Programação Orientada a Objectos

Exame época de recurso

2015/16

1ª Parte

DEIS – Engenharia Informática

19 / Fev / 2016

Nota: namespaces e #includes não são relevantes para os excertos de código apresentados.

1. a) (5%) Qual será a saída resultante da execução deste programa?

```
class A {
public:
    virtual void f() { cout << "\n f de A "; }
    void g() { cout << "\n g de A "; }
    void h() { cout << "\n h de A ";
    }
};
class B : public A {
public:
    virtual void f() { cout << "\n f de B ";   }
    void g() { cout << "\n g de B "; }
    virtual void h() { cout << "\n h de B "; }
};
void listar(A * a) {
    a->f();
    a->g();
    a->h();
}
int main() {
    cout << "\n ---1---";
    listar(new A);
    cout << "\n ---2---";
    listar(new B);
}
```

2. (5%) Defina a classe Leao, derivada de Vertebrado, de modo que seja concreta (possam existir objectos do tipo Leao). A nova classe deve ter apenas o estritamente necessário para ser concreta.

```
class Animal {
public:
    virtual void mover() = 0;
    virtual void respirar() = 0;
};
class Vertebrado : public Animal {
    string nome;
public:
    Vertebrado(string s){ nome = s; }
    virtual void respirar(){};
    virtual void alimentar() = 0;
};
```

3. (5%) Considere as seguintes definições:

```
class Figura {
    Ponto ponto; // um ponto relativo à figura, considera-se definida a classe Ponto
public:
    // funções membros consideradas definidas
};
class Poligono: public Figura{
    Ponto * p; // array dinâmico de pontos
    int nPontos; // numero de elementos do array dinâmico
public:
    // ...
};
```

Supondo que a classe Figura está completamente definida, escreva um construtor por cópia para a classe Poligono.

4. (5%) Indique quais as linhas deste programa em que ocorrem erros de compilação? Justifique.

```
class Teste {
    int num;
public:
    Teste():num(0){}
    int getNum(){ return num; }
    void setNum( int n){ num = n; }
};
void func1( Teste & a, const Teste & b){
    a.setNum(2);
    cout << a.getNum();
    b.setNum(4);
    cout << b.getNum();
}
void func2( const Teste * p1, Teste * const p2){
    p1->setNum(2);
    p1 = p2;
    p2->setNum(4);
    p2 = p2;
}
```

5. (5%) Considere o seguinte programa:

```
class Tabela {
    int * p;
    int n;
public:
    ~Tabela(){ delete [] p; }
    // ...
};
int main(){
    int x [] = { 1, 2, 3};
    Tabela a( a, 3);
    Tabela b = a;
    b = a;
    cout << a << b;
    a << 4;
    a[0] = 5;
    int n = a;
}
```

5. a) Escreva os protótipos (apenas os protótipos) das funções que necessitam de estar definidas para que este programa funcione correctamente, tanto do ponto de vista da compilação como da execução. As funções que indicar devem estar devidamente contextualizadas, ou seja, deve ficar claro se pertencem à classe Tabela ou se são globais.

5. b) Transcreva a função main() e, para cada linha, indique quais as funções cujos protótipos indicou na alínea anterior ou outras da classe Tabela estão envolvidas na sua execução.

6. (7%) O programa seguinte tem erros de execução. Escreva uma versão corrigida deste programa de forma que funcione da forma que seria normal de acordo com o significado indicado pelo nome das funções. Não pode alterar o membro variável, nem o protótipo das funções (e o objectivo que o nome sugere), nem o destrutor, nem a função main().

```
class Lista {
    vector<string *> nomes;
public:
    Lista(){}
    void acrescenta( string * a){
        string * p = new string(*a);
        string b = *p;
        nomes.push_back( &b);
    }
    void elimina( string a){
        for( unsigned int i = 0 ; i < nomes.size(); i++){
            if(*(nomes[i]) == a){
                delete nomes[i];
            }
        }
    }
    Lista( const Lista & ob){
        *this = ob;
    }
    Lista & operator=( const Lista & ob){
        for( unsigned int i = 0 ; i < nomes.size(); i++){
            delete nomes[i];
        }
        nomes.clear();
        for( unsigned int i = 0 ; i < ob.nomes.size(); i++){
            nomes.push_back(ob.nomes[i]);
        }
    }
    ~Lista(){
        for( int i = 0 ; i < nomes.size(); i++){
            delete nomes[i];
        }
    }
};
int main(){
    Lista a;
    string x = "um", y = "dois";
    a.acrescenta(&x);
    a.acrescenta(&y);
    Lista b(a), c;
    c = a;
    c = c;
    a.elimina("um");
}
```

7. (8%) As classes Gato e Canario representam as características dos gatos e canários que vão fazer parte de um jardim zoológico. A classe Zoo tem os dados de todos os animais que vivem nesse jardim zoológico. Todos os animais têm uma voz específica (falam). A função falamTodos() da classe Zoo tem como objectivo listar a voz de todos os seus animais. Para além de gatos e canários, o jardim zoológico vai receber leões, tigres, zebras e muitas outras espécies de animais. O código apresentado a seguir, não estando organizado de acordo com os conceitos da programação orientada a objectos (encapsulamento, herança e polimorfismo), dificilmente se poderia adaptar à integração destas variadas espécies.
Reescreva o código aplicando de forma correcta estes conceitos. Pode considerar que, na versão que vai apresentar, a relação entre Zoo e Animal é de agregação.

```
class Gato{
    int numPatas;
    string nome;
public:
    Gato(string s){ nome = s; numPatas = 4;}
    void falar(){
        cout << "\n miau";
    }
};
class Canario{
    int numPatas;
    string nome;
public:
    Canario(string s){ nome = s; numPatas = 2;}
    void falar(){
        cout << "\n piu piu";
    }
};
class Zoo {
    vector<Gato> gatos;
    vector<Canario> canarios;
public:
    void acrescentaGato( Gato a){
        gatos.push_back(a);
    }
    void acrescentaCanario( Canario a){
        canarios.push_back(a);
    }
    void falamTodos(){
        for(unsigned int i = 0 ; i < gatos.size(); i++){
            gatos[i].falar();
        }
        for(unsigned int i = 0 ; i < canarios.size(); i++){
            canarios[i].falar();
        }
    }
};
int main(){
    Zoo z;
    z.acrescentaGato(Gato("Soneca"));
    z.acrescentaCanario(Canario("Pompom"));
    z.falamTodos();
}
```