Caderno de Exercícios elaborado por José Exposto,
com contribuições e adaptações de Paulo Gouveia

Para os 4 exercícios que se seguem (problemas de modelação) faça o respetivo diagrama de classes, explicitando as associações simples e de agregação, bem como as relações de herança. Pretende-se um modelo simples, não sendo por isso necessário especificar os métodos e atributos das respetivas classes (apenas no exercício 4 tente completar as classes com alguns dos seus membros principais – atributos e métodos).

1. Suponha que pretende mandar construir uma piscina. São os seguintes os modelos disponíveis: Caraíbas, em formato oval com palmeiras; Hawai, em formato retangular com prancha de mergulho; Climatizada, retangular com cobertura; e Confortável, que é uma piscina retangular e é a melhor de todas uma vez que tem tudo o que as outras têm.

2. O Conselho Pedagógico (CP) é um órgão duma Escola onde estão representados alunos e docentes (professores e assistentes). Integram o CP uma Comissão Executiva (CE) e várias Comissões de Curso (CC). A CE é constituída por um presidente e dois vogais, eleitos de entre os membros do CP, sendo o presidente obrigatoriamente um professor e os vogais, necessariamente, um docente e um aluno. Cada CC é constituída por 3 alunos e 3 docentes, exercendo um destes docentes as funções de diretor de curso. Note que os 3 membros do CE são também membros de CC.

3. A ESTIG integra Departamentos, Centros de Recursos e os seguintes Serviços: Serviços Administrativos (S. Ad.), Serviços Académicos (S. Ac.) e Serviços Técnicos (S. Tec.). Os docentes que pertencem a cada departamento são Assistentes ou Professores Adjuntos, e todos eles, para além de exercerem funções de docência, são também investigadores. Cada departamento dispõe ainda de um coordenador, eleito de entre os Professores Adjuntos. Qualquer Centro de Recursos ou Serviços é dirigido por um responsável e dispõem de um ou mais funcionários. Porém, enquanto os responsáveis dos Centro de Recursos são obrigatoriamente docentes, os responsáveis pelos vários serviços são funcionários. Por sua vez, os Serviços Administrativos dispõem ainda de um secretário, que não deixa de ser, também ele, funcionário.

4. Um jogo de futebol disputa-se sobre um campo de futebol e inclui duas equipas de jogadores e uma de arbitragem. Cada equipa de jogadores é composta por um guarda-redes, 4 defesas, cuja missão é atrapalhar os avançados da equipa adversária, 4 médios que desenvolvem o jogo, e por 2 avançados para marcar golos. Todos os jogadores sabem correr, fazer faltas, ralhar com o árbitro e chutar a bola. A equipa de arbitragem é composta pelo árbitro principal e por 2 auxiliares. O árbitro principal limita-se a apitar sempre que pode e a mostrar o maior número possível de cartões, de preferência vermelhos. Os auxiliares só sabem assinalar foras de jogo.

Nos 3 exercícios que se seguem (também de modelação) construa um diagrama de classes UML mais rigoroso para o problema que se apresenta.

5. Pretende-se um sistema de gestão de uma biblioteca. A biblioteca dispõe apenas de livros, que podem ser requisitados por utentes. Um livro é caraterizado por um código, título e autor. Um utente é caraterizado pelo número de utente e pelo nome. Para não complicarmos demasiado a

solução, vamos, por agora, introduzir duas importantes simplificações no problema: a biblioteca apenas dispõe de exemplares únicos e a aplicação não vai precisar de memorizar as requisições passadas (já devolvidas) – a aplicação apenas reportará o estado atual das requisições. Um utente pode realizar diversas requisições, sendo apenas possível de concretizar se o exemplar a requisitar se encontrar disponível. Quando um exemplar é devolvido deve ficar livre para futuras requisições. O sistema deve permitir o registo de novos utentes e a aquisição de novos livros, devendo ainda possibilitar a realização de consultas, nomeadamente: listagem de todos os livros, com indicação do código, título, autor e se o mesmo se encontra requisitado; listagem de um ou de todos os utentes, com indicação do número, nome e número de requisições do utente; e a consulta de todas a requisições de um dado utente.

