Grupo I

a) Identifique … Jogador e GuardaRedes, entre Equipa e Jogador, e entre Equipa e GuardaRedes. [0.5 val.]
Herança, Agregação, Agregação

b) Diga, para cada uma das classes, quais os métodos que deveriam ser definidos como constantes. [1.0 val.]
```
Jogador: getNum(), mostrar()
GuardaRedes: mostrar()
Equipa: jaTemGuardaRedes(), mostrar()
```

c) Diga o que entende por classe abstracta, se alguma é abstracta e com se definem em C++? [1.5 val.]
Classe abstracta é uma classe não instanciável
Nenhuma classe é abstracta
Em C++ uma classe abstracta é uma classe que tem um ou mais métodos virtuais puros

d) Um destrutor não tem a implementação mais adequada. Diga qual é e dê-lhe a implement… [1.5 val.]
Destrutor da classe Equipa
```
~Equipa(){
      for(int i=0; i<njog; i++) delete jogadores[i];
}
```

e) Apresente o resultado que será visualizado na saída standard. [3.0]
```
Criada Equipa
Novo Jogador
Novo Jogador
Novo Guarda-redes
Novo Jogador
Jogadores da equipa:
Jogador 10
<Guarda-redes> Jogador 1
Jogador 5
Jogador 10 excluido
Guarda-redes excluido
Jogador 1 excluido
Jogador 5 excluido
```

f) E o que seria visualizado se o método Jogador::mostrar() não fosse virtual? Refira-se apenas… [1 val.]
```
<Guarda-redes> Jogador 1 passaria a Jogador 1
```

g) Considere agora ... Qual o resultado que será visualizado na saída standard. [1,5 v.]

```
Criada Equipa
slb
Criada Equipa
fcp
fcp excluido
```

b)

```
class CompAerea{
      Coleccao<Aeroporto> aeroportos;
      Coleccao<Aeronave> aeronaves;
public:
};

class Aeronave{
      int id;
      Coleccao<Escala> escalas;
public:
      Aeronave(int i){id=i;}
      virtual bool operator<(const Aeronave &outra)const {return id<outra.id;}
};

class Aeroporto{
      string nome;
      Coleccao<Escala *> escalas;
public:
      Aeroporto(const string &n): nome(n){}
      bool operator<(const Aeroporto &outro)const {return nome<outro.nome;}
};

class Escala{
      int id;
      string chegada, partida;
      Aeroporto* aporto;
      Aeronave* anave;
public:
      Escala(int i, const string &cheg,const string &part,Aeroporto* ap,Aeronave* an):
        chegada(cheg), partida(part){
            id=i;
            aporto=ap;
            anave=an;
        }
      virtual bool operator<(const Escala &outra)const {return id<outra.id;}
};
```

c)

```
classe CompAerea:
      bool addAeronave(int i){
            Aeronave an(i);
            return aeronaves.insert(an);
      }
      Bool addAeroporto(const string &n){
            Aeroporto ap(n);
            return aeroportos.insert(ap);
      }
```

d)

```
classe CompAerea:
	bool addEscala(int es, int an, string ap, string cheg, string part){
		Aeroporto *pap=findAeroporto(ap);
		if(pap!=NULL){
			Aeronave *pan=findAeronave(an);
			if(pan!=NULL){
				return pan->addEscala(es,pap,cheg,part);
			}else return false;
		}else return false;
	}
	Aeroporto *findAeroporto(const string &n){
		Aeroporto ap(n);
		return aeroportos.find(ap);
	}
	Aeronave *findAeronave(int i){
		Aeronave ap(i);
		return aeronaves.find(ap);
	}

classe Aeronave:
	bool addEscala(int es, Aeroporto *ap, string cheg, string part){
		Escala e(es,cheg,part,ap,this);
		if (escalas.insert(e)){
			ap->add(escalas.find(e));
			return true;
		}else return false;
	}

classe Aeroporto:
	bool add(Escala *pe){return escalas.insert(pe);}
```