas que integram uma qualquer figura. Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para o problema descrito.



- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e atributos descritos no diagrama, e tendo ainda em conta todas as seguintes considerações: [12 val.]

- *Não defina quaisquer outros métodos ou atributos, a não ser, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis;*
 - *Os ids das Formas Geométricas devem ser atribuídos de forma automática, garantindo que ao 1º elemento é atribuído o id 1, ao 2º o id 2, e assim sucessivamente;*
 - *Tenha em atenção que algumas das entidades poderão ser abstratas (representadas a itálico no diagrama UML);*
 - *As coleções deverão ser implementadas com base no template de classes Colecao ou ColecaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;*
 - *Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;*
 - *Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.*
- b) Acrescente ao problema os métodos que permitam apresentar na saída standard todos os elementos contidos numa figura, com indicação de que tipo de elemento se trata (Figura, Circulo, ...) e de todos os atributos que o caracterizam. [3 val.]
- c) Implemente um pequeno *main* que faça uso de todas as funcionalidades da classe Figura. [1 val.]

Grupo II
[4 valores]

No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com a devida atenção, responda às seguintes questões.

- a) Diga quais são os métodos construtores (definidos explicitamente ou não) que estão presentes em cada uma das classes do problema. [0.8 val.]
- b) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.8 val.]
- c) Diga que consequências é que teria, naquilo que é visualizado, se os destrutores de ambas as classes fossem definidos como virtuais. [0.4 val.]
- d) Faça as alterações que achar necessárias nas classes do problema para que a linha de código que se segue possa surgir na função *main*. [0.5 val.]
- if (eq1 == eq2) cout << "Equipas em igualdade pontual" << endl;
- e) Por fim, diga o que entende por *classe abstrata*, e se alguma das classes do problema pode ser classificada com tal. [0.5 val.]

ANEXO A

```
#include<iostream>
#include<string>
using namespace std;

class Equipa{
    string nome;
    int pontos;
public:
    Equipa(const string &n) : nome(n) {
        pontos = 0;
        cout << "Equipa: ";
        print();
    }
    string getNome() { return nome; }
    void operator+=(int val) { if (val>=1 && val<=3) pontos += val; }
    void print() { cout << nome << " (" << pontos << ")" << endl; }
    ~Equipa() {
        cout << "~Equipa: ";
        print();
    }
};

class Jogo{
    Equipa *anfitria;
    Equipa *visitante;
    int golos[2];
public:
    Jogo(Equipa *eq1, Equipa *eq2) {
        anfitria = eq1; visitante = eq2;
        golos[0] = golos[1] = 0;
        cout << "Inicio do jogo: ";
        print();
    }
    void setResultado(int ga, int gv) { golos[0] = ga; golos[1] = gv; }
    void print() {
        cout << anfitria->getNome() << " " << golos[0] << " - " << golos[1] << " "
            << visitante->getNome() << endl;
    }
    ~Jogo() {
        cout << "Fim do jogo: "; print();
        if (golos[0] > golos[1]) (*anfitria) += 3;
        else if (golos[0] < golos[1]) (*visitante) += 3;
        else { (*anfitria) += 1; (*visitante) += 1; }
    }
};

void main() {
    cout << "-0-" << endl;
    Equipa eq1("City"), eq2("Chelsea");
    cout << "-1-" << endl;
    Jogo jogo1(&eq1, &eq2);
    cout << "-2-" << endl;
    Jogo jogo2(jogo1);
    cout << "-3-" << endl;
    jogo2.setResultado(6, 0);
}
```

ANEXO B

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers{
public:
    bool operator()(const T &left, const T &right) const {
        return (*left < *right);
    }
};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Grupo I (16 valores)

- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [12 val.]

```
class Figura{
    int id;
    Ponto origem;
    Coleccao<Figura> figuras;
    ColeccaoHibrida<FormaGeometrica*> formasGeometricas;
public:
    Figura(int id, const Ponto &orig) : origem(orig) { this->id = id; }
    bool addFigura(const Figura &fig) {
        return figuras.insert(fig);
    }
    bool addCirculo(const Ponto &centro, int raio) {
        FormaGeometrica *f = new Circulo(centro, raio);
        f->setID(formasGeometricas.size() + 1);
        return formasGeometricas.insert(f);
    }
    bool addRetangulo(const Ponto &orig, int alt, int larg) {
        FormaGeometrica *f = new Retangulo(origem, alt, larg);
        f->setID(formasGeometricas.size() + 1);
        return formasGeometricas.insert(f);
    }
    bool moverFormaGeometrica(int id, const Ponto &destino) {
        FormaGeometrica *f = findFormaGeometrica(id);
        if (f == NULL) {
            cout << "Forma Geométrica inexistente!";
            return false;
        }
        else {
            f->mover(destino);
            return true;
        }
    }
    bool ampliarFormaGeometrica(int id, float fator) {
        FormaGeometrica *f = findFormaGeometrica(id);
        if (f == NULL) {
            cout << "Forma Geométrica inexistente!";
            return false;
        }
        else {
            f->ampliar(fator);
            return true;
        }
    }
    bool operator<(const Figura &outra) const { return id < outra.id; }
```

```

private:
    FormaGeometrica *Figura::findFormaGeometrica(int id) {
        Circulo f(Ponto(0, 0), 0);
        f.setID(id);
        return formasGeometricas.find(&f);
    }
};

class Ponto {
    int x, y;
public:
    Ponto(int x, int y) { this->x = x; this->y = y; }
};

class FormaGeometrica {
    int id;
    Ponto origem;
public:
    FormaGeometrica(const Ponto &orig): origem(orig) { }
    void setID(int val) { id = val; }
    void mover(const Ponto &destino) { origem = destino; }
    virtual void ampliar(float fator) = 0;
    bool operator<(const FormaGeometrica &outra)const { return id < outra.id; }
};

class Circulo: public FormaGeometrica{
    int raio;
public:
    Circulo(const Ponto &c, int r) : FormaGeometrica(c) { raio = r; }
    void ampliar(float fator) { raio = (int)(fator*raio+0.5); }
};

class Retangulo: public FormaGeometrica{
    int altura, largura;
public:
    Retangulo(Ponto orig, int alt, int larg) : FormaGeometrica(orig)
    { altura = alt; largura = larg; }
    void ampliar(float fator) {
        altura = (int)(fator*altura + 0.5);
        largura = (int)(fator*largura + 0.5);
    }
};

```

b) Acrescente ao problema os métodos que permitam apresentar na saída standard. [3 val.]

```

void Figura::print() const {
    cout << "Figura " << id << " com origem em "; origem.print();
    ColeccaoHibrida<FormaGeometrica*>::iterator it1;
    for (it1 = formasGeometricas.begin(); it1 != formasGeometricas.end(); it1++)
        (*it1)->print();
    cout << "[";
    Coleccao<Figura>::iterator it2;
    for (it2 = figuras.begin(); it2 != figuras.end(); it2++)
        it2->print();
    cout << "]" << endl;
}

```

```
void Ponto::print() const { cout << "(" << x << "," << y << ")" << endl; }

void FormaGeometrica::print() const {
    cout << "ID: " << id << " Origem: ";
    origem.print();
}
```

c) Implemente um pequeno *main* que faça uso de todas as funcionalidades da classe Figura. [1 val.]

```
void main() {
    Figura fig1(1, Ponto(1,1));
    fig1.addCirculo(Ponto(5, 5), 5);
    fig1.addRetangulo(Ponto(5, 5), 5, 10);
    fig1.moverFormaGeometrica(1, Ponto(15, 20));
    fig1.ampliarFormaGeometrica(2, 1.2f);
    Figura fig2(2, Ponto(0, 0));
    fig2.addFigura(fig1);
    fig2.print();
}
```

Grupo II (4 valores)

a) Diga quais são os métodos construtores (definidos explicitamente ou não) que estão ... [0.8 val.]

Na classe Equipa o construtor de conversão Equipa(const string &n) e o construtor de cópia.

Na classe Jogo o construtor de dois parâmetros Jogo(Equipa *eq1, Equipa *eq2) e o construtor de cópia.

b) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.8 val.]

```
-0-
Equipa: City (0)
Equipa: Chelsea (0)
-1-
Inicio do jogo: City 0 - 0 Chelsea
-2-
-3-
Fim do jogo: City 6 - 0 Chelsea
Fim do jogo: City 0 - 0 Chelsea
~Equipa: Chelsea (1)
~Equipa: City (4)
```

c) Diga que consequências é que teria, naquilo que é visualizado, se os destrutores [0.4 val.]

Nenhuma.

d) Faça as alterações que achar necessárias nas classes do problema para que a linha de ... [0.5 val.]

```
bool Equipa::operator==(const Equipa &outro) { return pontos==outro.pontos; }
```

e) Por fim, diga o que entende por *classe abstrata*, e se alguma das classes do problema .. [0.5 val.]

Uma classe abstrata é uma classe não instanciável, não sendo o caso de nenhuma das classes do problema.

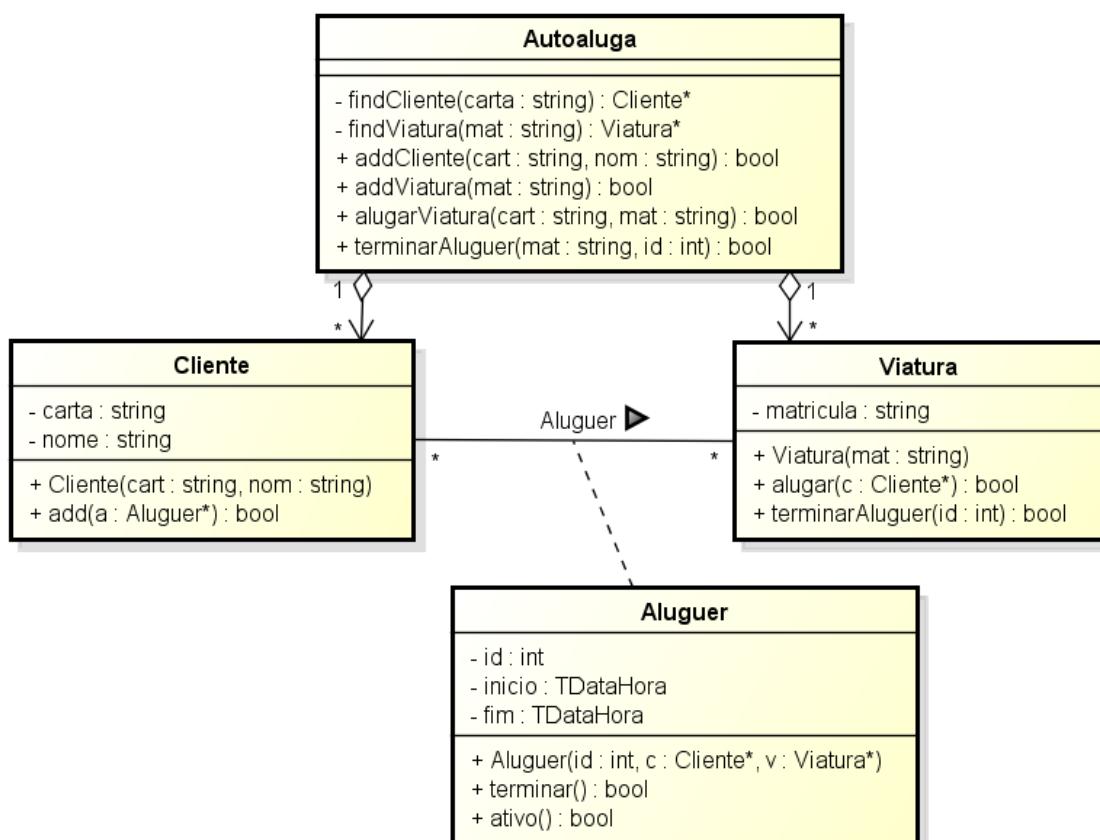
Notas importantes:

1. A duração da prova é de 2 horas.
2. Comece por preencher a zona reservada à sua identificação na folha de exame.

Grupo I

[14 valores]

A Autoaluga é uma empresa de aluguer de automóveis que, encontrando-se em franca expansão, necessita urgentemente de uma aplicação que lhe permita manter registado todos os alugueres das suas viaturas, atuais e passados. Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para a aplicação descrita. O id dos alugueres de cada viatura deverá ser automaticamente atribuído pelo sistema, usando o código 1 para o 1º elemento, o 2 para o 2º, e assim sucessivamente.



- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e atributos descritos no diagrama, e tendo ainda em conta todas as seguintes considerações: [10 val.]

- Não defina quaisquer outros métodos ou atributos, a não ser, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis;
- Para o tipo data/hora é usada a classe `TDataHora`, da qual se apresenta, no Anexo B deste enunciado, o conjunto de métodos e operadores que poderá usar na implementação das suas soluções;

- Admita que a aplicação está a correr em tempo real. Por isso, para as data/horas de início e fim de aluguer deve ser considerada a data/hora corrente. Usar, para o efeito, as funcionalidades da classe *TDataHora*;
- Para simplificar um pouco a solução, no aluguer de uma viatura não precisa de verificar se a viatura em causa está já alugada ou não;
- No momento em que se cria um novo aluguer, deve-se usar a data/hora corrente para o seu início e as 0 horas de 1 de janeiro de 1900 para o seu fim. Dessa forma, para se saber se o aluguer está ativo, bastará verificar-se se a data/hora ‘fim’ é anterior à data/hora ‘inicio’;
- A ação ‘terminar aluguer’ deve apenas consistir na atribuição da data/hora corrente ao atributo ‘fim’ do respetivo aluguer;
- As coleções deverão ser implementadas com base no template de classes *Colecao* ou *ColecaoHibrida* que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;
- Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;
- Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.

b) Acrescente ao problema os métodos que permitam apresentar na saída standard os dados de todos os alugueres realizados por um dado cliente. [4 val.]

Grupo II
[6 valores]

No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com a devida atenção, responda às seguintes questões.

- a) Alguma das classes é abstrata? Justifique. [0.6 val.]
- b) Diga que método deveria ser definido como constante e porquê. [0.6 val.]
- c) Apresente o resultado que será visualizado na saída standard com a execução do programa. [2.8 val.]
- d) Apresente o resultado que seria visualizado na saída standard se o método *print()* da classe base não fosse virtual. Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [0.4 val.]
- e) Tendo em conta as classes definidas no problema, dê uma breve explicação do erro cometido em cada uma das instruções que se seguem: [1.6 val.]
- (1) Figura *fig = new Figura(10,20);
 - (2) Circulo circulos[10];
 - (3) Quadrado q(3); delete q;
 - (4) Circulo c1(1), c2(2); if(c1==c2) cout<< “sao iguais” << endl;

ANEXO A

```
#include<iostream>
using namespace std;

class Figura {
    double x, y;
public:
    Figura(double x, double y) {
        this->x = x; this->y = y;
        cout << "Criada Figura" << endl;
    }
    virtual void print(){cout << "Com origem em " << "(" << x << ", " << y << ")" << endl;}
    virtual void ampliar(double fator) = 0;
    virtual ~Figura() { cout << "Xau Figura" << endl; }
};

class Circulo : public Figura {
    double raio;
public:
    Circulo(double x, double y, double r) : Figura(x, y) {
        raio = r;
        cout << "Criado Círculo" << endl;
    }
    void print() {
        cout << "Círculo de raio " << raio << endl;
        Figura::print();
    }
    void ampliar(double fator) { raio *= fator; }
    ~Circulo() { cout << "Xau Círculo->"; }
};

class Quadrado: public Figura {
    double lado;
public:
    Quadrado(int x, int y, int d) : Figura(x, y) {
        lado = d;
        cout << "Criado Retangulo" << endl;
    }
    void print() {
        cout << "Quadrado de lado " << lado << endl;
        Figura::print();
    }
    void ampliar(double fator) { lado *= fator; }
    ~Quadrado() { cout << "Xau Quadrado->"; }
};

void main() { Figura *f1 = new Quadrado(1, 2, 3);
    Circulo *f2 = new Circulo(4, 5, 6);
    cout << "-1-" << endl;
    Circulo c(*f2);
    cout << "-2-" << endl;
    f1->print(); f2->print();
    c.ampliar(2); c.print();
    cout << "-3-" << endl;
    delete f1;
    delete f2;
    cout << "-4-" << endl;
}
```

ANEXO B

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);  
    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers{
public:
    bool operator()(const T &left, const T &right) const {
        return (*left < *right);
    }
};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);  
    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Classe TDataHora

```
class TDataHora{
public:
    // cria o objeto com a data d/m/a e a hora h:min:s
    TDataHora(int d, int m, int a, int h, int min, double s);
    // cria o objeto com a data hora expressa em dt (string com o formato "x/x/yyyy h:m:s")
    TDataHora(const string & dt);
    // atribui ao objeto a data d/m/a e a hora h:min:s
    bool set(int d, int m, int a, int h, int min, double s);
    // atribui ao objeto a data hora expressa em dt (string com o formato "x/x/yyyy h:m:s")
    void set(const string &dt);
    // devolve uma string com a data hora do objeto expressa no formato "x/x/xx h:m:s"
    string toString() const;
    // verifica se a data hora do objeto é anterior a uma outra
    bool operator<(const TDataHora &outra) const;
    // verifica se a data hora do objeto é igual a uma outra
    bool operator==(const TDataHora &outra) const;
    // devolve um objeto do tipo TDataHora com a data e hora correntes retiradas do sistema
    static TDataHora hoje_agora();
};
```

Grupo I (14 valores)

a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [10 val.]