6. Imagine que teria que implementar um sistema de gestão de conteúdos via Web (SGC), em que diversos utilizadores contribuem com mensagens e comentários. Apenas os utilizadores registados podem publicar entradas, podendo uma entrada ser uma mensagem ou um comentário. A uma mensagem podem estar associados vários comentários. Um utilizador é caraterizado pelo seu nome e por uma palavra-chave. Uma entrada é caraterizada por um identificador, um título e por um corpo de texto. O sistema deve se idealizado para permitir a inserção, pesquisa, listagem e remoção de utilizadores e entradas (mensagens e comentários). Deve ainda ser possível listar as mensagens de dado utilizador e contabilizar o número total de comentários às mensagens de um utilizador.

7. Uma escola leciona cursos, caraterizados pelo nome e nos quais se inscrevem alunos, caraterizados pelo nome e número mecanográfico. Aos alunos pode-se atribuir uma nota que obtiveram a uma determinada disciplina. Cada disciplina pertence a um curso, podendo os cursos terem várias disciplinas. Além disso, a escola tem ainda um corpo docente constituído por professores, cada um caraterizado pelo nome e categoria. Cada professor pode lecionar diversas disciplinas, assim como cada disciplina pode ser lecionada por diversos professores. Deve ser possível adicionar e remover Cursos, Professores, Disciplinas e Alunos. Deve poder atribuir-se disciplinas a professores e notas aos alunos.

Seguem-se exercícios de implementação em C++.

8. Implemente em C++ uma classe pessoa descrita pelo nome e pelo ano de nascimento. Implemente os construtores e um método para calcular a idade. Construa, por fim, um pequeno *main* que faça uso de várias instâncias dessa classe.

9. Implemente uma classe em C++ que represente um contador, em que seja possível incrementar e decrementar o seu valor. Deve também ser possível reiniciar o contador e observar o valor que o mesmo possui num determinado instante. Por fim crie um pequeno *main* que faça uso dessa classe.

10. Imagine um termómetro vulgar com uma escala limitada entre dois valores, permitindo registar a temperatura ambiente. Para o seu funcionamento, o termómetro utiliza dois controlos para aumentar e diminuir a temperatura consoante é registada. É também possível observar a

temperatura atual. O valor atual não pode ultrapassar os limites da escala. Crie uma classe que represente este termómetro e um pequeno *main* em que a mesma seja instanciada.

11. Implemente uma classe que represente um retângulo com uma posição e dimensões. Implemente métodos que permitam aceder e modificar a sua posição e as suas dimensões. Adicione ainda um método para calcular a área e crie um pequeno main que faça uso dessa classe.

12. Pegue novamente na solução do exercício 10, acrescente-lhe um *array* de termómetros, substitua as operações de aumento e diminuição de temperatura por operadores sobrecarregados (+= e -=) e passe a constante todas as entidades que considere terem essa natureza.

13. Implemente uma classe em C++ que represente um número complexo. Defina os construtores adequados e as quatro operações básicas sobrecarregando as operações tradicionais (adição, subtração, multiplicação e divisão).

14. Implemente uma classe em C++ que represente números racionais, um numerador e um denominador. Defina os construtores adequados e as quatro operações básicas sobrecarregando as operações tradicionais (adição, subtração, multiplicação e divisão).

15. Implemente uma classe Ponto representada por duas coordenadas. Implemente métodos que permitam alterar o valor das coordenadas, escrever as coordenadas na saída standard e calcular a distância entre dois pontos. De seguida implemente uma classe Circunferência, em que se representa o seu centro e o seu raio. Escreva um método para escrever os dados da circunferência na saída standard e outro para calcular o seu perímetro. Construa um pequeno *main* que faça uso das classes que definiu.

16. Construa um template em C++ que permita criar uma lista de elementos de um qualquer tipo e dimensão variável, especificados como parâmetros do template. Devem ser suportadas as operações de inserção de um elemento e remoção do último elemento inserido.

17. Defina um template de funções que lhe permita trocar o conteúdo de duas variáveis de um qualquer tipo. Instancie depois esse template de forma a trocar o conteúdo de variáveis do tipo int, double e string.

