

**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;
    Colecao<Figura> figuras;
    ColecaoHibrida<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();
    ColecaoHibrida<FormaGeometrica*>::iterator it1;
    for (it1 = formasGeometricas.begin(); it1 != formasGeometricas.end(); it1++)
        (*it1)->print();
    cout << "[";
    Colecao<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.]  
Nenhumas.

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

```
bool Equipa::operator==(const Equipa &outra) { return pontos==outra.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.