-50-

```

class Aluguer { -10-
    int id;
    TDataHora inicio;
    TDataHora fim;
    Cliente *cliente;
    Viatura *viatura;
public:
    Aluguer(int id, Cliente *c, Viatura *v) {
        this->id = id;
        inicio = TDataHora::hoje_agora();
        fim = TDataHora(1, 1, 1900, 0, 0, 0);
        cliente = c;
        viatura = v;
    }
    bool terminar() {
        if (ativo()) {
            fim = TDataHora::hoje_agora();
            return true;
        } else return false;
    }
    bool ativo() const { return fim < inicio; }
    bool operator<(const Aluguer &outro) const { return id < outro.id; }
};

class Viatura { -14-
    string matricula;
    Colecao<Aluguer> alugueres;
public:
    Viatura(string mat) :matricula(mat){}
    bool alugar(Cliente *c) {
        int id = alugueres.size() + 1;
        Aluguer a(id, c, this);
        alugueres.insert(a);
        return c->add(alugueres.find(a));
    }
    bool terminarAluguer(int id){
        Aluguer a(id, NULL, NULL);
        Aluguer *p = alugueres.find(a);
        if (p == NULL) return false; //aluguer inexistente!
        else return p->terminar();
    }
    bool operator<(const Viatura &outra) const{return matricula<outra.matricula;}
};

```

```

class Cliente{    -6-
    string carta, nome;
    Colecao<Aluguer*> alugueres;
public:
    Cliente(string cart, string nom) :carta(cart), nome(nom){}
    bool add(Aluguer *a) { return alugueres.insert(a); }
    bool operator<(const Cliente &outro) const {return carta < outro.carta;}
};

class Autoaluga{    -20-
    Colecao<Viatura> viaturas;
    Colecao<Cliente> clientes;

    Cliente *findCliente(string cart) const{
        Cliente c(cart, "");
        return clientes.find(c);
    }
    Viatura *findViatura(string mat) const{
        Viatura v(mat);
        return viaturas.find(v);
    }
public:
    bool addCliente(string cart, string nom) {
        Cliente c(cart, nom);
        return clientes.insert(c);
    }

    bool addViatura(string mat) {
        Viatura v(mat);
        return viaturas.insert(v);
    }

    bool alugarViatura(string cart, string mat) {
        Cliente *c = findCliente(cart);
        if (c == NULL) return false; //Cliente inexistente!
        else {
            Viatura *v = findViatura(mat);
            if (v == NULL) return false; //Viatura inexistente!
            else return v->alugar(c);
        }
    }

    bool terminarAluguer(string mat, int id) {
        Viatura *v = findViatura(mat);
        if (v == NULL) return false; //Viatura inexistente!
        else return v->terminarAluguer(id);
    }
};

```

b) Acrescente ao problema os métodos que permitam apresentar na saída... [4 val.]

-20-

```
void Autoaluga::mostrarAlugueresCliente(string cart) const {           2
    Cliente *c = findCliente(cart);                                     1
    if (c != NULL)                                                       1
        c->mostrarAlugueres();                                            1
}

void Cliente::mostrarAlugueres() const {                                     2
    Colecao<Aluguer*>::iterator it;                                       1
    for (it = alugueres.begin(); it != alugueres.end(); it++)            2
        (*it)->print();                                                 2
}

void Aluguer::print()const {                                              2
    cout << "Aluguer " << id << " da viatura " << viatura->getMatricula(); 1
    if (ativo())
        cout << " iniciado a "<<inicio.toString()<< " e ainda decorrer" << endl; 1
    else cout << " de " << inicio.toString()<< " a " << fim.toString()<< endl; 2
}

string Viatura::getMatricula()const { return matricula; }                  2
```

Grupo II (6 valores)

a) Alguma das classes é abstrata. Justifique. [0.6 val.]

A classe Figura, dado que possui um método virtual puro (abstrato).

b) Diga que método deveria ser constante e porquê. [0.6 val.]

O método print(), dado que não altera o estado do objeto (valor dos atributos).

c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [2.8 val.]

```
Criada Figura
Criado Retangulo
Criada Figura
Criado Círculo
-1-
-2-
Quadrado de lado 3
Com origem em (1,2)
Círculo de raio 6
Com origem em (4,5)
Círculo de raio 12
Com origem em (4,5)
-3-
Xau Quadrado->Xau Figura
Xau Círculo->Xau Figura
-4-
Xau Círculo->Xau Figura
```

d) Apresente o resultado que seria visualizado na saída standard se o método `print()` da classe base não fosse virtual. [0.4 val.]

A única diferença é que deixava de aparecer a frase “Quadrado de lado 3”.

e) Tendo em conta as classes definidas no problema, dê uma breve explicação do erro cometido em cada uma das instruções que se seguem: [1.6 val.]

(1) Figura *fig = new Figura(10,20);

Sendo `Figura` uma classe abstrata, a mesma não pode ser instanciada.

(2) Circulo circulos[10];

Como a classe `Circulo` não tem construtor por defeito, não é possível criar-se um array de instâncias dessa classe.

(3) Quadrado q(3); delete q;

O operador `delete` só pode ser usado para desalocar objetos criados de forma dinâmica.

Outra resposta válida: a classe `Quadrado` não dispõe do construtor de conversão de parâmetro inteiro; na sua instanciação deveria ser invocado o construtor de 3 parâmetros inteiros.

(4) Circulo c1(1), c2(2); if(c1==c2) cout<< “sao iguais” << endl;

A classe `Circulo` não dispõe do operador igualdade (`==`), nem o mesmo se encontra sobreescarregado como função global para esse tipo de operandos.

Outra resposta válida: a classe `Circulo` não dispõe do construtor de conversão de parâmetro inteiro; na sua instanciação deveria ser invocado o construtor de 3 parâmetros inteiros.

Engenharia Informática
Informática de Gestão

Época de Recurso – 13 de fevereiro de 2017

Notas importantes:

1. *O Grupo III é de resolução facultativa e destina-se a substituir as notas do trabalho prático e minitestes*
 2. *Duração da prova: 1h30 (Grupos I e II) + 1h (Grupo III)*
 3. *Comece por preencher a zona reservada à sua identificação na folha de exame;*
 4. *Resolva o Grupo III em folha de exame separada.*
-

Grupo I
(5 valores)

No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com a devida atenção, responda às seguintes questões.

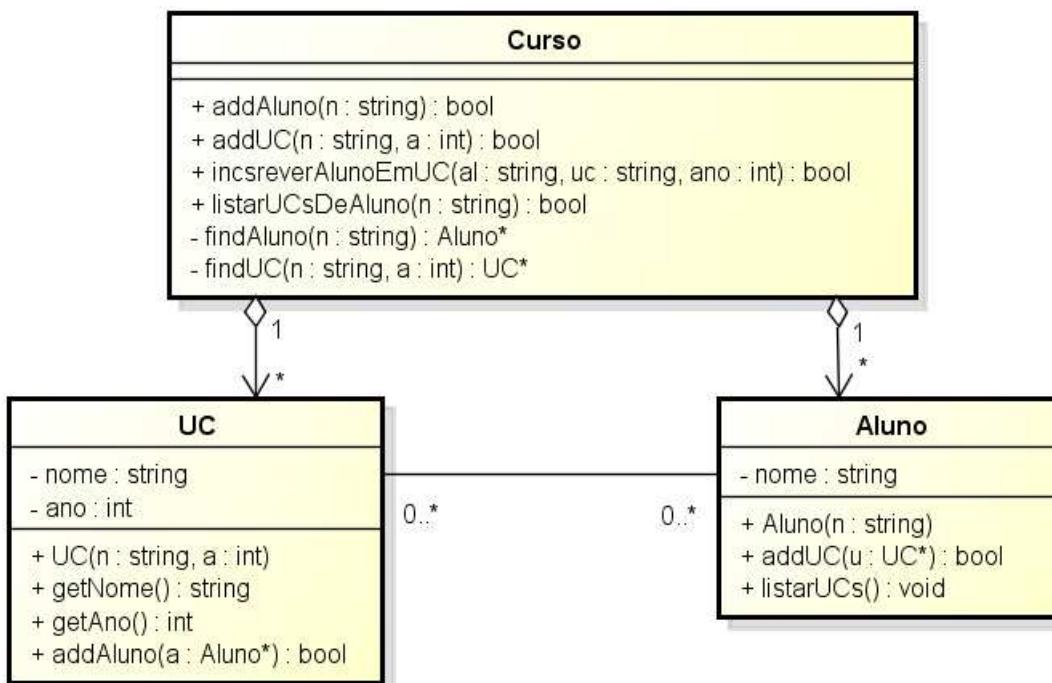
- a) Diga o que entende por classe abstrata, se alguma das classes do problema é abstrata e de que forma é que se podem definir classes abstratas em C++. [1 val.]
- b) Deveríamos ter definido, em ambas as classes, o método `toString()` como constante? Justifique. [.6 val.]
- c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.8 val.]
- d) Apresente o resultado que seria visualizado na saída standard se os métodos destrutor e `toString()`, da classe `Base`, fossem virtuais. Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [.6 val.]
- e) Se definíssemos o método construtor da classe `AErasmus` como se segue, estaríamos a introduzir, com essa alteração, dois erros. Diga quais. [1 val.]

```
AErasmus(const string &nom, const string &p) {  
    nome = nom;  
    pais = p;  
    cout << "Erasmus de " << p << endl;  
}
```

Grupo II
(15 valores)

Suponha que pretende desenvolver uma pequena aplicação que lhe permita gerir as inscrições dos alunos nas diferentes unidades curriculares (UCs) do seu curso. No sentido de simplificar a solução, considere que o aluno é caracterizado unicamente pelo seu nome, e que cada UC é caracterizada quer pelo nome quer pelo ano letivo a que diz respeito. Tenha, portanto, em atenção, que no sistema vão coexistir várias UCs com o mesmo nome, mas de anos diferentes. Para além das funcionalidades básicas de gestão das entidades, a aplicação deverá permitir listar as UCs em que se encontra inscrito um dado aluno do curso.

Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para a aplicação descrita.



- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e atributos descritos no diagrama. Não defina quaisquer outros métodos ou atributos, a não ser, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis. [12 val.]

NOTA 1: As coleções deverão ser implementadas com base no template de classes Colecao ou ColecaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;

NOTA 2: Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;

NOTA 3: Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.

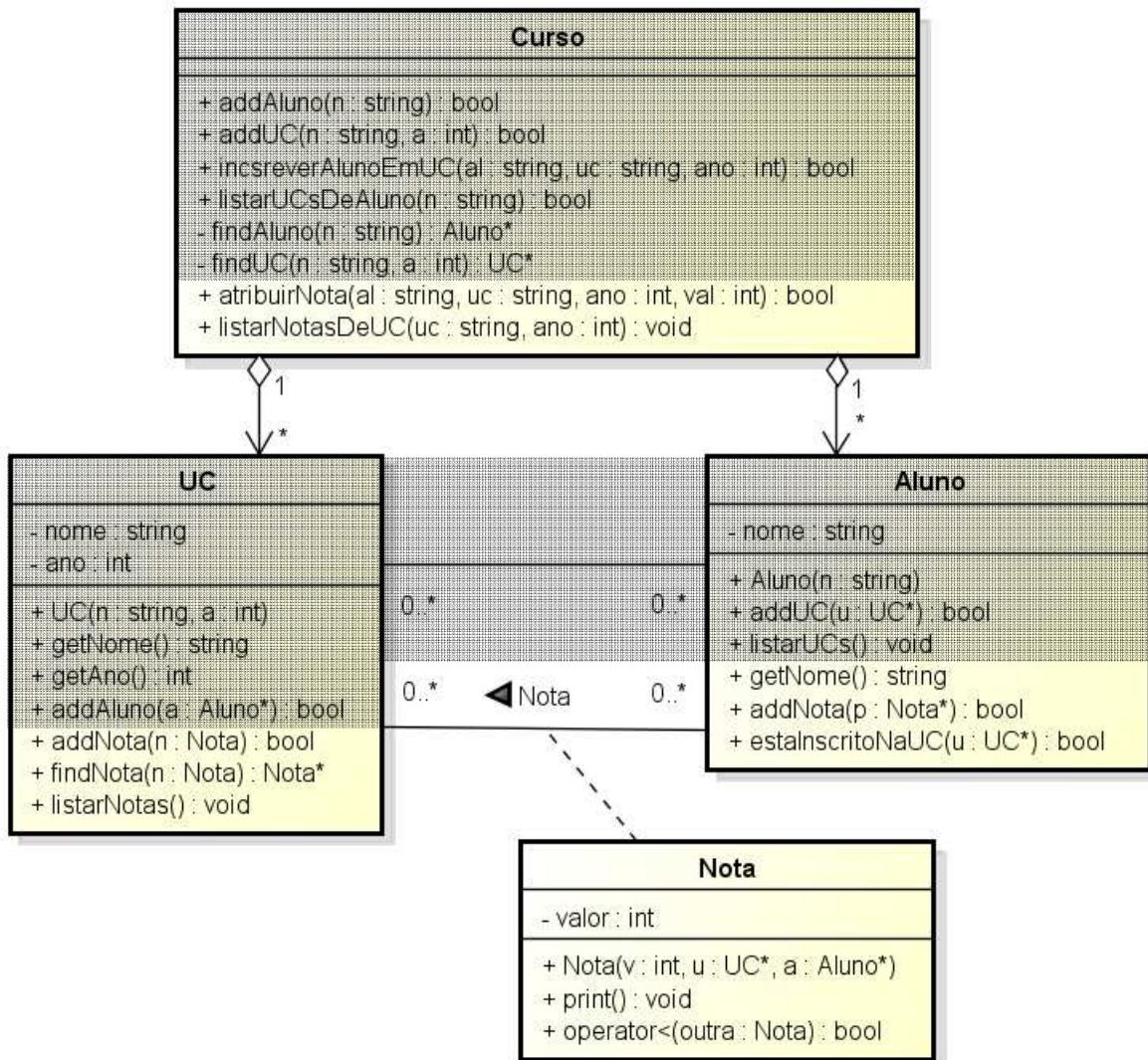
- b) Acrescente ao problema os métodos que permitam remover do sistema uma dada unidade curricular, garantido a eliminação prévia de qualquer associação que possa existir dessa UC a outras entidades dos sistemas. [3 val.]

Grupo III

(10 valores)

*NOTA: O problema que se segue é um exercício suplementar e facultativo que se destina a substituir a classificação obtida durante o período letivo (miniteste+trabalho). Com a sua resolução, ou tentativa de resolução, essa componente de avaliação deixará de contar para a época de recurso, e a sua nota final passará a ser determinada pela seguinte fórmula: (GI+GII+GIII)*2/3.*

No seguimento do exercício do grupo II, pretende-se agora acrescentar à aplicação a capacidade de registar as notas dos alunos nas diferentes UCs. Segue-se o diagrama de classes UML que descreve de forma precisa a nova solução que agora se pretende.



Este exercício destina-se a acrescentar novas funcionalidades ao problema que começou a implementar no grupo de questões anterior. Por isso, nas suas respostas deve ter sempre em conta o código que já implementou anteriormente para o problema.

Defina então em C++ a nova classe Nota e acrescente às restantes os métodos e atributos que assegurem as novas funcionalidades do problema (métodos não sombreados do diagrama). [10 val]

NOTA: A adição de uma nova nota a uma UC apenas deverá ser permitida se o aluno em causa estiver já inscrito nessa UC.

ANEXO A

```
#include<iostream>
#include<string>
using namespace std;

class Aluno {
    string nome;
public:
    Aluno(const string &n) {
        cout << "Novo aluno ";
        nome = n;
    }
    string toString() { return "Nome: " + nome; }
    ~Aluno() { cout << nome << " cancelou a sua matricula" << endl; }
};

class AErasmus : public Aluno {
    string pais;
public:
    AErasmus(const string &nom, const string &p) :Aluno(nom) {
        pais = p;
        cout << "Erasmus de " << p << endl;
    }
    string toString() { return Aluno::toString() + " Aluno de " + pais; }
    ~AErasmus() { cout << "O aluno de " << pais << " "; }
};

void main() {
    Aluno *v[3];
    cout << "OUTPUT 0:" << endl;
    v[0] = new AErasmus("Pirlo", "Italia");
    cout << "OUTPUT 1:" << endl;
    AErasmus ae("Paco", "Espanha");
    v[1] = &ae;
    cout << "OUTPUT 2:" << endl;
    Aluno a("Ana");
    v[2] = &a;
    cout << endl << "OUTPUT 3:" << endl;
    int i = 0;
    cout << "0: " << v[0]->toString() << endl;
    cout << "1: " << v[1]->toString() << endl;
    cout << "2: " << v[2]->toString() << endl;
    cout << "OUTPUT 4:" << endl;
    delete v[0];
    cout << "OUTPUT 5:" << endl;
}
```

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers{
public:
    bool operator()(const T &left, const T &right) const {
        return (*left < *right);
    }
};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Grupo I
(5 valores)

- a) Diga o que entende por classe abstrata, se alguma das classes do problema é abstrata e de que forma é que se podem definir classes abstratas em C++. [1 val.]

Classe abstrata é uma classe não insaciável; nenhuma das classes do problema é abstrata; a forma mais habitual de definir em C++ uma classe abstrata é incluir-lhe um ou mais métodos abstratos (virtuais puros).

- b) Deveríamos ter definido, em ambas as classes, o método `toString()` como constate? Justifique. [.6 val.]

Claro que sim, pois em nenhuma das classes o método modifica o valor dos atributos. É claramente um método de consulta.

- c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.8 val.]

OUTPUT 0:

Novo aluno Erasmus de Italia

OUTPUT 1:

Novo aluno Erasmus de Espanha

OUTPUT 2:

Novo aluno

OUTPUT 3:

0: Nome: Pirlo

1: Nome: Paco

2: Nome: Ana

OUTPUT 4:

Pirlo cancelou a sua matricula

OUTPUT 5:

Ana cancelou a sua matricula

O aluno de Espanha Paco cancelou a sua matricula

- d) Apresente o resultado que seria visualizado na saída standard se os métodos destrutor e `toString()`, da classe Base, fossem virtuais. Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [.6 val.]