18. Crie uma coleção de strings e guarde nessa coleção sucessivas palavras inseridas pelo utilizador. Por fim mostre as strings guardadas por ordem alfabética.

19. Para o diagrama do exercício 7, implemente as classes Escola, Curso e Disciplina e as operações de adição de um curso à escola e de uma disciplina ao curso.

20. Para o diagrama do exercício 7, implemente a classe Professor e as operações de adição de um professor e atribuição de um professor a uma disciplina.

21. Para o diagrama do exercício 7, implemente as classes Aluno e Nota e as operações de atribuição de uma nota a um aluno a uma dada disciplina.

22. Implemente em C++ uma classe que represente um ponto com três dimensões. Defina um método que calcule a distância entre dois pontos e outro que permita alterar as suas coordenadas.

23. Implemente em C++ uma classe que represente uma pessoa, em que são descritos o atributo nome e a operação *print* para escrever os atributos na saída standard. Defina, de seguida, uma classe Aluno, adicionalmente caraterizada pelo número mecanográfico e outra Professor, adicionalmente caraterizada pela categoria. Ambas as classes devem escrever os atributos na saída standard na operação *print*.

24. Para as classes do exercício anterior, explorar o polimorfismo com apontadores no método *print*.

25. Implemente uma classe que permita representar um *array* de apontadores de Pessoa. Devem ser definidas operações para adicionar uma pessoa e listar as pessoas do array.

26. Implemente uma classe String que permita manipular uma cadeia de carateres, para que sejam ocultados os detalhes de representação e manipulação da string tipo C (char *). Codifique os construtores que julgue convenientes, assim como as seguintes operações: cálculo do tamanho da String, acesso individual aos seus carateres, concatenação de duas strings, comparação de strings, verificação de inclusão de strings e, por fim, a leitura e escrita de uma string pela entrada e saída standard.

27. Defina uma classe Instituição na qual trabalhem pessoas. Defina as operações de adição e remoção de um Aluno e Professor à instituição.

28. Considere o seguinte programa:

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

class Animal{
    string nome;
public:
    Animal(string n): nome(n){cout << "Animal::Animal(" << n << ")" << endl;}
    string getNome() const{ return(nome);}
    virtual void DizOla(Animal &a)=0;
    ~Animal(){ cout << "Animal::~Animal()" << endl;}
};

class Gato: public Animal{
public:
```

```
    Gato(string n): Animal(n){cout << "Gato::Gato(" << n << ")" << endl;}
    void DizOla(Animal &a){
       cout << this->getNome() << " diz miau para " << a.getNome() << endl;
    }
    ~Gato(){ cout << "Gato::~Gato()" << endl;}
};

class Cao: public Animal{
public:
    Cao(string n): Animal(n){ cout << "Cao::Cao(" << n << ")" << endl;}
    void DizOla(Animal &a){
       cout << this->getNome() << " diz au para " << a.getNome() << endl;
    }
    ~Cao(){ cout << "Cao::~Cao()" << endl;}
};

void main(){
    Gato *g1=new Gato("Mia");
    Cao *c1=new Cao("Nely");
    Gato *g2=new Gato("Garfield");
    Cao *c2=new Cao("Snoopy");
    Animal *a1=new Animal("Bobi");

    a1->DizOla(*c1);
    g1->DizOla(*g2);
    c2->DizOla(*g1);

    delete a1;
    delete g1;
    delete c1;
    delete g2;
    delete c2;
}
```

a) Identifique o tipo de relação existente entre Animal e Gato.
b) Diga se o programa é funcional, isto é, compila e executa. Justifique e, caso não seja funcional, efetue as alterações necessárias em conformidade com o restante código.
   NOTA: As alterações que eventualmente sejam efetuadas devem ser levadas em conta na resolução das restantes alíneas deste grupo.
c) Apresente o resultado que será visualizado na saída standard, após a execução do programa.
d) Considere agora as seguintes alterações no código anterior e a substituição da função main anterior:

```
class Cao: public Animal{
    …
    string getNome() const{
       string s="Cao de nome " + Animal::getNome();
       return(s);
    }
```

