Engenharia Informática
Informática de Gestão
Época de Recurso – 7 de fevereiro de 2015

Grupo I
(7.2 valores)

a) Identifique o tipo de relação existente entre as classes *Desenho* e *Figura*. [0.2 val.]
Agregação. Desenho agrega 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){}            [1]
      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();
      }
```