OUTPUT 3:

0: Nome: Pirlo Aluno de Italia

1: Nome: Paco Aluno de Espanha

2: Nome: Ana

OUTPUT 4:

O aluno de Italia Pirlo cancelou a sua matricula

- e) Se definíssemos o método construtor da classe AErasmus como se segue, estaríamos a introduzir, com essa alteração, dois erros. Diga quais. [1 val.]

Erro 1: o atributo nome, sendo privado na classe base, surge oculto na classe derivada AErasmus; logo não se pode aceder ao mesmo de forma direta (é o que se tenta fazer na instrução nome = nom;);

Erro 2: uma vez que não tem qualquer lista de inicialização, quando executado este construtor, por omissão é invocado o construtor por defeito da classe base, método que, como se sabe, não existe.

Grupo II (15 valores)

a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [12 val.]

```

class Curso{
    Coleccao<UC> ucs;           2
    Coleccao<Aluno> alunos;       2
public:
    bool addAluno(const string &n) {
        Aluno a(n);
        return alunos.insert(a);
    }

    bool addUC(const string &n, int a){           2
        UC uc(n,a);
        return ucs.insert(uc);
    }
    bool incsreverAlunoEmUC(const string &al, const string &uc, int ano){      4
        Aluno *a=findAluno(al);
        if (a!=NULL) {
            UC *u = findUC(uc, ano);
            if (u!=NULL)
                if (a->addUC(u)) return u->addAluno(a);
        }
        return false;
    }
    bool listarUCsDeAluno(const string &name){           3
        Aluno *a=findAluno(name);
        if (a != NULL) {
            a->listarUCs();
            return true;
        }
        else return false;
    }
private:
    Aluno *findAluno(const string &n){           1
        Aluno a(n);
        return alunos.find(a);
    }
    UC *findUC(const string &n, int a){           2
        UC uc(n,a);
        return ucs.find(uc);
    }
};

class Aluno{           1
    string nome;
    Coleccao<UC*> ucs;           2
public:
    Aluno(const string &n): nome(n){}     1
}

```

```

bool addUC(UC *u){ return ucs.insert(u); }           2

void listarUCs(){                                     3
    Coleccao<UC*>::iterator it;
    cout<<"UCs do aluno "<<nome<<":\n";
    for(it=ucs.begin(); it!=ucs.end(); it++)
        cout<<(*it)->getNome()<< " "<<(*it)->getAno()<<endl;
}

bool operator<(const Aluno& outro) const {return nome<outro.nome;}      2
};

class UC{                                         1
    string nome;
    int ano;
    Coleccao<Aluno*> alunos;                  2
public:
    UC(const string &n, int a) {nome=n; ano=a;}
    string getNome() const {return nome;}
    int getAno() const {return ano;}
    bool operator<(const UC& outra) const {      3
        if (nome != outra.nome) return nome < outra.nome;
        else return ano<outra.ano;
    }

    bool addAluno(Aluno *a){return alunos.insert(a);}
}

```

b) Acrescente ao problema os métodos que permitam remover do sistema uma dada unidade... [3 val.]

```

void Curso::remUC(const string &n, int a) {           4
    UC *u=findUC(n, a);
    if(u!=NULL){
        u->remAlunos();
        UC uc(n,a);
        ucs.erase(uc);
    }else cout << "UC " << n << " de " <<a<< " nao existe!\n";
}

void UC::remAlunos(){                                4
    Coleccao<Aluno*>::iterator it;
    for(it=alunos.begin(); it!=alunos.end(); it++)
        (*it)->remUC(this);
    alunos.clear();
}

void Aluno::remUC(UC *u) {ucs.erase(u);}             2

```

Grupo III
(10 valores)

Defina então em C++ a nova classe Nota e acrescente às restantes os métodos ... [10 val]

```
class Nota {                                1
    int valor;
    UC *uc;                               1
    Aluno *aluno;                          1
public:
    Nota(int v, UC *u, Aluno *a) {        1
        uc = u;
        aluno = a;
        valor = v;
    }

    void print()const{                   2
        cout << aluno->getNome() << " " << valor << endl;
    }

    Bool operator<(const Nota &outra) const {      2
        return(aluno->getNome() < outra.aluno->getNome());
    }
};

class UC{
    ...
    Coleccao<Nota> notas;                2
public:
    ...

    bool addNota(const Nota &n) {          2
        return notas.insert(n);
    }

    Nota* findNota(const Nota &n) {        2
        return notas.find(n);
    }

    void listarNotas() {                  3
        Coleccao<Nota>::iterator it;
        cout << "Pauta de " << nome << " " << ano << ":" \n";
        for (it = notas.begin(); it != notas.end(); it++)
            it->print();
    }
};
```

```
class Aluno{  
    ...  
    Coleccao<Nota*> notas;           2  
public:  
    ...  
    string getName() const { return nome; }      1  
  
    bool addNota(Nota *p) { return notas.insert(p); }      2  
  
    bool estaInscritoNaUC(UC *u) { return ucs.find(u)!=NULL; }      2  
};  
  
bool Curso::atribuirNota(string al, string uc, int ano, int val) {      4  
    UC *u = findUC(uc,ano);  
    if (u != NULL) {  
        Aluno *a = findAluno(al);  
        if (a != NULL)  
            if (a->inscritoNaUC(u)) {  
                Nota n(val, u, a);  
                if (u->addNota(n))  
                    return a->addNota(u->findNota(n));  
            }  
    }  
    return false;  
}  
  
void Curso::listarNotasDeUC(string uc, int ano) {      2  
    UC *u = findUC(uc, ano);  
    if (u != NULL) u->listarNotas();  
}
```

Engenharia Informática
Informática de Gestão

Época de Recurso – 7 de fevereiro de 2015

Notas importantes:

1. *O Grupo III é de resolução facultativa e destina-se a substituir as notas do trabalho prático e minitestes*
 2. *Duração da prova: 1h30 (Grupos I e II) + 1h (Grupo III)*
 3. *Comece por preencher a zona reservada à sua identificação na folha de exame;*
 4. *Resolva os três grupos em folhas de exame separadas.*
-

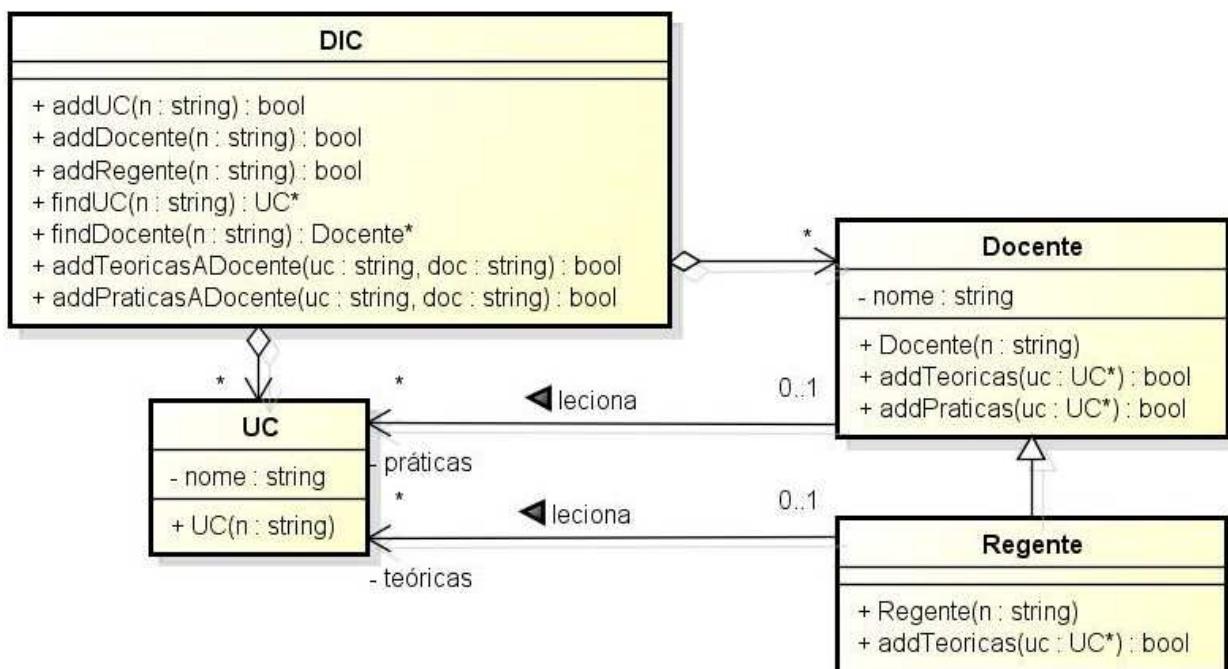
Grupo I
(7.2 valores)

No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com a devida atenção, responda às seguintes questões.

- a) Identifique o tipo de relação existente entre as classes *Desenho* e *Figura*. [0.2 val.]
- b) Alguma das classes é abstrata? Se sim, diga qual e o que a torna abstrata. [0.7 val.]
- c) Diga quais os métodos construtores que estarão presentes em cada uma das classes do problema. [1.6 val.]
- d) Apresente o resultado que será visualizado na saída standard com a execução do programa. [4.0 val.]
- e) O que seria visualizado se o método *print* da classe *Figura* passasse a virtual? Refira-se apenas à parte da visualização que achar que se altera em relação ao output da alínea anterior. [0.7 val.]

Grupo II
(12.8 valores)

Suponha que o incumbiram de desenvolver uma pequena aplicação que permita gerir a distribuição de serviço docente do depart. de informática e comunicações (DIC). Pretende-se com essa aplicação que cada docente saiba quais as unidades curriculares (UCs) em que leciona as aulas práticas e quais as UCs em que leciona as aulas teóricas. Se as aulas práticas podem ser asseguradas por um qualquer docente, as teóricas só poderão ser asseguradas por regentes. Com o objetivo de simplificar o problema, assume-se que a UC desconhece os docentes que a lecionam (associação unidirecional) e que cada uma das suas componentes, teórica ou prática, é lecionada por inteiro por um único docente. Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para a aplicação descrita.



- a) Defina em C++ todas as classes do problema, respeitando integralmente as associações, os métodos e os atributos descritos no diagrama. Não implemente quaisquer outros métodos ou atributos; apenas acrescente, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis. [9.0 val.]

NOTA 1: Quando associa a componente teórica ou prática duma UC a um docente/regente (addTeoricas() e addPraticas()) não precisa de verificar se essas componentes já estão a ser asseguradas por outros docentes;

NOTA 2: As coleções deverão ser implementadas com base no template de classes Colecao ou ColecaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;

NOTA 3: Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;

NOTA 4: Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.

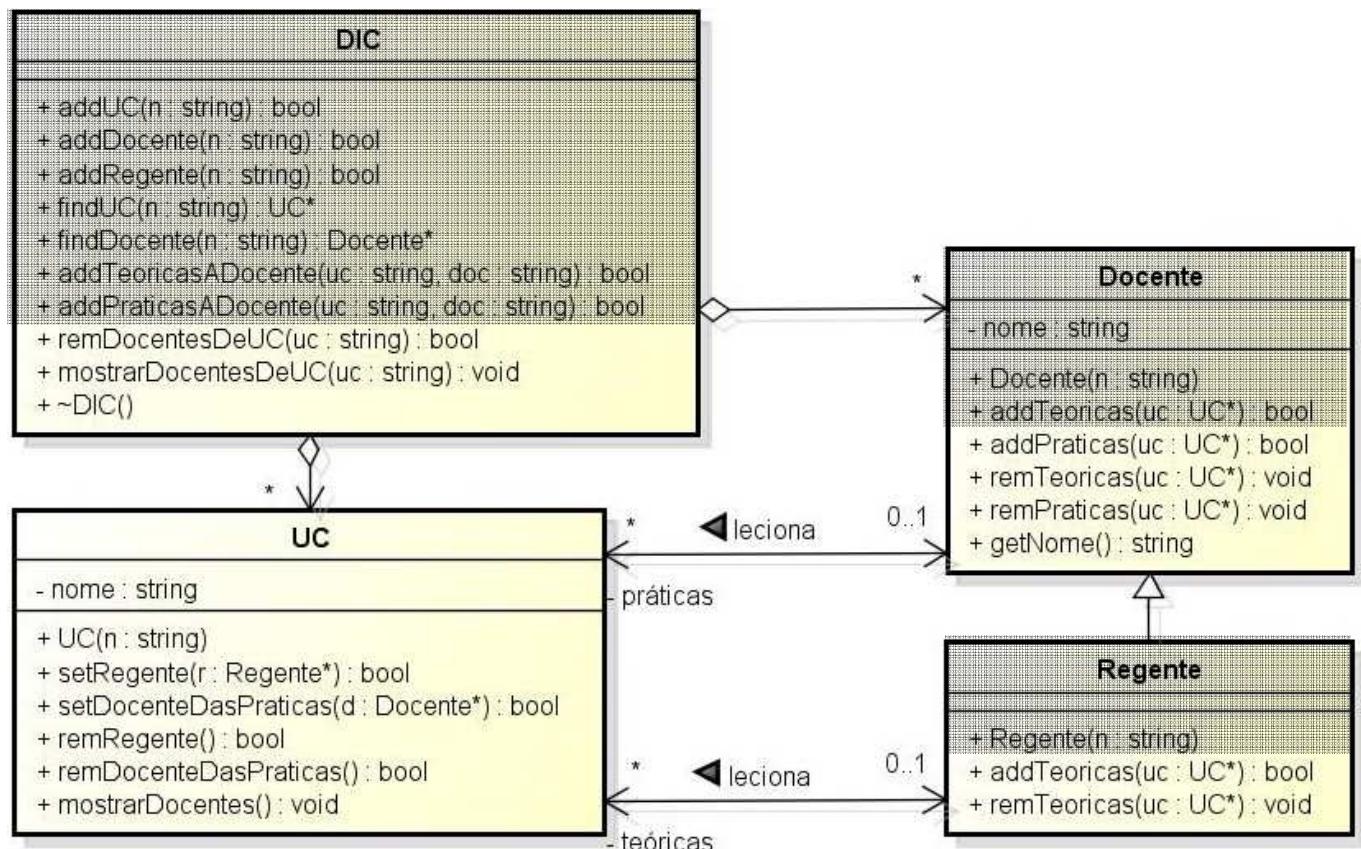
- b) Acrescente ao problema os métodos que permitam mostrar o nome de todas as UCs lecionadas por um dado docente (que pode ser regente). [2.4 val.]
- c) Escreva um pequeno *main* que faça uso de todas as funcionalidades da aplicação. [1.4 val.]

Grupo III

(10 valores)

*NOTA: O problema que se segue é um exercício suplementar e facultativo que se destina a substituir a classificação obtida durante o período letivo (miniteste+trabalho). Com a sua resolução, ou tentativa de resolução, essa componente de avaliação deixará de contar para a época de recurso, e a sua nota final será: (G1+GII+GIII)*2/3.*

A ideia agora é permitir que cada UC reconheça quem é o regente que está a lecionar as suas aulas teóricas e quem é o docente que está a lecionar as suas aulas práticas (repare que as associações entre essas entidades são agora bidirecionais). Adicionalmente, pretende-se acrescentar à aplicação a capacidade de dissociar os docentes/regentes das UCs, de mostrar os docentes de uma determinada UC e de destruição adequada da entidade centralizadora (DIC). Segue-se o diagrama de classes UML que descreve de forma precisa a nova solução que agora se pretende.



Este exercício destina-se a acrescentar novas funcionalidades ao problema que começou a implementar no grupo de questões anterior. Por isso, nas suas respostas deve ter sempre em conta o código que já implementou anteriormente para o problema.

Defina então em C++ novamente toda a classe UC e acrescente às restantes os métodos que asseguram as novas funcionalidades do problema (métodos não sombreados do diagrama). [10 val]

NOTA: Agora a associação de uma componente (teórica ou prática) da UC a um docente/regente (`addTeoricas()` e `addPraticas()`) não deve ser permitida se essa componente já estiver a ser assegurada por outro docente;

ANEXO A

```
#include<iostream>
using namespace std;

class Figura{
    int x, y;
public:
    Figura(){ x = y = 0;
               cout << "Criada Figura na origem" << endl;
    }
    Figura(int x, int y){
        this->x = x;
        this->y = y;
        cout << "Criada ";
        print();
    }
    void print(){ cout << "Figura na posicao (" << x << ',' << y << ')' << endl; }
    ~Figura(){ cout << "Destruida ";
               print();
    }
};

class Desenho{
    Figura *figuras;
    int n, max;
public:
    Desenho(){
        n = max = 0;
        cout << "Criado Desenho que nao vai poder incluir figuras" << endl;
    }
    Desenho(int m){
        max = m; n = 0;
        figuras = new Figura[max];
        cout << "Criado Desenho com capacidade para incluir " << max
            << " figuras" << endl;
    }
    void addFigura(const Figura &f){ if (n<max) figuras[n++] = f; }
    void print(){
        for (int i = 0; i<n; i++) figuras[i].print();
    }
    ~Desenho(){
        if (figuras!=NULL) delete[] figuras;
        cout << "Desenho com " << n << " figuras destruido" << endl;
    }
};

void main(){
    cout << "####Criacao 1###" << endl;
    Figura f0, f1(2,3);
    cout << "####Criacao 2###" << endl;
    Desenho d(3);
    cout << "####Adicao###" << endl;
    d.addFigura(f0);
    d.addFigura(f1);
    cout << "####Print###" << endl;
    d.print();
    cout << "####Destruicao###" << endl;
}
```

ANEXO B

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers{
public:
    bool operator()(const T &left, const T &right) const {
        return (*left < *right);
    }
};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Grupo I
(7.2 valores)

- a) Identifique o tipo de relação existente entre as classes *Desenho* e *Figura*. [0.2 val.]