```
    …
};

void main(){
    Gato *g1=new Gato("Mia");
    Cao *c1=new Cao("Nely");
    g1->DizOla(*c1);
    delete g1;
    delete c1;
}
```

Qual o resultado que será visualizado na saída standard, após a execução do programa?

29. Considere o código em C++ apresentado de seguida.

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

class A{
    int a;
public:
    A(){
       a=0;
       cout << "A()" << endl;
    }
    A(int x){
       a=x*5;
       cout << "A(" << a << ")"<< endl;
    }
    int valor(){ return(a);}
    ~A(){ cout << "~A(): " << a << endl;}
};

class B: public A{
    int b;
public:
    B(int x, int y): A(y), b(x){ cout << "B(" << x << ", "<< y << ")" << endl;}
    int valor(){ return(b);}
    ~B(){cout << "~B(): " << b << endl;}
};

class C: public B{
    int c;
public:
    C(int x, int y, int z): B(y, z), c(x){
       cout << "C("<< x << ", " << y << ", " << z << ")" << endl;
    }
    int valor(){ return(c); }
    ~C(){ cout << "~C(): " << c << endl; }
};
```

```
void func(A a){
    cout << "func() invocada com: " << a.valor() << endl;
}

void main(){
    A *obj=new C(1, 2, 3);
    C c(6, 7, 8);
    cout << obj->valor() << endl;
    func(10);
    delete obj;
    cout << c.valor() << endl;
}
```

a) Verifique a existência de erros de compilação, caso existam, indique-os e corrija-os de forma a que o que programa execute. De seguida escreva o resultado na saída standard, após a execução do programa.
b) Escreva um método na classe C que, devolva a soma dos valores dos parâmetros passados no construtor C::C(int, int, int).
c) O seguinte main() tem a invocação de um método não definido. Pretende-se que o método imprima no ecrã o nome da classe onde é invocado. Proceda à alteração do código que julgue necessária, de modo a que, no exemplo abaixo, seja escrito “class C”, no ecrã.

```
void main(){
    A *obj=new C(2, 3, 4);
    obj->nome();
    delete obj;
}
```

30. Segue-se um programa em C++ que implementa um modelo muito simples de contas bancárias, com apenas algumas funcionalidades elementares. Depois de o analisar com a devida atenção, responda às questões que lhe são colocadas.

```
#include<iostream>

#include<string>

using namespace std;

class Conta{
protected:
      double saldo;
public:
      Conta(): saldo(0.0){
            cout << "Abertura de Conta ";
      }
      Conta(double s): saldo(s){
            cout << endl << "Abertura com deposito inicial " << saldo;
```

```
	}
	virtual string getTipo()const =0;
	virtual ~Conta(){cout << " fechada com saldo " << saldo <<endl;}
};

class CPrazo: public Conta{
	double txJuro;
public:
	CPrazo(double tx, double dep=0.0): Conta(dep){
		txJuro=tx;
		cout << " a Prazo ";
	}
	string getTipo()const {return " a Prazo ";};
	~CPrazo(){cout << "Conta com taxa " << txJuro;}
};

class CPEspecial: public CPrazo{
public:
	CPEspecial(): CPrazo(1.5){
		saldo+=100.0;
		cout << "Especial 100" << endl;
	}
	CPEspecial(double tx): CPrazo(tx){
		saldo+=200.0;
		cout << "Especial 200";
	}
	string getTipo()const {return "Especial ";}
	~CPEspecial(){cout << getTipo();}
};


void main(){
	CPrazo *contas[3];
	contas[0]=new CPEspecial(3.5);
	contas[1]=new CPrazo(2.5, 500.0);
	CPEspecial c;
	contas[2]=&c;
	delete contas[0];
	delete contas[1];
}
```

a) Alguma das classes do problema é abstrata? Se sim, diga qual e por que motivo a considera abstrata.
b) Diga quais os métodos construtores de que dispõe cada uma das classes (os explícitos e os implícitos).
c) Apresente o resultado que será visualizado na saída standard após a execução do programa.
d) E o que seria visualizado se o método destrutor da classe Conta não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior.

e) Como se percebe, na implementação sugerida, qualquer conta a Prazo que seja criada pode ter associada uma taxa de juro diferente das restantes. Que tipo de alteração proporia à solução apresentada, de modo a garantir que todas as contas a Prazo que viessem a ser criadas passassem a ter a mesma taxa de juro. Explique, não implemente.

