INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e de Gestão

Departamento de Informática e Comunicações

Engenharia Informática / Informática de Gestão
2º Ano
Época Normal

Programação Orientada por Objectos — Data: 23/01/2012
1º Semestre — Duração: 1h30

Resolva os 2 grupos em folhas de exame separadas — Tolerância: 0h15

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 representação de uma equipa de futebol, formada por onze jogadores, um dos quais necessariamente guarda-redes. Depois de o analisar com a devida atenção, responda às seguintes questões.

a) Identifique o tipo de relação existente entre Jogador e GuardaRedes, entre Equipa e Jogador, e entre Equipa e GuardaRedes. [0.5 val.]

b) Diga, para cada uma das classes, quais os métodos que deveriam ser definidos como constantes. [1.0 val.]

c) Diga o que entende por classe abstracta, se alguma das classes do problema é abstracta e de que forma é que se definem classes abstractas em C++? [1.5 val.]

d) Numa das classes o destrutor não tem a implementação mais adequada. Diga qual é esse destrutor e dê-lhe a implementação desejada. [1.5 val.]

e) Apresente o resultado que será visualizado na saída standard após a execução do programa e depois de resolvida a inconformidade mencionada na alínea anterior. [3.0]

f) E o que seria visualizado se o método Jogador::mostrar() não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [1 val.]

g) Considere agora a classe adicional e o código substituto para a função main() que a seguir se apresentam. Qual o resultado que será visualizado na saída standard. [1,5 v.]

```
class EquipaComNome: public Equipa{
      string nome;
public:
      EquipaComNome(const string &name): nome(name){
            cout << nome << endl;
      }
      ~EquipaComNome(){cout << nome << " excluido" <<endl;}
};
```

```
void main(){
      Equipa *pe1=new EquipaComNome("slb");
      EquipaComNome e2("fcp");
      delete pe1;
}
```

Grupo II
(10 valores)

Uma importante Companhia Aérea (CA) necessita de uma pequena aplicação que lhe permita manter registadas todas as escalas realizadas pelas suas aeronaves nos diferentes aeroportos em que opera (entenda-se por escala o período de permanência de uma aeronave num dado aeroporto). A companhia dispõe de uma frota considerável de aeronaves, identificadas por um código inteiro. Já os aeroportos por elas visitados são identificados pelo seu nome. Relativamente a cada escala, pretende-se que fique registado, para além código inteiro que identifique a escala, as datas de chegada e de partida da aeronave.
(Pode utilizar o tipo string da STL para o tipo de dados a usar nas datas, e considere que essas strings vão conter informação quer relativamente à data quer relativamente à hora.)

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 coleccionáveis. [3 val.]
Não indique quaisquer outros métodos.

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

> *NOTA1: 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 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 directivas #include.*

c) Implemente os métodos necessários para adicionar à aplicação um novo aeroporto e uma nova aeronave. [1 val.]

d) Implemente o(s) método(s) necessário(s) para registar uma nova escala. [3 val.]

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

class Jogador{
protected:
	int num;
public:
	Jogador(int n): num(n){
		cout << "Novo Jogador " << endl;
	}
	int getNum(){return num;}
	virtual void mostrar(){cout <<"Jogador "<< num <<endl;}
	virtual ~Jogador(){cout << "Jogador " << num << " excluido" <<endl;}
};

class GuardaRedes: public Jogador{
public:
	GuardaRedes(): Jogador(1){ cout << "Novo Guarda-redes" << endl;}
	void mostrar(){
		cout << "<Guarda-redes> ";
		Jogador::mostrar();
	}
	~GuardaRedes(){cout << "Guarda-redes excluido" <<endl;}
};

class Equipa{
	static const int MAXJOG=11;
	Jogador *jogadores[MAXJOG];
	int njog;
public:
	Equipa():njog(0){cout << "Criada Equipa" << endl;}
	bool jaTemGuardaRedes(){
		int i;
		for(i=0; i<njog && jogadores[i]->getNum()!=1; i++);
		return i<njog;
	}
	void addJogador(int n){
		if(njog<MAXJOG-1 || (njog==MAXJOG-1 && jaTemGuardaRedes()))
			jogadores[njog++]=new Jogador(n);
	}
	void addGuardaRedes(){
		if(!jaTemGuardaRedes()) jogadores[njog++]=new GuardaRedes();
	}
	void mostrar(){
		cout << "Jogadores da equipa:" << endl;
		for(int i=0; i<njog; i++) jogadores[i]->mostrar();
	}
	~Equipa(){}
};

void main(){
	Equipa slb;
	slb.addJogador(10);
	slb.addGuardaRedes();
	slb.addJogador(5);
	slb.mostrar();
}
```

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();
};
```