Agregação. Desenho agrupa Figura, numa relação de um para muitos.

- b) Alguma das classes é abstrata? Se sim, diga qual e o que a torna abstrata. [0.7 val.]

Não, nenhuma é abstrata.

- c) Diga quais os métodos construtores que estarão presentes em cada uma das classes ... [1.6 val.]

Para além do construtores por omissão e de cópia, que estão presentes em ambas as classes, a classe Desenho dispõe ainda do construtor de conversão Desenho(int) e a classe Figura do construtor Figura(int, int).

- d) Apresente o resultado que será visualizado na saída standard com a execução do programa. [4.0 val.]

```
####Criacao 1###
Criada Figura na origem      [2]
Criada Figura na posicao (2,3)    [2]
####Criacao 2###
Criada Figura na origem      [1]
Criada Figura na origem      [1]
Criada Figura na origem      [1]
Criado Desenho com capacidade para incluir 3 figuras      [2]
####Adicao###
####Print###
Figura na posicao (0,0)      [2]
Figura na posicao (2,3)      [2]
####Destruicao###
Destruida Figura na posicao (0,0)      [1]
Destruida Figura na posicao (2,3)      [1]
Destruida Figura na posicao (0,0)      [1]
Desenho com 2 figuras destruido      [2]
Destruida Figura na posicao (2,3)      [1]
Destruida Figura na posicao (0,0)      [1]
```

- e) O que seria visualizado se o método *print* da classe *Escola* não fosse virtual? Refira-se... [0.7 val.]

Seria visualizado o mesmo output. Nada se alteraria.

Grupo II (12.8 valores)

- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [9.0 val]

```
class DIC{           [4.2 val.]
    Colecao<UC> ucs;          [1]
    ColecaoHibrida<Docente*> docentes;      [2]
public:
    bool addUC(string n){       [1]
        UC uc(n);
        return ucs.insert(uc);
    }
    bool addDocente(string n){   [1]
        Docente *d = new Docente(n);
        return docentes.insert(d);
    }
    bool addRegente(string n){   [1]
        Regente *r = new Regente(n);
        return docentes.insert(r);
    }

    UC *findUC(string n){       [1]
        UC uc(n);
        return ucs.find(uc);
    }
    Docente *findDocente(string n){   [1]
        Docente d(n);
        return docentes.find(&d);
    }

    bool addTeoricasADocente(string uc, string doc){      [3]
        UC *puc = findUC(uc);
        if (puc != NULL){
            Docente *pdoc = findDocente(doc);
            if (pdoc != NULL)
                return pdoc->addTeoricas(puc);
            else { cout << "Docente nao existente" << endl; return false; }
        }else { cout << "UC nao existente" << endl; return false; }
    }

    bool addPraticasADocente(string uc, string doc){      [3]
        UC *puc=findUC(uc);
        if(puc!=NULL){
            Docente *pdoc=findDocente(doc);
            if(pdoc!=NULL)
                return pdoc->addPraticas(puc);
            else { cout << "Docente nao existente" << endl; return false; }
        }else { cout << "UC nao existente" << endl; return false; }
    }
};

class UC{           [0.9 val.]
    string nome;          [1]
public:
    UC(string n) : nome(n){}
    bool operator<(const UC &outra) const { return nome<outra.nome; } [1]
};
```

```

class Docente{ [2.1 val.]
    string nome;
    Colecao<UC *> praticas; [1]
public:
    Docente(string n): nome(n){} [1]
    bool addPraticas(UC *uc){ return praticas.insert(uc); } [2]
    virtual bool addTeoricas(UC *uc){ return false; } [2]
    bool operator<(const Docente &outro) const { return nome<outro.nome; } [1]
};

class Regente: public Docente{ [2] [1.8 val.]
    Colecao<UC *> teoricas; [1]
public:
    Regente(string n) : Docente(n){} [1]
    bool addTeoricas(UC *uc){ return teoricas.insert(uc); } [2]
};

```

b) Acrescente ao problema os métodos que permitam mostrar o nome de todas as UCs... [2.4 val.]

```

void DIC::mostrarUCsLecionasPorDocente(string doc){ [2]
    Docente *pdoc = findDocente(doc);
    if (pdoc != NULL)
        return pdoc->mostrarUCsLecionas();
    else cout << "Docente nao existente" << endl;
}

virtual void Docente::mostrarUCsLecionas(){ [2]
    cout << "Aulas do docente " << nome << endl;
    cout << "Aulas praticas :" << endl;
    Colecao<UC*>::iterator it;
    for (it = praticas.begin(); it != praticas.end(); it++)
        cout << (*it)->getNome() << ' ';
    cout << endl;
}

void Regente::mostrarUCsLecionas(){ [3]
    Docente::mostrarUCsLecionas();
    cout << "Aulas teoricas:" << endl;
    Colecao<UC*>::iterator it;
    for (it = teoricas.begin(); it != teoricas.end(); it++)
        cout << (*it)->getNome() << ' ';
    cout << endl;
}

string UC::getNome(){ return nome; } [1]

```

c) Implemente um pequeno main que faça uso de todas as funcionalidades da aplicação. [1.4 val.]

```

void main(){
    DIC dic;
    dic.addUC("POO");
    dic.addDocente("Ana");
    dic.addRegente("Adele");
    dic.addTeoricasADocente("POO", "Adele");
    dic.addPraticasADocente("POO", "Ana");
    dic.mostrarUCsLecionasPorDocente("Ana");
    dic.mostrarUCsLecionasPorDocente("Adele");
}

```

Grupo III (10 valores)

Defina em C++ as novas classes Jogador e Presenca, e acrescente às já existentes ... [10 val]

```
//classe UC: [4.2 val.]
class UC{
    string nome;
    Regente *reg;
    Docente *docP; [2]
public:
    UC(string n) : nome(n){ docP = reg = NULL;} [1]
    bool setRegente(Regente *r){ [2]
        if (reg == NULL) { reg = r; return true; }
        else return false;
    }
    bool setDocenteDasPraticas(Docente *d){ [2]
        if (docP == NULL) { docP = d; return true; }
        else return false;
    }
    bool remRegente(){ [2]
        if (reg == NULL) return false;
        else {
            reg->remTeoricas(this);
            reg = NULL;
            return true;
        }
    }
    bool remDocenteDasPraticas(){ [2]
        if (docP == NULL) return false;
        else {
            docP->remPraticas(this);
            docP = NULL;
            return true;
        }
    }
    void mostrarDocentes()const{ [3]
        cout << "Docentes da UC " << nome << ':' << endl;
        if (reg != NULL) cout << reg->getNome() << " (Regente)" << endl;
        else cout << "Nao tem regente associado!" << endl;
        if (docP != NULL) cout << docP->getNome() << " (Assistente)" << endl;
        else cout << "Nao tem assistente associado!" << endl;
    }
    bool operator<(const UC &outra)const { return nome<outra.nome; }
};

//classe Docente: [1.8 val.]
bool addPraticas(UC *uc){ [3]
    if (uc->setDocenteDasPraticas(this)) return praticas.insert(uc);
    else {
        cout << "UC ja tem associado docente das praticas" << endl;
        return false;
    }
}
virtual void remTeoricas(UC *uc){ [1]
void remPraticas(UC *uc){ praticas.erase(uc); } [1]
string getNome()const{ return nome; } [1]
```

```

//classe Regente: [1.2 val.]
    bool addTeoricas(UC *uc){ [3]
        if (uc->setRegente(this)) return teoricas.insert(uc);
        else{
            cout << "UC ja tem regente" << endl;
            return false;
        }
    }
    void remTeoricas(UC *uc){ teoricas.erase(uc); } [1]

//classe DIC: [2.7 val.]
    bool remDocentesDeUC(string uc){ [3]
        bool res = false;
        UC *puc = findUC(uc);
        if (puc != NULL)
            return puc->remDocenteDasPraticas() || puc->remRegente();
        else { cout << "UC nao existente" << endl; return false; }
    }
    void mostrarDocentesDeUC(string uc){ [3]
        UC *puc = findUC(uc);
        if (puc != NULL)
            puc->mostrarDocentes();
        else cout << "UC nao existente" << endl;
    }
    ~DIC(){
        ColecaoHibrida<Docente *>::iterator it; [3]
        for (it = docentes.begin(); it != docentes.end(); it++)
            delete *it;
        docentes.clear();
    }
}

```

Engenharia Informática
Informática de Gestão

Época de Recurso – 8 de fevereiro de 2014

Notas importantes:

1. *O Grupo III é de resolução facultativa e destina-se a substituir as notas do trabalho prático e minitestes*
 2. *Duração da prova: 1h30 (Grupos I e II) + 1h (Grupo III)*
 3. *Comece por preencher a zona reservada à sua identificação na folha de exame;*
 4. *Resolva os três grupos em folhas de exame separadas.*
-

Grupo I
(8 valores)

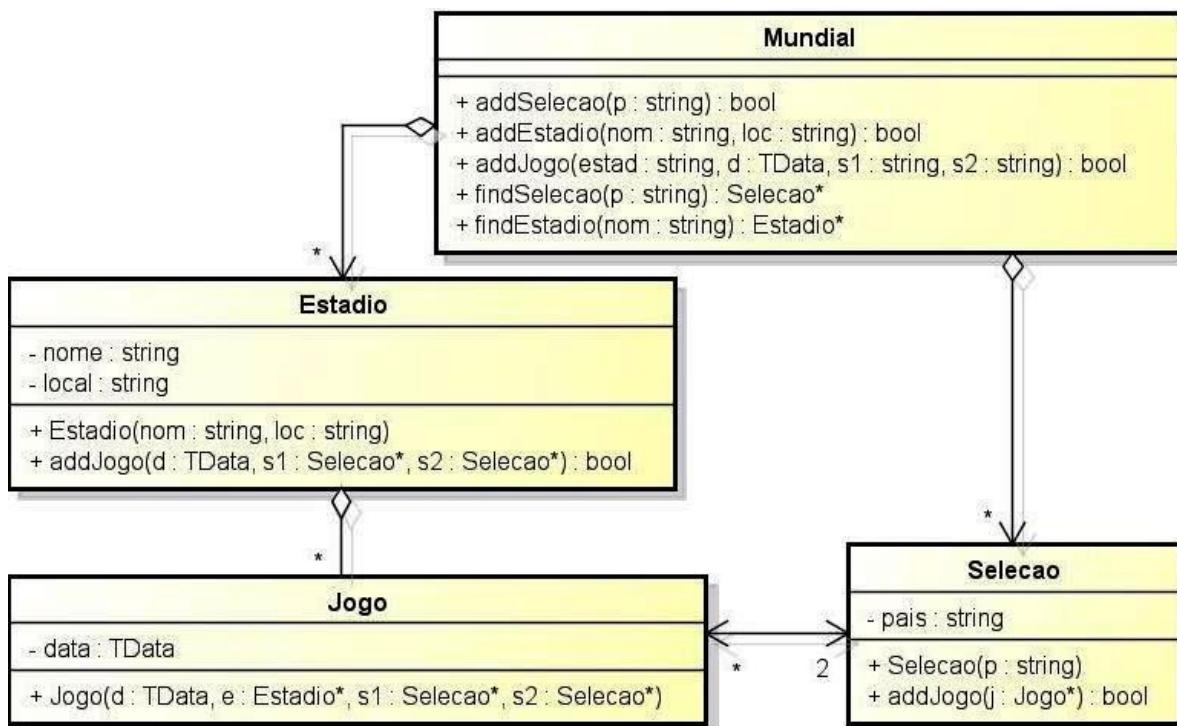
No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com a devida atenção, responda às seguintes questões.

- a) Identifique o tipo de relação existente entre *Instituto* e *Escola* e entre *Escola* e *ESTiG*. [0.6 val.]
- b) Diga quais são os métodos construtores que estarão presentes em cada uma das classes do problema. [1.0 val.]
- c) Nas classes do problema existem métodos que deveriam ser definidos como constantes? Se sim, diga quais. [0.6 val.]
- d) Apresente o resultado que será visualizado na saída standard com a execução do programa. [3.6 val.]
- e) O que seria visualizado se o método *print* da classe *Escola* não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [1.0 val.]
- f) E se fosse o método destrutor da classe *Escola* a não ser virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea d). [0.6 val.]
- g) Diga se as declarações *ESTiG V1[5]; ESA *V2[5];* são ou não válidas no problema considerado. Caso não sejam, introduza as alterações necessárias nas classes do problema para que as declarações passem a ser válidas. Diga, por fim, quantos construtores serão invocados em cada uma das declarações. [0.6 val.]



Estamos já a escassos meses do Mundial 2014, Campeonato do Mundo de futebol que mais uma vez contará com a presença de Portugal e que este ano terá o Brasil como país anfitrião.

Suponha que o incumbiram a si de desenvolver uma pequena aplicação que permita gerir informaticamente a distribuição dos jogos da competição pelos diferentes estádios disponibilizados para o evento. Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para a aplicação descrita.



- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e atributos descritos no diagrama. Não implemente quaisquer outros métodos ou atributos; apenas acrescente, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis. [8.5 val.]

NOTA 1: Admita que num mesmo estádio não podem ocorrer dois jogos no mesmo dia;

NOTA 2: Para o tipo data é usada a classe TData, estudada nas aulas da UC, e da qual se apresentam, no Anexo B deste enunciado, o protótipo de alguns métodos e operadores;

NOTA 3: As coleções deverão ser implementadas com base no template de classes Colecao ou ColecaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;

NOTA 4: Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;

NOTA 5: Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.

- b) Acrescente ao problema os métodos que permitem mostrar todas os jogos realizados num dado estádio, com a indicação da data e das duas seleções opositoras. [2.5 val.]

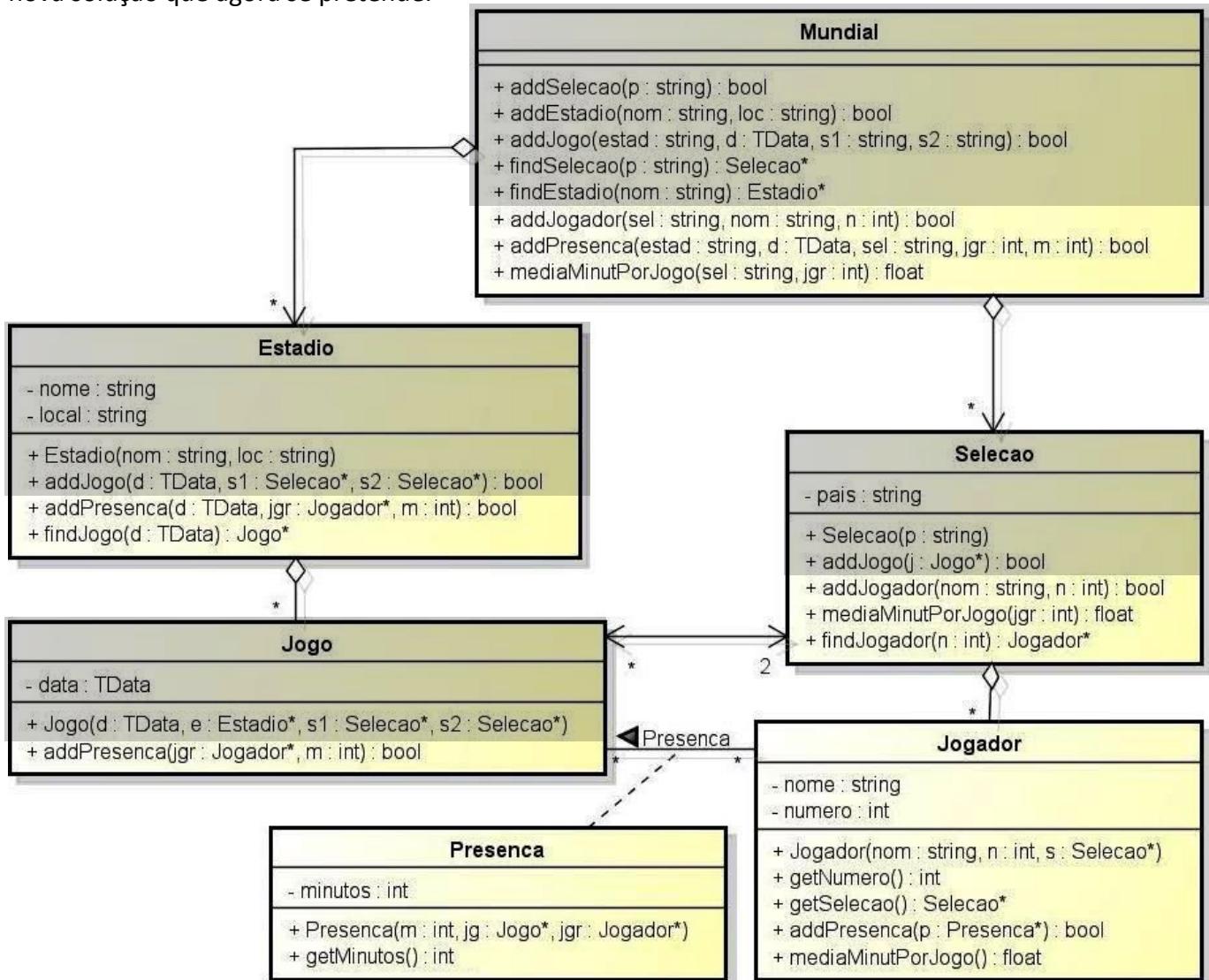
- c) Implemente um pequeno main que faça uso de todas as funcionalidades da aplicação. [1.0 val.]

