Grupo I

a) Propositadamente, todos os métodos da classe AveRara contêm um erro de compilação ou de execução. Identifique-os e proponha, para cada um deles, uma solução adequada (correções que deverá levar em conta na resolução das restantes alíneas). [1.6 val.]

```
AveRara():Ave(0) {...
void acrescentar(int n) const{...
void mostrar(void) const{
     ...
     Ave::mostrar();
}
~AveRara(){cout<<"menos uma ave rara com "<<num_penas<<" penas\n";}
                          Declara-se o atributo como protected na classe Ave
};
```

b) Alguma das classes do problema é abstrata? Se sim, diga qual ... motivo a considera abstrata. [0.4 val.]
Não

c) Como sabe, para além dos métodos que são definidos de forma explícita em cada classe, existirão outros definidos automaticamente pelo C++. Diga quais são os métodos implícitos que estarão presentes em cada uma das duas classes do problema. [1.0 val.]
Na classe Ave:
- Construtor de cópia
- Operador afetação

Na classe AveRara:
- Construtor de cópia
- Operador afetação

d) Apresente o resultado que será visualizado na saída standard após a execução do programa. [2.6]

```
nova ave com 0 penas
nova ave rara
nova ave com 500 penas
tenho pena de nao ter penas!
ACRESCENTAR:
tenho pena de nao ter penas!
MOSTRAR:
regeitei 13 penas
tenho 0 penas
regeitei 13 penas
tenho 0 penas
tenho 520 penas
ABATER:
uma ave a menos
menos uma ave rara com 0 penas
uma ave a menos
```

e) O que seria visualizado se o método *mostrar* da classe Ave não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [0.8 val.]

```
MOSTRAR:
regeitei 13 penas
tenho 0 penas
regeitei 13 penas
tenho 0 penas
tenho 520 penas
ABATER:
```

f) E o que seria visualizado se fosse o método *acrescentar* a não ser virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea d). [0.8 val.]

```
ACRESCENTAR:
tenho pena de nao ter penas!
MOSTRAR:
regeitei 13 3 penas
tenho 0 10 penas
regeitei 13 3 penas
tenho 0 10 penas
tenho 520 penas
ABATER:
uma ave a menos
menos uma ave rara com 0 10 penas
uma ave a menos
```

g) Considerando as seguintes declarações: `AveRara a;` `Ave *p=&a;` diga, justificando, se a instrução `(*p)++;` é ou não válida no problema considerado. Caso não seja, introduza as alterações necessárias nas classes do problema para que a instrução passe a ser válida. [0.8 val.]

```
void Ave::operator++(int){
      num_penas++;
}
```

a)

```
class Viatura{
      string matricula;
      Colecao<Aluguer *> alugueres;
public:
      Viatura(const string &m): matricula(m){}

      bool addAluguer(Aluguer *a){return alugueres.insert(a);}

      bool operator<(const Viatura &outra)const {return matricula<outra.matricula;}
};


class Aluguer{
      string levantamento, devolucao;
      Viatura* viatura;
      Cliente* cliente;
public:
      Aluguer(const string &lev, const string &dev, Viatura* v, Cliente* c):
             levantamento(lev), devolucao(dev){
            viatura=v;
            cliente=c;
      }

      bool operator<(const Aluguer &outro)const {
            return levantamento<outro.levantamento;
      }
};


class Cliente{
      string nome;
      Colecao<Aluguer> alugueres;
public:
      Cliente(string n){nome=n;}

      bool addAluguer(string lev, string dev, Viatura *v){
            Aluguer a(lev,dev,v,this);
            if (alugueres.insert(a)) return v->addAluguer(alugueres.find(a));
            else return false;
      }

      bool operator<(const Cliente &outro)const {return nome<outro.nome;}
};
```

```
class AutoAluga{
	Colecao<Viatura> viaturas;
	Colecao<Cliente> clientes;
public:
	bool addCliente(const string &nom){
		Cliente cl(nom);
		return clientes.insert(cl);
	}

	bool addViatura(const string &mat){
		Viatura v(mat);
		return viaturas.insert(v);
	}

	Viatura *findViatura(const string &mat){
		Viatura v(mat);
		return viaturas.find(v);
	}

	Cliente *findCliente(const string &nom){
		Cliente cl(nom);
		return clientes.find(cl);
	}

	bool addAluguer(string lev, string dev, string mat, string nom){
		Viatura *pv=findViatura(mat);
		if(pv!=NULL){
			Cliente *pc=findCliente(nom);
			if(pc!=NULL) return pc->addAluguer(lev,dev,pv);
			else return false;
		}else return false;
	}

};


b)
int AutoAluga::totalAlugueres()const{
	int tot=0;
	Colecao<Cliente>::iterator it;
	for(it=clientes.begin(); it!=clientes.end(); it++)
		tot+=it->quantidadeAlugueres();
	return tot;
}

int Cliente::quantidadeAlugueres()const {return alugueres.size();}


c)
void main(){
	AutoAluga aa;
	aa.addCliente("Ana Rita");
	aa.addViatura("11-22-AA");
	aa.addAluguer("1/1/2012 <19:25>", "3/1/2012 <19:05>", "11-22-AA", "Ana Rita");
	cout<<"Total de alugueres: "<<aa.totalAlugueres()<<endl;
}
```