Grupo III

(10 valores)

*NOTA: O problema que se segue é um exercício suplementar e facultativo que se destina a substituir a classificação obtida durante o período letivo (minitestes+trabalho). Com a sua resolução, ou tentativa de resolução, essa componente de avaliação deixará de contar para a época de recurso, e a sua nota final será: (G1+GII+GIII)*2/3.*

A ideia agora é acrescentar à aplicação a capacidade de registar, para cada jogador, o número de minutos que esteve em campo em cada um dos jogos em que participou. Como nova funcionalidade, a aplicação deve passar a permitir calcular, para um dado jogador, o valor médio do número de minutos que esteve em campo por jogo. Segue-se o diagrama de classes UML que descreve de forma precisa a nova solução que agora se pretende.



Este exercício destina-se a acrescentar novas funcionalidades ao problema que começou a implementar no grupo de questões anterior. Por isso, nas suas respostas deve ter sempre em conta o código que já implementou anteriormente para o problema.

- Defina em C++ as novas classes **Jogador** e **Presenca**, e acrescente às já existentes as novas funcionalidade do problema, respeitando integralmente os métodos e atributos descritos no diagrama. Não implemente quaisquer outros métodos ou atributos; apenas acrescente, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis. [9.0 val]
- Implemente um pequeno main que faça uso das novas funcionalidades da aplicação. [1.0 val.]

ANEXO A

```
#include <string>
#include <iostream>
using namespace std;

class Escola{
    string nome;
public:
    Escola(){cout<<"Criada escola sem nome!"<<endl;}
    Escola(const string &n):nome(n){cout<<"Criada a "<<nome<<. ";}
    virtual void print(){cout<<"A escola e' mesmo fixe!"<<endl;}
    virtual ~Escola(){cout<<nome<<" encerrada!"<<endl;}
};

class ESTiG: public Escola{
public:
    ESTiG():Escola("ESTiG"){cout<<"A melhor escola do pais!"<<endl;}
    void print(){cout<<"A ESTiG e' mesmo fixe!"<<endl;}
    ~ESTiG(){cout<<"Encerrada a melhor escola do pais!"<<endl;}
};

class ESA: public Escola{
public:
    ESA():Escola("ESA"){cout<<"A 2a melhor escola do pais!"<<endl;}
    void print(){cout<<"A ESA e' mesmo fixe!"<<endl;}
    ~ESA(){cout<<"Encerrada a 2a melhor escola do pais!"<<endl;}
};

class Instituto{
    Escola *esc;
public:
    Instituto(){cout<<"Novo instituto sem escolas"<<endl; esc=NULL;}
    Instituto(Escola *e){esc=e;}
    void print(){esc->print();}
    ~Instituto(){cout<<"Instituto encerrado"<<endl;}
};

void main(){
    ESTiG estig;
    Escola *esa=new ESA;
    Instituto i1(&estig), i2(esa);
    i1.print();
    i2.print();
    delete esa;
}
```

ANEXO B

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers{
public:
    bool operator()(const T &left, const T &right) const {
        return (*left < *right);
    }
};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Classe TData

```
class TData{
public:
    TData();
    TData(int d, int m, int a); // cria o objeto com a data d/m/a
    TData(const string & dat); // cria o objeto com a data expressa na string dat
                                //(string com uma data no formato "x/x/yyyy" ou "x-x-xx")
    TData(const char *dat); // cria o objeto c/ a data expressa na string tipo C dat
    string data_str() const; // devolve uma string com a data no formato "x/Mes/xx"
    bool operator<(const TData &outra) const;
    static TData hoje(); //devolve data corrente
    ...
};

ostream &operator<<(ostream &os, const TData &dat);
istream &operator>>(istream &is, TData &dat);
```

Engenharia Informática

Informática de Gestão

Época de Recurso – 8 de fevereiro de 2014

Grupo I

(8 valores)

- a) Identifique o tipo de relação existente entre *Instituto* e *Escola* e entre *Escola* e *ESTiG*. [0.6 val.]
Respetivamente, associação simples (Instituto associa Escola) e Herança (ESTiG deriva de Escola). (do ponto de vista conceptual, seria também aceitável considerar agregação em vez de associação simples)
- b) Diga quais são os métodos construtores que estarão presentes em cada uma das classes... [1.0 val.]
Tanto o construtor por defeito como o de cópia estão presentes em todas as classes; as classes Escola e Instituto dispõem ainda dum construtor de conversão cada.
- c) Nas classes do problema existem métodos que deveriam ser definidos constantes? Se ... [0. 6 val.]
Sim, o método `print()` de todas as classes.
- d) Apresente o resultado que será visualizado na saída standard com a execução do programa. [3.6 val.]
Criada a ESTiG. A melhor escola do pais!
Criada a ESA. A 2a melhor escola do pais!
A ESTiG e' mesmo fixe!
A ESA e' mesmo fixe!
Encerrada a 2a melhor escola do pais!
ESA encerrada!
Instituto encerrado
Instituto encerrado
Encerrada a melhor escola do pais!
ESTiG encerrada!
- e) O que seria visualizado se o método `print` da classe *Escola* não fosse virtual? Refira-se... [1.0 val.]
No lugar das mensagens
A ESTiG e' mesmo fixe!
A ESA e' mesmo fixe!
surgiria:
A escola e' mesmo fixe!
A escola e' mesmo fixe!
- f) E se fosse o método destrutor da classe *Escola* a não ser virtual? Refira-se apenas ... [0. 6 val.]
Não era mostrada a mensagem
Encerrada a 2a melhor escola do pais!
- g) Diga se as declarações `ESTiG V1[5];` `ESA *V2[5];` são ou não válidas no problema ... [0. 6 val.]
Ambas são válidas. Na primeira declaração são invocados 10 construtores; na segunda não é invocado qualquer construtor.

Grupo II
(12 valores)

a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [8.5 val]

```
class Mundial{
    Colecao<Estadio> estadios;
    Colecao<Selecao> selecoes;
public:
    bool addSelecao(string p){
        Selecao s(p);
        return selecoes.insert(p);
    }

    bool addEstadio(string nom, string loc){
        Estadio e(nom,loc);
        return estadios.insert(e);
    }

    bool addJogo(string estad, TData d, string s1, string s2){
        Estadio *pe=findEstadio(estad);
        if(pe!=NULL){
            Selecao *ps1=findSelecao(s1);
            if(ps1!=NULL){
                Selecao *ps2=findSelecao(s2);
                if(ps2!=NULL)
                    return pe->addJogo(d,ps1,ps2);
            }
        }
        return false;
    }

    Selecao *findSelecao(string p){
        Selecao s(p);
        return selecoes.find(s);
    }

    Estadio *findEstadio(string nom){
        Estadio e(nom, "");
        return estadios.find(e);
    }
};

class Selecao{
    string pais;
    Colecao<Jogo *> jogos;
public:
    Selecao(string p): pais(p){}
    bool addJogo(Jogo *j){return jogos.insert(j);}
    bool operator<(const Selecao &outra) const {return pais<outra.pais;}
};
```

```

class Estadio{
    string nome;
    string local;
    Colecao<Jogo> jogos;
public:
    Estadio(string nom, string loc):nome(nom),local(loc){}
    bool addJogo(TData d, Selecao *s1, Selecao *s2){
        Jogo jogo(d,this,s1,s2);
        jogos.insert(jogo);
        Jogo *pj=jogos.find(jogo);
        if (pj!=NULL)
            if (s1->addJogo(pj))
                if (s2->addJogo(pj)) return true;
        return false;
    }
    bool operator<(const Estadio &outro) const {return nome<outro.nome;}
};

class Jogo{
    TData data;
    Estadio *estadio;
    Selecao *opositor1, *opositor2;
public:
    Jogo(TData d, Estadio *e, Selecao *s1, Selecao *s2): data(d){
        estadio=e;
        opositor1=s1; opositor2=s2;
    }
    bool operator<(const Jogo &outro) const {return data<outro.data;}
};

```

b) Acrescente ao problema os métodos que permitam mostrar todas os jogos realizados ... [2.5 val.]

```

void Mundial::printJogosDeEstadio(string estad){
    Estadio *pe=findEstadio(estad);
    if(pe!=NULL) pe->printJogos();
    else cout<<"Estadio inexistente!"<<endl;
}

void Estadio::printJogos() const{
    cout<<"Jogos do estadio "<< nome<< ":"<<endl;
    Colecao<Jogo>::iterator it;
    for(it=jogos.begin(); it!=jogos.end(); it++)
        it->print();
}

void Jogo::print() const{
    cout<<data<< ":" " << opositor1->getPaís() << " x " << opositor2->getPaís()<<endl;
}

string Selecao::getPaís(){return pais;}

```

c) Implemente um pequeno main que faça uso de todas as funcionalidades da aplicação. [1.0 val.]

```
void main(){
    Mundial brasil2014;
    brasil2014.addEstadio("Maracana", "Rio de Janeiro");
    brasil2014.addSelecao("Portugal");
    brasil2014.addSelecao("Brasil");
    brasil2014.addJogo("Maracana", "13/07/14", "Brasil", "Portugal");
    brasil2014.printJogosDeEstadio("Maracana");
}
```

Grupo III (20 valores)

a) Defina em C++ as novas classes Jogador e Presenca, e acrescente às já existentes ... [9.0 val]

```
class Presenca{
    int minutos;
    Jogo *jogo;
    Jogador *jogador;
public:
    Presenca(int m, Jogo *jg, Jogador *jgr){
        minutos=m;
        jogo=jg; jogador=jgr;
    }
    int getMinutos(){return minutos;}
    bool Presenca::operator<(const Presenca &outra) const{
        if(jogador->getSelecao()->getPais() != outra.jogador->getSelecao()->getPais())
            return jogador->getSelecao()->getPais() < outra.jogador->getSelecao()->getPais();
        else return jogador->getNumero() < outra.jogador->getNumero();
    }
};

class Jogador{
    string nome;
    int numero;
    Selecao *selecao;
    Colecao<Presenca *> presencias;
public:
    Jogador(string nom, int n, Selecao *s): nome(nom){
        numero=n;
        selecao=s;
    }
    int getNumero(){return numero;}
    Selecao *getSelecao(){return selecao;}
    bool addPresenca(Presenca *p){return presencias.insert(p);}
    float mediaMinutPorJogo(){
        float m=0.0;
        Colecao<Presenca *>::iterator it;
        for(it=presencias.begin(); it!=presencias.end(); it++)
            m+=(*it)->getMinutos();
        m/=presencias.size();
        return m;
    }
    bool operator<(const Jogador &outro) const {return numero<outro.numero;}
};
```

```

class Selecao{
    string pais;
    Colecao<Jogo *> jogos;
    Colecao<Jogador> jogadores;
public:
    ...
    bool addJogador(string nom, int n){
        Jogador j(nom, n, this);
        return jogadores.insert(j);
    }
    float mediaMinutPorJogo(int jgr){
        Jogador *pj=findJogador(jgr);
        if(pj!=NULL) return pj->mediaMinutPorJogo();
        else return 0.0;
    }
    Jogador *findJogador(int n){
        Jogador j("",n,NULL);
        return jogadores.find(j);
    }
};

class Jogo{
    TData data;
    Estadio *estadio;
    Selecao *opositor1, *opositor2;
    Colecao<Presenca> presencias;
public:
    ...
    bool addPresencia(Jogador *jgr, int m){
        Presenca p(m,this,jgr);
        if(jgr->getSelecao()!=opositor1 && jgr->getSelecao()!=opositor2){
            cout<<"A Selecao desse jogador nao jogou esse encontro!"<<endl;
            return false;
        }else
            if (presencias.insert(p))
                return jgr->addPresencia(presencias.find(p));
            else return false;
    }
};

class Estadio{
    string nome; string local; Colecao<Jogo> jogos;
public:
    ...
    bool addPresencia(TData d, Jogador *jgr, int m){
        Jogo *pj=findJogo(d);
        if(pj!=NULL) return pj->addPresencia(jgr,m);
        else return false;
    }
    Jogo *findJogo(TData d){
        Jogo j(d,NULL,NULL,NULL);
        return jogos.find(j);
    }
};

```

```
class Mundial{
    Colecao<Estadio> estadios;
    Colecao<Selecao> selecoes;
public:
    ...
    bool addJogador(string sel, string nom, int n){
        Selecao *ps=findSelecao(sel);
        if(ps!=NULL) return ps->addJogador(nom, n);
        else {cout<<"Selecao inexistente!"<<endl; return false;}
    }
    bool addPresenca(string estad, TData d, string sel, int jgr, int m){
        Estadio *pe=findEstadio(estad);
        if(pe!=NULL){
            Selecao *ps=findSelecao(sel);
            if(ps!=NULL){
                Jogador *pj=ps->findJogador(jgr);
                if(pj!=NULL)
                    return pe->addPresenca(d,pj,m);
            }
        }
        return false;
    }
    float mediaMinutPorJogo(string sel, int jgr){
        Selecao *ps=findSelecao(sel);
        if(ps!=NULL) return ps->mediaMinutPorJogo(jgr);
        else return 0.0;
    }
};
```

b) Implemente um pequeno main que faça uso das novas funcionalidades da aplicação. [1.0 val.]

```
void main(){
    Mundial brasil2014;
    brasil2014.addEstadio("Maracana", "Rio de Janeiro");
    brasil2014.addSelecao("Portugal");
    brasil2014.addSelecao("Brasil");
    brasil2014.addJogo("Maracana", "13/07/14", "Brasil", "Portugal");
    brasil2014.printJogosDeEstadio("Maracana");
    brasil2014.addJogador("Portugal", "Cristiano Ronaldo", 7);
    brasil2014.addPresenca("Maracana", "13/07/14", "Portugal", 7, 85);
    cout<<"Media dos Minutos/Jogo de CR7: " <<
        brasil2014.mediaMinutPorJogo("Portugal", 7)<<endl;
}
```

Engenharia Informática
Informática de Gestão

Época de Recurso – 9 de fevereiro de 2013

Notas importantes:

1. *O Grupo III é de resolução facultativa e destina-se a substituir as notas do trabalho prático e minitestes*
 2. *Duração da prova: 1h30 (Grupos I e II) + 1h (Grupo III)*
 3. *Comece por preencher a zona reservada à sua identificação na folha de exame;*
 4. *Resolva os três grupos em folhas de exame separadas.*
-

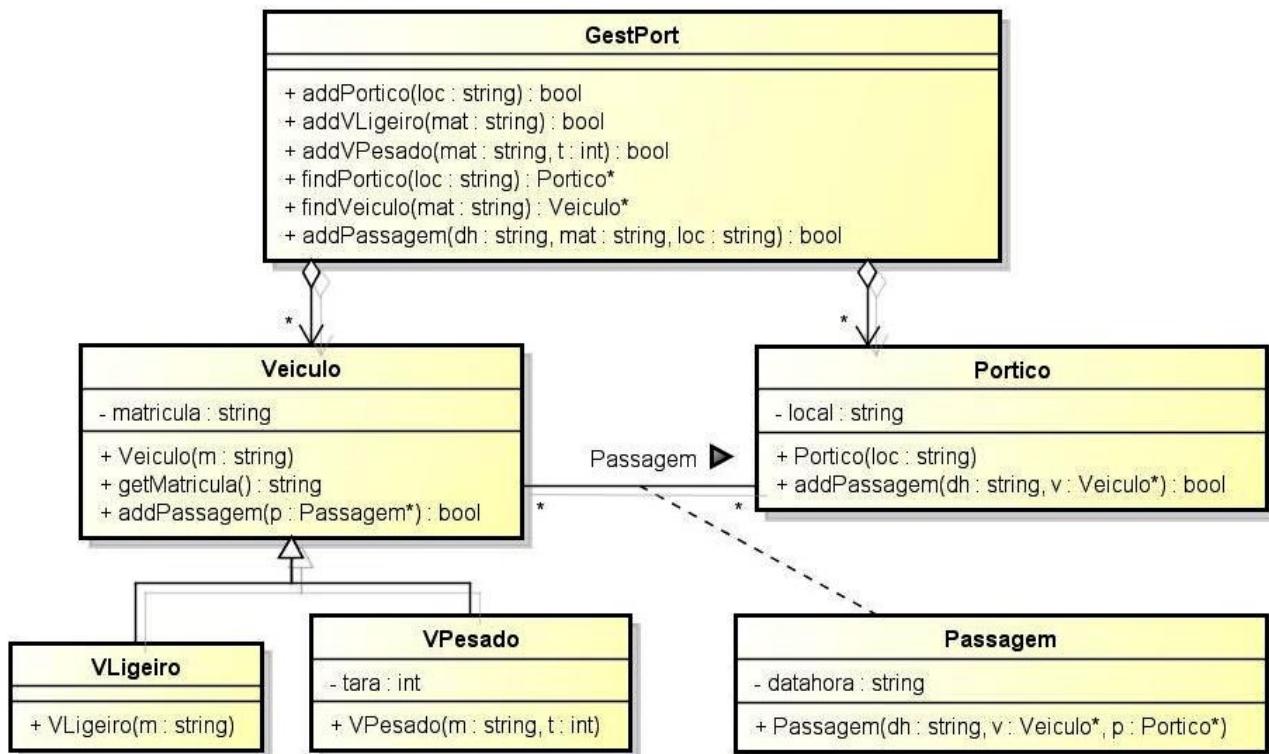
Grupo I
(8 valores)

No Anexo A do presente enunciado encontra-se o código de um pequeno programa em C++. Depois de o analisar com a devida atenção, responda às seguintes questões.

- a) Diga quais são os métodos construtores que estarão presentes em cada uma das classes do problema. [1.4 val.]
- b) Apresente o resultado que será visualizado na saída standard com a execução do programa. [2.7]
- c) O que seria visualizado se o método *get* da classe *I* não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [0.5 val.]
- d) O que seria visualizado se o método *print* da classe *I* não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea b). [0.5 val.]
- e) E o que seria visualizado se fosse o método destrutor da classe *I* a não ser virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea b). [0.5 val.]
- f) Diga se as declarações *I V1[10]; I *V2[20];* são ou não válidas no problema considerado. Caso não sejam, introduza as alterações necessárias nas classes do problema para que as declarações passem a ser válidas. Diga, por fim, quantos construtores serão invocados com as referidas declarações. [1.2 val.]
- g) Faça as alterações que julgue necessárias para que a função *main* do problema considerado possa finalizar com a seguinte instrução: *if (a>0) cout<<a;* [1.2 val.]

Grupo II
(12 valores)

Suponha que a *GestPort, Lda* é a empresa responsável pelo sistema eletrónico de cobrança de portagens nas antigas SCUTs. Pretende-se uma pequena aplicação informática que lhe permita registar a data e a hora (apenas uma string) da passagem de veículos pelos pórticos de cobrança eletrónica colocados estratégicamente ao longo das inúmeras vias concessionadas. Portanto, nesta primeira fase, o sistema ainda não deve assegurar a gestão de cobranças. Segue-se o diagrama de classes UML que descreve de forma precisa a solução que se pretende para a aplicação descrita.



- a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e atributos descritos no diagrama. Não implemente quaisquer outros métodos ou atributos; apenas acrescente, nas classes em que se justifique, o operador que permita que os objetos sejam colecionáveis. [11 val]

NOTA 1: Incluindo as autoestradas duas ou mais vias de trânsito, admite que num mesmo pórtico poderão ser registados veículos que passem na mesma data e hora;

NOTA 2: As coleções deverão ser implementadas com base no template de classes Colecao ou ColecaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;

NOTA 3: Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;

NOTA 4: Nesta e nas restantes alíneas, não precisa de indicar as diretivas #include.

- b) Implemente um pequeno main que faça uso de todas as funcionalidades da aplicação. [1 val.]

Grupo III
(20 valores)

NOTA: O problema que se segue é um exercício suplementar e facultativo que se destina a substituir a classificação obtida durante o período letivo (miniteste+trabalho). Com a sua resolução, ou tentativa de resolução, essa componente de avaliação deixa de contar para a época de recurso.

As alíneas que se seguem destinam-se a acrescentar novas funcionalidades ao problema que começou a implementar no grupo de questões anterior. Por isso, nas suas respostas deve ter sempre em conta o código que já implementou anteriormente para o problema.

- a) Implemente o(s) método(s) necessário(s) que permita(m) listar todos os veículos registados no sistema, mostrando, para cada um deles, a matrícula, se se trata de um veículo ligeiro ou pesado e, tratando-se deste segundo caso, a respetiva tara. [4 val.]
- b) Implemente o(s) método(s) necessário(s) que permita(m) listar todas as passagens registadas num dado pórtico, mostrando, para cada uma delas, a matrícula do veículo e a data e a hora da sua passagem. [4 val.]
- c) Para que a aplicação se revele verdadeiramente útil, vamos agora acrescentar-lhe os mecanismos que lhe permitam gerir a cobrança de portagens. Para o efeito, nesta alínea, acrescente-lhe o(s) atributo(s) e método(s) que permitam definir uma tarifa unitária específica para cada pórtico (tarifa que será posteriormente usada para o cálculo da taxa da portagem de cada veículo que passe no pórtico). [3 val.]
- d) Implemente o(s) método(s) necessário(s) para calcular o valor total das portagens faturado pela empresa GestPort, admitindo o seguinte: tratando-se de um veículo ligeiro, a taxa de portagem tem sempre o valor da tarifa unitária; caso se trate de um pesado, a taxa a cobrar será o dobro da tarifa unitária mais o valor dessa tarifa por cada 5 toneladas da sua tara (exemplo: sendo a tarifa 3€ e a tara 21000Kg, a taxa será $2 \cdot 3\text{€} + 4 \cdot 3\text{€} = 18\text{€}$). [5 val.]
- e) Implemente o método destrutor, com o comportamento adequado, para a classe GestPort. [1 val.]
- f) Implemente o(s) método(s) necessário(s) para remover do sistema um dado veículo. A remoção só deverá ser levada a cabo se o veículo em causa ainda não tiver passado em nenhum pórtico. [2 val.]
- g) Escreva uma função main() que faça uso de todas as novas funcionalidades do problema. [1 val.]

ANEXO A

```
#include<iostream>
using namespace std;

class I{
public:
    int virtual get()const{return 0;}
    void virtual print()const{cout<<"I::print"<<endl;}
    virtual ~I(){}
};

class A: public I{
    int val;
    static int n;
public:
    A(){val=++n; cout<<"novo A "<<val<<endl;}
    int get()const{return val;}
    void print()const{cout<<"A::print "<<val<<endl;}
    ~A(){cout<<"A "<<val<<" eliminado"<<endl;}
};

int A::n=0;

class B: public I{
    int val;
    A a[2];
public:
    B(): val(0){cout<< "novo B "<<endl;}
    B(A &x):val(x.get()){cout<< "A => B "<<val<<endl;}
    void print()const{
        cout<<"B::print "<<val<<endl;
        a[0].print();
        a[1].print();
    }
    ~B(){cout<<"B "<<val<<" eliminado"<<endl;}
};

void main(){
    cout<<"<1>"<<endl;
    B b1;
    cout<<"<2>"<<endl;
    I *p=&b1;
    cout<<"<3>"<<endl;
    p->print();
    cout<<"<4>"<<endl;
    A a;
    cout<<"<5>"<<endl;
    B b2(a);
    cout<<"<6>"<<endl;
}
```

Template Colecao

```
#include<set>
using namespace std;

template<class K>
class Colecao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Template ColecaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers
{
public:
    bool operator()(const T &left, const T &right) const
    {
        return (*left < *right);
    }
};

template<class K>
class ColecaoHibrida: public set<K, less_pointers<K>>
{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Grupo I

- a) Diga quais são os métodos construtores que estarão presentes em cada uma das classes... [1.4 val.]

Na classe I:

Construtor por defeito

Construtor de cópia

Na classe A:

Construtor por defeito

Construtor de cópia

Na classe B:

Construtor por defeito

Construtor de cópia

Construtor de conversão B(A &)

- b) Apresente o resultado que será visualizado na saída standard com a execução do programa. [2.7]

```
<1>
novo A 1
novo A 2
novo B
<2>
<3>
B::print 0
A::print 1
A::print 2
<4>
novo A 3
<5>
novo A 4
novo A 5
A => B 3
<6>
B 3 eliminado
A 5 eliminado
A 4 eliminado
A 3 eliminado
B 0 eliminado
A 2 eliminado
A 1 eliminado
```

- c) O que seria visualizado se o método *get* da classe *I* não fosse virtual? Refira-se apenas... [0.5 val.]

Seria visualizado exatamente o mesmo.

- d) O que seria visualizado se o método *print* da classe *I* não fosse virtual? Refira-se apenas... [0.5 val.]

```
...
<3>
I::print
<4>
...
...
```

- e) E o que seria visualizado se o método destrutor da classe *I* não fosse virtual? Refira-se... [0.5 val.]

Seria visualizado exatamente o mesmo que em b).

f) Diga se as declarações `I V1[10]; I *V2[20];` são ou não válidas no problema considerado... [1.2 val.]
São ambas válidas;
São invocados 10 construtores (na instanciação dos 10 objetos do vetor V1).

g) Faça as alterações que julgue necessárias para que a função `main ... if (a>0) cout<<a;` [1.2 val.]

```
bool A::operator>(int v) const{return val>v;}

ostream &operator<<(ostream &o, const A &a){
    o<<a.get();
    return o;
}
```

GRUPO II

a)

```
class Veiculo{
    string matricula;
    Colecao<Passagem *> passagens;
public:
    Veiculo(string m): matricula(m){}

    string getMatricula() const {return matricula; }

    bool addPassagem(Passagem *p){return passagens.insert(p);}

    bool operator<(const Veiculo &outro) const {return matricula<outro.matricula;}
};

class VLigeiro: public Veiculo{
public:
    VLigeiro(string m): Veiculo(m){}
};

class VPesado: public Veiculo{
    int tara;
public:
    VPesado(string m, int t): Veiculo(m){tara=t;}
};

class Passagem{
    string datahora;
    Veiculo* veiculo;
    Portico* portico;
public:
    Passagem(string dh, Veiculo* v, Portico* p): datahora(dh){
        veiculo=v;
        portico=p;
    }

    bool operator<(const Passagem &outra) const {
        if (datahora!=outra.datahora) return datahora<outra.datahora;
        else return veiculo->getMatricula()<outra.veiculo->getMatricula();
    }
};
```

```

class Portico{
    string local;
    Colecao<Passagem> passagens;
public:
    Portico(string loc){local=loc;}
    bool addPassagem(string dh, Veiculo *v){
        Passagem p(dh,v,this);
        if (passagens.insert(p)) return v->addPassagem(passagens.find(p));
        else return false;
    }

    bool operator<(const Portico &outro) const {return local<outro.local;}
};

class GestPort{
    ColecaoHibrida<Veiculo*> veiculos;
    Colecao<Portico> porticos;
public:
    bool addPortico(string loc){
        Portico p(loc);
        return porticos.insert(p);
    }

    bool addVLigeiro(string mat){
        Veiculo *v = new VLigeiro(mat);
        return veiculos.insert(v);
    }

    bool addVPesado(string mat, int t){
        Veiculo *v = new VPesado(mat,t);
        return veiculos.insert(v);
    }

    Portico *findPortico(string loc){
        Portico p(loc);
        return porticos.find(p);
    }

    Veiculo *findVeiculo(string mat){
        Veiculo v(mat);
        return veiculos.find(&v);
    }

    bool addPassagem(string dh, string mat, string loc){
        Veiculo *v=findVeiculo(mat);
        if(v!=NULL){
            Portico *p=findPortico(loc);
            if(p!=NULL) return p->addPassagem(dh,v);
            else return false;
        }else return false;
    }
};

```

```

b)
void main(){
    GestPort a;
    a.addPortico("A24 Km 53");
    a.addVLigeiro("11-22-AA");
    a.addVPesado("33-44-BB", 21000);
    a.addPassagem("1/2/12 19:00", "33-44-BB", "A24 Km 53");
}

```

GRUPO III

a)

```

void GestPort::printVeiculos()const{
    ColecaoHibrida<Veiculo *>::iterator it;
    for(it=veiculos.begin(); it!=veiculos.end(); it++)
        (*it)->print();
}

virtual void Veiculo::print()const=0;

void VLigeiro::print()const{ cout<<"Veiculo Ligeiro "<< getMatricula();}

void VPesado::print()const{
    cout<<"Veiculo Pesado "<< getMatricula()<<" com tara "<< tara << endl;
}

```

b)

```

void GestPort::printPassagensPortico(string loc){
    Portico *p=findPortico(loc);
    if(p!=NULL) p->printPassagens();
}

void Portico::printPassagens()const{
    cout<<"Portico "<< local<<":\n";
    Colecao<Passagem>::iterator it;
    for(it=passagens.begin(); it!=passagens.end(); it++)
        it->print();
}

void Passagem::print()const{
    cout<<"Veiculo "<< veiculo->getMatricula()<< " passou a "<<datahora<<endl;
}

```

c)

```

class Portico{
    float tarifa;
    ...
public:
    ...
    void setTarifa(float val){tarifa=val;}
};

bool GestPort::novaTarifa(float val, string loc){
    Portico *p=findPortico(loc);
    if(p!=NULL) {p->setTarifa(val); return true;}
    else return false;
}

```

d)

```

float GestPort::totalFaturado()const{
    float tot=0;
    Colecao<Portico>::iterator it;
    for(it=porticos.begin(); it!=porticos.end(); it++)
        tot+=it->totalFaturado();
    return tot;
}

float Portico::totalFaturado()const{
    float tot=0;
    Colecao<Passagem>::iterator it;
    for(it=passagens.begin(); it!=passagens.end(); it++)
        tot+=it->numTarifas()*tarifa;
    return tot;
}

int Passagem::numTarifas()const{ return veiculo->numTarifas();}

virtual int Veiculo::numTarifas()const=0;

int VLigeiro::numTarifas()const{return 1;}

int VPesado::numTarifas()const{return 2 + tara/5000;}

```

e)

```

GestPort::~GestPort (){
    ColecaoHibrida<Veiculo *>::iterator it;
    for(it=veiculos.begin(); it!=veiculos.end(); it++) delete *it;
    veiculos.clear();
}

```

f)

```

bool GestPort::remVeiculo(string mat){
    Veiculo *v=findVeiculo(mat);
    if(v!=NULL && v->numPassagens()==0){
        veiculos.erase(v);
        delete v;
        return true;
    }else {cout<<"Remocao mal sucedida\n"; return false;}
}

int Veiculo::numPassagens()const {return passagens.size();}

```

g)

```

void main(){
    GestPort a;
    a.addPortico("A24 Km 53");
    a.addVLigeiro("11-22-AA");
    a.addVPesado("33-44-BB", 21000);
    a.addPassagem("1/2/12 19:00", "33-44-BB", "A24 Km 53");
    a.printVeiculos();
    a.printPassagensPortico("A24 Km 53");
    a.novaTarifa(3,"A24 Km 53");
    cout<<"Total faturado: "<<a.totalFaturado()<<endl;
    a.remVeiculo("11-22-AA");
}

```

Departamento de Informática e Comunicações

Engenharia Informática / Informática de Gestão

2º Ano

Época de Recurso

Programação Orientada por Objectos
1º Semestre

Data: 11/02/2012
Duração: 3h

Resolva os 3 grupos em folhas de exame separadas

1h30 (Grupos I e II) + 1h30 (Grupo III - opcional)

Grupo I
(10 valores)

No Anexo A do presente enunciado encontra-se o código de um programa em C++ que implementa um modelo muito simples de contas bancárias, com apenas algumas funcionalidades elementares. Depois de o analisar com a devida atenção, responda às seguintes questões.

- a) Alguma das classes do problema é abstracta? Se sim, diga qual e por que motivo a considera abstracta. [0.7 val.]
- b) Diga quais os métodos construtores de que dispõe cada uma das classes (os explícitos e os implícitos). [1.1 val.]
- c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [3.2]
- d) E o que seria visualizado se o método destrutor da classe Conta não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [1 val.]
- e) Como se percebe, na implementação sugerida, qualquer conta a Prazo que seja criada pode ter associada uma taxa de juro diferente das restantes. Que tipo de alteração proporia à solução apresentada, de modo a garantir que todas as contas a Prazo que viesssem a ser criadas passassem a ter a mesma taxa de juro. Explique, não implemente. [1 val.]
- f) Considerando as seguintes instanciações: CPespecial c1, c2; diga, justificando, se cada uma das instruções c1=c2; e c1++; é ou não válida no problema considerado. [1 val.]
- g) Considere agora o código substituto para a função main() que a seguir se apresenta. Identifique dois erros presentes nesse código e corrija-os efectuando apenas alterações nas classes do problema. [2 v.]

```
void main(){
    CPrazo contas[10];
    for(int i=0; i<10; i++) contas[i]+=10.0; //depósitos de 10 euros
}
```

Grupo II
(10 valores)

Um importante aeroporto necessita de uma pequena aplicação que lhe permita monitorizar, a qualquer momento, todas as aeronaves que se encontram conectadas às suas portas de embarque/desembarque (*gates*). Considere que a cada uma das *gates* apenas pode estar conectada, num dado momento, uma única aeronave (ou nenhuma), para embarque ou desembarque dos respectivos passageiros ou mercadorias, caso se trate de uma aeronave de passageiros ou de carga, respectivamente. As *gates* são identificadas por um código inteiro e, quando conectadas, deverão dar ainda a indicação se estão a operar como porta de embarque ou de desembarque para as respectivas aeronaves. Cada aeronave, sendo também identificada por um código inteiro, é ainda caracterizada pela sua lotação, no caso de se tratar de uma aeronave de passageiros, ou então pela respectiva tara, caso se trate de uma aeronave de carga.

- a) Apresente, através de um diagrama de classes em UML, a solução por si idealizada para sustentar a aplicação descrita, indicando para cada classe apenas os atributos estritamente necessários, um método construtor e, nas classes em que se justifique, o operador que permita que os objectos sejam colecionáveis. [3 val.]

Não indique quaisquer outros métodos.

NOTA 1: No momento em que se regista uma nova aeronave no sistema, a mesma não deve ficar logo conectada a qualquer gate;

NOTA 2: No momento em que se regista uma nova gate no sistema, também ela não deve ficar conectada a qualquer aeronave;

- b) Codifique em C++ a sua solução, tal como a descreveu no diagrama da alínea anterior. [3v.]

NOTA 3: As colecções deverão ser implementadas com base no template de classes Coleccao ou ColeccaoHibrida que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no Anexo B deste enunciado;

NOTA 4: Defina os métodos no momento em que está a declarar as classes (não separe a declaração da implementação) e considere que todo o código reside num único ficheiro;

NOTA 5: Nesta e nas restantes alíneas, não precisa de indicar as directivas #include.

- c) Implemente os métodos necessários para adicionar à aplicação uma nova *gate*, uma nova aeronave de passageiros e uma nova aeronave de carga. [1.5 val.]

- d) Implemente o(s) método(s) necessário(s) para conectar uma aeronave a uma *gate*. [2.5val.]

Grupo III

(20 valores)

NOTA: O problema que se segue é um exercício suplementar e facultativo que se destina a substituir a classificação obtida durante o período lectivo (miniteste+trabalho). Com a sua resolução, ou tentativa de resolução, essa componente de avaliação deixa de contar para a época de recurso.

As alíneas que se seguem destinam-se a acrescentar novas funcionalidades ao problema que começou a implementar no grupo de questões anterior. Por isso, nas suas repostas deve ter sempre em conta o código que já implementou anteriormente para o problema.

- a) Implemente o(s) método(s) necessário(s) que permita(m) verificar se uma dada *gate* se encontra conectada (ou seja, se existe alguma aeronave conectada a essa *gate*). [2.5 val.]
- b) Implemente o(s) método(s) necessário(s) para desconectar uma dada *gate*. [2.5 val.]
- c) Implemente o(s) método(s) necessário(s) para remover do sistema uma dada *gate*. [2 val.]
- d) Implemente o(s) método(s) necessário(s) que permita(m) listar todas as aeronaves registadas, mostrando, para cada aeronave, o seu identificador, se se encontra conectada, e o valor da tara ou da lotação. [5 val.]
- e) Implemente o(s) método(s) necessário(s) que permita(m) listar as *gates* que se encontram conectadas, mostrando, para cada *gate*, o seu identificador, se está a ser usada para embarque ou desembarque, e os dados relativos à aeronave a que se encontra conectada. [3.5 val.]
- f) Implemente o método destrutor, com o comportamento adequado, para a classe Aeroporto. [2 val.]
- g) Escreva uma função main() que faça uso de todas as funcionalidades do problema. [2.5 val.]

```

#include<iostream>
#include<string>
using namespace std;

class Conta{
protected:
    double saldo;
public:
    Conta(): saldo(0.0){
        cout << "Abertura de Conta ";
    }
    Conta(double s): saldo(s){
        cout << endl << "Abertura com deposito inicial " << saldo;
    }
    virtual string getTipo()const =0;
    virtual ~Conta(){cout << " fechada com saldo " << saldo << endl;}
};

class CPrazo: public Conta{
    double txJuro;
public:
    CPrazo(double tx, double dep=0.0): Conta(dep){
        txJuro=tx;
        cout << " a Prazo ";
    }
    string getTipo()const {return " a Prazo ";}
    ~CPrazo(){cout << "Conta com taxa " << txJuro;}
};

class CPEspecial: public CPrazo{
public:
    CPEspecial(): CPrazo(1.5){
        saldo+=100.0;
        cout << "Especial 100" << endl;
    }
    CPEspecial(double tx): CPrazo(tx){
        saldo+=200.0;
        cout << "Especial 200";
    }
    string getTipo()const {return "Especial";}
    ~CPEspecial(){cout << getTipo();}
};

void main(){
    CPrazo *contas[3];
    contas[0]=new CPEspecial(3.5);
    contas[1]=new CPrazo(2.5, 500.0);
    CPEspecial c;
    contas[2]=&c;
    delete contas[0];
    delete contas[1];
}

```

Template Coleccao

```
#include<set>
using namespace std;

template<class K>
class Coleccao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Template ColeccaoHibrida

```
#include<set>
using namespace std;

template<class T>
class less_pointers
{
public:
    bool operator()(const T &left, const T &right) const
    {
        return (*left < *right);
    }
};

template<class K>
class ColeccaoHibrida: public set<K, less_pointers<K>>
{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};

};
```

Grupo I

a) Alguma das classes do problema é abstracta? Se sim, diga qual e por que motivo a considera. [0.7 val.]
A classe Conta, por ter um método virtual puro.

b) Diga de que métodos construtores dispõe cada uma das classes. [1.1 val.]

Conta: por defeito; de cópia; de conversão Conta(double s);

CPrazo: de cópia; CPrazo(double tx, double dep=0.0);

CPEspecial: por defeito; de cópia; de conversão CPEspecial(double tx);

c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [3.2]

Abertura com deposito inicial 0 a Prazo Especial 200

Abertura com deposito inicial 500 a Prazo

Abertura com deposito inicial 0 a Prazo Especial 100

Especial Conta com taxa 3.5 fechada com saldo 200

Conta com taxa 2.5 fechada com saldo 500

Especial Conta com taxa 1.5 fechada com saldo 100

d) E o que seria visualizado se o método destrutor da classe Conta não fosse virtual? [1 val.]

Especial Conta com taxa 3.5 fechada com saldo 200

e) Que tipo de alteração proporia à solução apresentada de modo a garantir ... a mesma taxa de juro. [1 val.]

O atributo txJuro passaria a estático na classe CPrazo

f) diga ... se cada uma das instruções c1=c2; e c1++; é ou não válida no problema considerado. [1 val.]

c1=c2; válida - está a ser invocado o operador afectação que é gerado de forma automática;
c1++; inválida - a classe CPEspecial não dispõe do operador incremento

g) Identifique dois erros presentes nesse código e corrija-os. [2 v.]

CPrazo contas[10]; -> a classe CPrazo não tem construtor por defeito

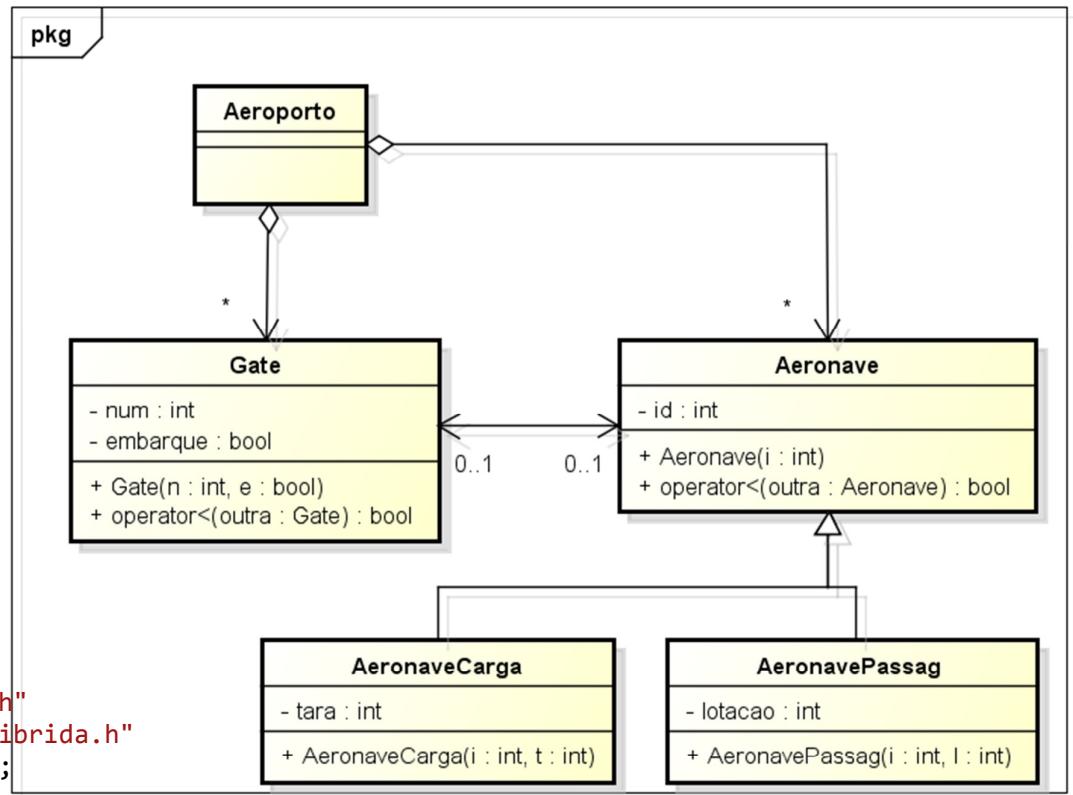
CPrazo::CPrazo(){txJuro=0.0;}

contas[i]+=10; -> a classe CPrazo não tem o operador +=

void operator+=(double val){saldo+=val;}

GRUPO II

a)



powered by astah®

```
#include <iostream>
#include "Coleccao.h"
#include "ColeccaoHibrida.h"
using namespace std;

class Aeroporto{
    Coleccao<Gate> gates;
    ColeccaoHibrida<Aeronave*> aeronaves;
};

class Gate{
    int num;
    bool embarque;
    Aeronave *aeronave;
public:
    Gate(int n): num(n){aeronave=NULL;}
    bool operator<(const Gate &outra) const {return num<outra.num;}
};

class Aeronave{
    int id;
    Gate *gate;
public:
    Aeronave(int i){id=i; gate=NULL;}
    virtual bool operator<(const Aeronave &outra) const {return id<outra.id;}
};

class AeronavePassag: public Aeronave{
    int lotacao;
public:
    AeronavePassag(int i, int lot): Aeronave(i){lotacao=lot;}
};

class AeronaveCarga: public Aeronave{
    int tara;
public:
    AeronaveCarga(int i, int tr): Aeronave(i){tara=tr;}
};
```

c)

```
bool Aeroporto::addGate(int n){
    Gate g(n);
    return gates.insert(g);
}
bool Aeroporto::addAeronavePassag(int i, int lot){
    Aeronave* pa= new AeronavePassag(i, lot);
    return aeronaves.insert(pa);
}
bool Aeroporto::addAeronaveCarga(int i, int tr){
    Aeronave* pa= new AeronaveCarga(i, tr);
    return aeronaves.insert(pa);
}
```

d)

```
bool conectarGateAeronave(int ng, int na, bool en){
    Gate *g=findGate(ng);
    if(g!=NULL && g->getAeronave()==NULL){
        Aeronave *a=findAeronave(na);
        if(a!=NULL && a->getGate()==NULL){
            g->setAeronave(a);
            g->setEmbarque(en);
            a->setGate(g);
            return true;
        }else return false;
    }else return false;
}

Gate *Aeroporto::findGate(int i){
    Gate g(i);
    return gates.find(g);
}
Aeronave *Aeroporto::findAeronave(int i){
    Aeronave a(i);
    return aeronaves.find(&a);
}

void Gate::setEmbarque(bool en){ embarque=en; }

Aeronave *Gate::getAeronave(){ return aeronave; }
void Gate::setAeronave(Aeronave *a){ aeronave=a; }

Gate *Aeronave::getGate(){ return gate; }
void Aeronave::setGate(Gate *g){ gate=g; }
```

GRUPO III

a)

```
bool Aeroporto::gateConecada(int ng){  
    Gate *g=findGate(ng);  
    if(g!=NULL) return g->conectada();  
    else return false;  
}  
  
bool Gate::conectada()const{return aeronave!=NULL;}
```

b)

```
bool Aeroporto::desconectar(int ng){  
    Gate *g=findGate(ng);  
    if(g!=NULL && g->conectada()){  
        g->getAeronave()->setGate(NULL);  
        g->setAeronave(NULL);  
        return true;  
    }else return false;  
}
```

c)

```
void Aeroporto::remGate(int ng){  
    if(gateConecada(ng)) desconectar(ng);  
    Gate g(ng);  
    gates.erase(g);  
}
```

d)

```
void Aeroporto::listarAeronaves(){  
    cout<<"Aeronaves:"<<endl;  
    ColeccaoHibrida<Aeronave*>::iterator it;  
    for(it=aeronaves.begin(); it!=aeronaves.end(); it++)  
        (*it)->print();  
}  
  
virtual void Aeronave::print()const{  
    cout<<"Num "<<id;  
    if(gate!=NULL) cout<<" conectada";  
}  
  
void AeronavePassag::print()const{  
    Aeronave::print();  
    cout<<": Aeronave de passageiros com lotacao "<<lotacao<<endl;  
}  
  
void AeronaveCarga::print()const{  
    Aeronave::print();  
    cout<<": Aeronave de carga de tara "<<tara<<endl;  
}
```

```

e)
void Aeroporto::listarGatesConectadas(){
    cout<<"Gates conectadas:"<<endl;
    Coleccao<Gate>::iterator it;
    for(it=gates.begin(); it!=gates.end(); it++)
        if(it->conectada()) it->print();
}

void Gate::print()const{
    cout<<"Gate "<<num;
    if(!conectada()) cout<<" desconectada"<<endl;
    else{
        cout<<(embarque?" Embarque":" Desembarque")<<" da aeronave : "<<endl<<"\t";
        aeronave->print();
    }
};

f)
Aeroporto::~Aeroporto(){
    ColeccaoHibrida<Aeronave*>::iterator it;
    for(it=aeronaves.begin(); it!=aeronaves.end(); it++)
        delete *it;
}

g)
void main(){
    Aeroporto ap;
    ap.addAeronavePassag(1, 100);
    ap.addAeronaveCarga(2, 25000);
    ap.addGate(5);
    ap.addGate(7);
    ap.conectarGateAeronave(5,1,true);
    ap.conectarGateAeronave(7,2,false);
    ap.listarAeronaves();
    ap.listarGatesConectadas();
    ap.desconectar(7);
    ap.remGate(7);
}

```

Departamento de Informática e Comunicações

Engenharia Informática / Informática de Gestão
2º Ano
Época Recurso

Programação Orientada por Objectos
1º Semestre

Data: 12/02/2011
Duração: 2h00

resolva os 2 grupos em folhas de exame separadas

Grupo I
(12 valores)

Uma empresa transportadora de grande dimensão necessita de uma pequena aplicação que lhe permita facilmente saber, a qualquer momento, que motorista é que conduz cada uma das viaturas que se encontram em trânsito e qual o seu número de telemóvel (basta que a viatura esteja afecta a um motorista para que se considere que a mesma se encontra em trânsito). A empresa dispõe de uma frota considerável de viaturas, quer de passageiros, quer de mercadorias, sendo as primeiras caracterizadas pela matrícula e lotação, e as segundas pela matrícula e peso bruto, e derivado do estrangulamento financeiro que a afecta, todas as viaturas circulam com um único motorista.

Repare que, ainda que todos os motoristas estejam habilitados a conduzir qualquer viatura, num dado momento só uma viatura pode estar afecta a um motorista (teria que ser muito habilidoso para conseguir conduzir várias ao mesmo tempo...).

- a) Apresente, através de um diagrama de classes em UML, a solução por si idealizada para sustentar a aplicação descrita, indicando para cada classe apenas os atributos estritamente necessários, um método construtor e, nas classes em que se justifique, o operador que permita que os objectos sejam colecciónáveis. [3 val.]
Não indique quaisquer outros métodos.

- b) Codifique em C++ a sua solução, tal como a descreveu no diagrama da alínea anterior. [2.5 val.]

NOTA: As colecções deverão ser implementadas com base no template de classes *Coleccao* ou *ColeccaoHibrida* que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no final do enunciado.

- c) Implemente os métodos necessários para adicionar (registar) à aplicação um novo motorista e um novo veículo de passageiros. [1 val.]

- d) Implemente o(s) método(s) necessário(s) para afectar uma viatura a um motorista. [2.7 val.]

- e) Implemente o(s) método(s) que permita(m) obter o número de telemóvel do motorista que conduza uma viatura com uma dada matrícula. [1.4 val.]

- f) Implemente o(s) método(s) necessário(s) para escrever na saída standard uma listagem das viaturas em circulação com indicação das matrículas e do nome dos respectivos condutores. [1.4 val.]

Grupo II
(8 valores)

O programa que se segue implementa um sistema de gestão (muito simples) de um prédio habitacional formado por habitações e garagens.

```

class Fraccao{
protected:
    char id;
    int permilagem;
public:
    Fraccao(char i, int p): id(i), permilagem(p){
        cout << "Criada Fraccao " << id;
    }
    virtual void mostrar(){
        cout << id << " com permilagem " << permilagem << endl;
    }
    virtual ~Fraccao(){}
};

class Habitacao: public Fraccao{
public:
    Habitacao(char i): Fraccao(i, 40){
        cout << " do tipo Habitacao" << endl;
    }
    void mostrar(){
        cout << "Habitacao ";
        Fraccao::mostrar();
    }
    ~Habitacao(){ cout << "Removida Habitacao " << id << endl;}
};

class Garagem: public Fraccao{
public:
    Garagem(char i): Fraccao(i, 12){
        cout << " do tipo Garagem" << endl;
    }
    void mostrar(){
        cout << "Garagem ";
        Fraccao::mostrar();
    }
    ~Garagem(){ cout << "Removida Garagem " << id << endl;}
};

```

```

class Predio{
    static const int MAXFRAC=20;
    Fracao *fraccoes[MAXFRAC];
    int nfrac;
public:
    Predio():nfrac(0){
        cout << "Criado Predio." << endl;
    }
    void addGaragem(char i){
        if(nfrac<MAXFRAC) fraccoes[nfrac++]=new Garagem(i);
    }
    void addHabitacao(char i){
        if(nfrac<MAXFRAC) fraccoes[nfrac++]=new Habitacao(i);
    }
    void mostrarFraccoes(){
        cout << endl << "Fraccoes do Predio:" << endl;
        for(int i=0; i<nfrac; i++) fraccoes[i]->mostrar();
    }
    ~Predio(){}
};

void main(){
    Predio p;
    p.addHabitacao('A');
    p.addGaragem('K');
    p.addHabitacao('B');
    p.mostrarFraccoes();
}

```

- Identifique o tipo de relação existente entre Predio e Habitacao, entre Fracao e Garagem, e entre Predio e Fracao. [0.6 val.]
- Alguma das classes é abstracta? [0.2 val.]
- Numa das classes o destrutor não tem a implementação mais adequada. Diga qual é esse destrutor e dê-lhe a implementação desejada. [1.6 val.]
- Apresente o resultado que será visualizado na saída standard após a execução do programa e depois de resolvida a inconformidade referenciada na alínea anterior. [2.6]
- E o que seria visualizado se o método Fracao::Mostrar() não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [1 val.]
- Admita agora que o prédio passou a poder incorporar, para além de habitações e garagens, lojas comerciais com diferentes permilagens. Acrescente ao programa os métodos e/ou classes necessárias, para que passe a reflectir esta nova realidade. [2 v.]

Anexo

Class Coleccao

```
#include<set>
using namespace std;

template<class K>
class Coleccao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

Class ColeccaoHibrida

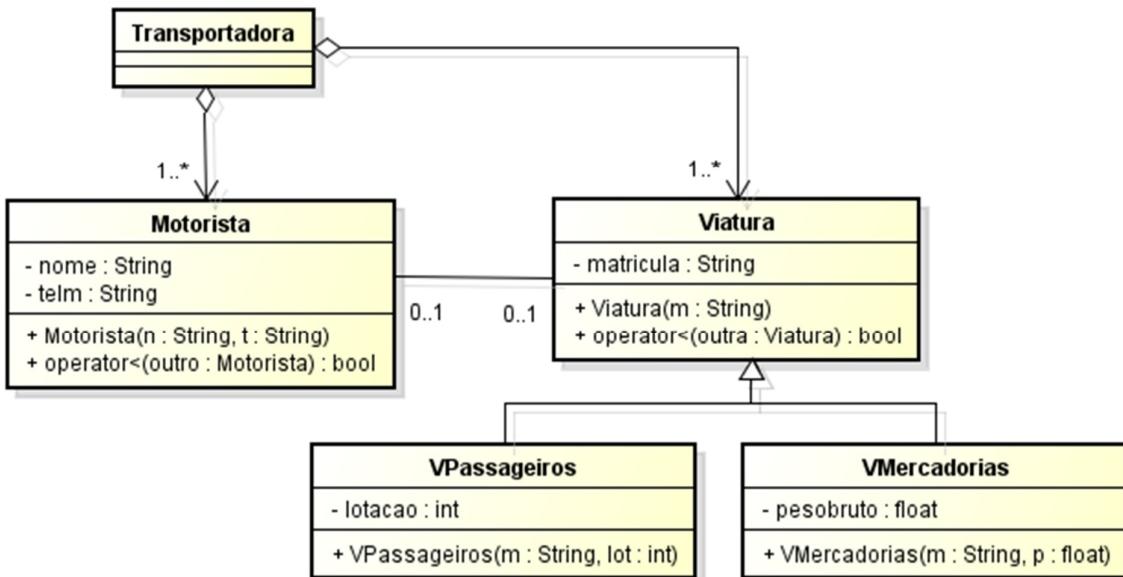
```
#include<set>
using namespace std;

template<class T>
class less_pointers
{
public:
    bool operator()(const T &left, const T &right) const
    {
        return (*left < *right);
    }
};

template<class K>
class ColeccaoHibrida: public set<K, less_pointers<K>>
{
public:
    bool insert(const K &c);
    K find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

a) Apresente, através de um diagrama de classes em UML, a solução por si



b) Codifique em C++ a sua solução, tal como a descreveu no diagrama da anterior.

```

class Transportadora{
    Coleccao<Motorista> motoristas;
    ColeccaoHibrida<Viatura*> viaturas;
};

class Motorista{
    string nome;
    string telm;
    Viatura *viatura;
public:
    Motorista(const string &n, const string &t): nome(n), telm(t){
        viatura=NULL;
    }
    bool operator<(const Motorista &outro) const {return nome<outro.nome;}
};
class Viatura{
    string matricula;
    Motorista *motorista;
public:
    Viatura(const string &m): matricula(m){motorista=NULL;}
    virtual bool operator<(const Viatura &outra) const {
        return matricula<outra.matricula;
    }
};

class VPassageiros: public Viatura{
    int lotacao;
public:
    VPassageiros(const string &m, int lot): Viatura(m){lotacao=lot;}
};

class VMercadorias: public Viatura{
    float pesobruto;
public:
    VMercadorias(const string &m, float p): Viatura(m){pesobruto=p;}
};

```

c) Implemente os métodos necessários para adicionar (registar) à aplicação um novo motorista e um novo veículo de passageiros.

```
bool Transportadora::addMotorista(const string &n, const string &t){  
    Motorista m(n,t);  
    return motoristas.insert(m);  
}  
  
bool Transportadora::addVPassageiros(const string &m, int lot){  
    VPassageiros *p = new VPassageiros(m,lot);  
    return viaturas.insert(p);  
}
```

d) Implemente o(s) método(s) necessários para afectar uma viatura a um motorista.

```
bool Transportadora::afectaViatMotorista(const string &m, const string &n){  
    Viatura *pv=findViatura(m);  
    if(pv==NULL) {cout<<"viatura inexistente!\n"; return false;}  
    else{  
        Motorista *pm=findMotorista(n);  
        if(pm==NULL) {cout<<"Motorista inexistente!\n"; return false;}  
        else{  
            pm->setViatura(pv);  
            pv->setMotorista(pm);  
            return true;  
        }  
    }  
}  
Viatura *Transportadora::findViatura(const string &m){  
    Viatura v(m);  
    return viaturas.find(&v);  
}  
  
Motorista *Transportadora::findMotorista(const string &n){  
    Motorista m(n,"");  
    return motoristas.find(m);  
}  
  
void Motorista::setViatura(Viatura *v){viatura=v;}  
  
void Viatura::setMotorista(Motorista *m){ motorista=m;}
```

e) Implemente o(s) método(s) que permita(m) obter o número de telemóvel do motorista que conduza uma viatura com uma dada matrícula.

```
string Transportadora::getTelemMotViat(const string &m){  
    Viatura *pv=findViatura(m);  
    if(pv==NULL){  
        cout<<"Viatura inexistente!";  
        return "";  
    }else if(pv->getMotorista()==NULL){  
        cout<<"Viatura parada!";  
        return "";  
    }else return pv->getMotorista()->getTelem();  
}  
  
Motorista *Viatura::getMotorista() const{return motorista;}  
  
string Motorista::getTelem() const{return telm;}
```

f) listar ... com indicação das matrículas e do nome dos respectivos condutores.

```
void Transportadora::listarViatEmCirc()const {
    ColeccaoHibrida<Viatura*>::iterator it;
    for(it=viaturas.begin(); it!=viaturas.end(); it++)
        if((*it)->getMotorista()!=NULL)
            cout<<(*it)->getMotorista()->getNome()
                <<" conduz "<<(*it)->getMatricula()<<endl;
}

string Motorista::getNome() const{return nome;}

string Viatura::getMatricula() const{return matricula;}
```

Grupo II

- a) agregação/herança/agregação
- b) não
- c) ~Predio(){ for(int i=0; i<nfrac; i++) delete fraccoes[i]; }
- d)

```
Criado Predio.
Criada Fracao A do tipo Habitacao
Criada Fracao K do tipo Garagem
Criada Fracao B do tipo Habitacao

Fracoes do Predio:
Habitacao A com permilagem 40
Garagem K com permilagem 12
Habitacao B com permilagem 40
Removida Habitacao A
Removida Garagem K
Removida Habitacao B
```

- e)
- Fracoes do Predio:
A com permilagem 40
K com permilagem 12
B com permilagem 40

f)

```
void Predio::addLoja(char i, int p){
    if(nfrac<MAXFRAC) fraccoes[nfrac++]=new Loja(i,p);
}

class Loja: public Fracao{
public:
    Loja(char i, int p): Fracao(i,p){
        cout << " do tipo Loja" << endl;
    }
    void mostrar(){
        cout << "Loja ";
        Fracao::mostrar();
    }
    ~Loja(){ cout << "Removida Loja "<< id << endl;}
};
```

Departamento de Informática e Comunicações

Engenharia Informática / Informática de Gestão
2º Ano
Época de Recurso

Programação Orientada por Objectos
1º Semestre

Data: 13/2/2010
Duração: 2h00

Grupo I
(10 valores)

Suponha que pretende desenvolver uma pequena aplicação que lhe permita gerir as vigilâncias que os vários docentes têm de assegurar durante uma época de exames. No sentido de simplificar a solução, considere que o docente é caracterizado apenas pelo seu nome, e cada prova de exame pelo nome da disciplina a que respeita e pela data de realização. Tenha ainda em conta que a vigilância de cada exame é, por norma, assegurada por vários docentes, e um mesmo docente tem a seu cargo um máximo de 10 vigilâncias. Considere, por fim, que podem existir vários exames para uma mesma disciplina, desde que em dias diferentes.

- a) Apresente, através de um diagrama de classes em UML, a solução por si idealizada para sustentar a aplicação descrita, indicando para cada classe apenas os atributos estritamente necessários, um método construtor e outros componentes que considere essenciais ao funcionamento da solução (*getters*, *setters* e, nas classes em que se justifique, o operador que permita que os objectos sejam colecccionáveis). [2val]
- b) Codifique em C++ a sua solução, tal como a descreveu no diagrama da alínea anterior. [3 val.]

NOTA: As colecções deverão ser implementadas com base no template de classes *Coleccao* que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos:

```
template<class K>
class Coleccao: public set<K>{
public:
    bool insert(const K &c);
    K *find(const K &c);
    int size() const;
    void erase(const K &);

    //void clear();
    //bool empty() const;
    //iterator begin();
    //iterator end();
};
```

- c) Implemente o(s) método(s) necessário(s) para registar um novo exame na aplicação. [1 val.]
- d) Implemente o(s) método(s) necessário(s) para mostrar todas a vigilâncias de um dado docente. [2 val.]
- e) Implemente o(s) método(s) necessário(s) para remover um dado exame. [2 val.]

Grupo II
(10 valores)

Considere o seguinte programa:

```
#include<iostream>
#include<string>
using namespace std;

class Cliente{
    string nome;
public:
    Cliente(const string & name): nome(name){}
    void print()const{cout<<nome<<endl;}
};

class Conta{
protected:
    int numero;
    double saldo;
    Cliente *titular1;
    static int numDeContas;
public:
    Conta(Cliente *t): titular1(t){
        numero=++numDeContas;
        saldo=0.0;
        cout<<"Criada conta n."<<numero<<endl;
    }
    virtual void depositar(double val) {if(val>0.0) saldo+=val;}
    virtual void levantar(double val) {
        if(val>0.0 && val<=saldo) saldo-=val;
    }
    virtual void print() const {
        cout<<"Conta n."<<numero<<" saldo:"<<saldo<<endl;
        cout<<"com lo titular: ";
        titular1->print();
    }
    virtual ~Conta(){cout<<"Fechada\n";}
};
int Conta::numDeContas=0;
```

```

class ContaAPrazo: public Conta{
    Cliente *titular2;
public:
    ContaAPrazo(Cliente *t1, Cliente *t2): Conta(t1), titular2(t2){
        cout<<"<Conta a Prazo>\n";
    }
    void depositar(double val) {
        if(val>=100.0) Conta::depositar(val);
        else cout<<"Deposito invalido\n";
    }
    void print() const {
        Conta::print();
        cout<<"com 2o titular: ";
        titular2->print();
    }
    ~ContaAPrazo(){cout<<"Conta a Prazo ";}
};

void main(){
    Cliente cl1("Ana"), cl2("Rita");
    Conta *c=new Conta(&cl1);
    delete c;
    c=new ContaAPrazo(&cl1,&cl2);
    c->depositar(50);
    c->levantar(10);
    c->print();
    delete c;
}

```

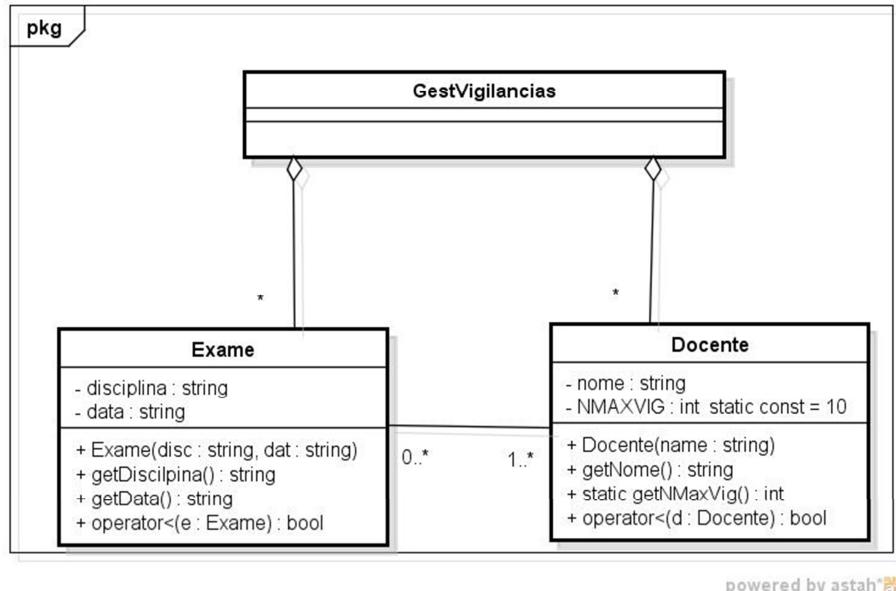
- a) Identifique o tipo de relação existente entre as diferentes classes definidas no programa. [1 val.]
- b) Apresente o resultado que será visualizado na saída standard com a execução do programa. [4 val.]
- c) Diga que alterações é que teria no resultado visualizado se na classe Conta [2 val.]
- i) o método “void depositar(double val)” não fosse virtual.
 - ii) o método “void levantar(double val)” não fosse virtual.
 - iii) o método “void print()” não fosse virtual.
 - iv) o método “~Conta()” não fosse virtual.
- d) Se acrescentássemos à função `main()` a linha de código que se segue, daí resultaria algum erro na nossa aplicação? Justifique. [1 val.]

```
ContaAPrazo contas[1000];
```

- e) Acrescente ao programa uma classe para um novo tipo de conta, de um só titular, denominada ContaAOrdem. Esse novo tipo de conta deverá ser caracterizado por não permitir que o saldo fique, em nenhum momento, a zero e pela obrigatoriedade de se efectuar um depósito (não importa o montante) no momento da sua abertura (criação da conta). [\[2 val.\]](#)

Grupo I

a)



powered by astah®

b)

```

class GestVigilancias{
    Coleccao<Exame> exames;
    Coleccao<Docente> docentes;
public:
};

class Exame{
    string disciplina;
    string data;
    Coleccao<Docente*> vigilantes;
public:
    Exame(const string &disc, const string &dat) {disciplina=disc; data=dat;}
    string getDisciplina() const {return disciplina;}
    string getData() const {return data;}
    bool operator<(const Exame& exam) const {
        return disciplina+data<exam.disciplina+exam.data;
    }
};

class Docente{
    string nome;
    Coleccao<Exame*> vigilancias;
    static const int NMAXVIG;
public:
    Docente(const string &name): nome(name){}
    string getName() const{return nome;}
    static int getNMaxVig(){return NMAXVIG;}
    bool operator<(const Docente& doc) const {return nome<doc.nome;}
};

const int Docente::NMAXVIG=2;

```

```

c)
bool GestVigilancias::addExame(const string &disc, const string &dat){
    Exame e(disc,dat);
    return exames.insert(e);
}

d)
void GestVigilancias::listarVigilanciasDeDocente(const string &name){
    Docente *fd=findDocente(name);
    if(fd!=NULL) fd->listarVigilancias();
    else cout << "Docente " << name << " nao existe!\n";
}

void Docente::listarVigilancias(){
    Coleccao<Exame*>::iterator it;
    cout<<"Vigilancias do docente "<<nome<<":\n";
    for(it=vigilancias.begin(); it!=vigilancias.end(); it++)
        cout<<(*it)->getDisciplina()<< " "<<(*it)->getData()<<endl;
}

e)
void GestVigilancias::removerExame(const string &disc, const string &dat) {
    Exame *fe=findExame(disc, dat);
    if(fe!=NULL){
        fe->removerVigilantes();
        Exame e(disc,dat);
        exames.erase(e);
    }else cout << "Exame de " << disc << " na data " <<dat<< " nao existe!\n";
}

void Exame::removerVigilantes(){
    Coleccao<Docente*>::iterator it;
    for(it=vigilantes.begin(); it!=vigilantes.end(); it++)
        (*it)->removerVigilancia(this);
    vigilantes.clear();
}

void Docente::removerVigilancia(Exame *e) {vigilancias.erase(e);}

```

Grupo II

- a) Identifique o tipo de relação existente entre as diferentes classes definidas no programa. [1 val.]

Herança entre Conta e ContaAPrazo

Agregação/Associação entre Conta e Cliente

- b) Apresente o resultado que será visualizado na saída standard com a execução do programa. [3 val.]

```

Criada conta n.1
Fechada
Criada conta n.2
<Conta a Prazo>
Deposito invalido
Conta n.2 saldo:0
com 1º titular: Ana
com 2º titular: Rita
Conta a Prazo Fechada

```

c) Diga que alterações é que teria no resultado visualizado se na classe Conta [2 val.]

- i) o método depositar(.) não fosse virtual.
 não aparecia “Deposito invalido”
 em vez de “saldo:0” aparecia “saldo:40”
- ii) o método levantar(.) não fosse virtual.
 não haviam alterações
- iii) o método print(.) não fosse virtual.
 não era mostrado o 2º titular
- iv) o método ~Conta(.) não fosse virtual.
 em vez de “Conta a Prazo Fechada” aparecia apenas “Fechada”

d) Se acrescentássemos à função main() a linha de código que se segue, resultaria daí algum erro na nossa aplicação? Justifique. [1 val.]

```
ContaAPrazo contas[10000];
```

Sim. Só é possível criar arrays de instâncias de classes se estas dispuserem de um construtor por defeito, algo que não acontece com a ContaAPrazo.

e) Acrescente ao programa uma classe para um novo tipo de conta, de um só titular, denominada ContaAOrdem. Esse novo tipo de conta deverá ser caracterizado por não permitir que o saldo fique, em nenhum momento, a zero e pela obrigatoriedade de se efectuar um depósito (não importa o montante) no momento da sua abertura (criação da conta). [3 val.]

```

class ContaAOrdem: public Conta{
public:
    ContaAOrdem(Cliente *t1, double dep): Conta(t1){depositar(dep);}
    void levantar(double val) {if(val<saldo) levantar(val);}
    ~ContaAOrdem(){cout<<"Conta a Ordem ";}
};
```