f) Considerando as seguintes instanciações: `CPEspecial c1, c2;` diga, justificando, se cada uma das instruções `c1=c2;` e `c1++;` é ou não válida no problema considerado.

g) Considere agora o código substituto para a função main() que a seguir se apresenta. Identifique dois erros presentes nesse código e corrija-os efetuando apenas alterações nas classes do problema.

```
void main(){
      CPrazo contas[10];
      for(int i=0; i<10; i++) contas[i]+=10.0; //depósitos de 10 euros
}
```

31. Considere o seguinte programa:

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

class Animal{
    string nome;
public:
    Animal(string n): nome(n){}
    virtual string Especie() const =0;
};

class Gato: public Animal{
public:
    Gato(string n): Animal(n){}
    string Especie() const{ return("Gato");}
};

class Cao: public Animal{
public:
    Cao(string n): Animal(n){}
    string Especie() const{ return("Cao");}
};


void main(){
    Gato *g1=new Gato("Mia");
    Cao *c1=new Cao("Nely");
    cout << g1->Especie() << endl;
    cout << c1->Especie() << endl;
    delete g1;
    delete c1;
}
```

a) Diga se o programa é funcional, isto é, compila e executa. Justifique e, caso não seja funcional, efetue as alterações necessárias em conformidade com o restante código.
NOTA: As alterações que eventualmente sejam efetuadas devem ser levadas em conta na resolução das restantes alíneas deste grupo.

b) Considere a seguinte substituição da função main:

```
void main(){
    Gato *g1=new Gato("Mia");
    Cao *c1=new Cao("Nely");
    if(Animal::Compativeis(*c1, *g1))
       cout << "Compativeis" << endl;
    else
       cout << "Incompativeis" << endl;
    delete g1;
    delete c1;
}
```

Implemente o método int Animal::Compativeis(…) de modo que devolva verdadeiro se os animais contidos nos parâmetros forem da mesma espécie, e falso, caso contrário.

32. Considere o código em C++ apresentado de seguida:

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

class A{
    int a;
public:
    A(){
       nome();
       cout << "Construtor por omissão." << endl;
       a=0;
    }
    A(int x){
       nome();
       cout << "Construtor com 1 parametro: " << x << endl;
       a=x;
    }
    void nome(){cout << "[A]: ";}
    void operator++(int){ a++;}
    void  operator--(int){ a--;}
    ~A(){
       nome();
       cout << "Destrutor: " << a << endl;
    }
};

class B{
    A a;
```

```
    int b;
public:
    B(){
      nome();
      cout << "Construtor por omissão." << endl;
      b=0;
    }
    B(int x){
      nome();
      cout << "Construtor com 1 parametro: " << x << endl;
      b=x;
    }
    void operator++(int){
      a--;
      b++;
    }
    void nome(){ cout << "[B]: ";}
    ~B(){
      nome();
      cout << "Destrutor: " << b << endl;
    }
};

void main(){
    A *obj1=new A[2];
    B *obj2=new B(15);
    B obj3(20);
    (*obj2)++;
    obj3++;
    delete []obj1;
    delete obj2;
}
```

a) Verifique a existência de erros de compilação, caso existam, indique-os e corrija-os de forma que o que programa execute. De seguida escreva o resultado na saída standard, após a execução do programa.

33. Considere o seguinte programa:

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

class Ponto{
    int x, y;
public:
    Ponto(int x, int y){
      this->x=x;
      this->y=y;
      cout << "Ponto::Ponto(" << x << ", " << y << ")" << endl;
```

```
    }

    Ponto(const Ponto &p){
       x=p.x;
       y=p.y;
       cout << "Ponto::Ponto(const Ponto &)" << endl;
    }

    void Desenha(){
       cout << "(" << x << ", " << y << ")" << endl;
    }

    void operator+=(const Ponto &p){
       x+=p.x;
       y+=p.y;
    }

    ~Ponto(){
       cout << "Ponto::~Ponto()" << endl;
    }
};

class Figura{
proteted:
    Ponto origem;
public:
    Figura(int x, int y): origem(x, y){}

    Ponto getOrigem() const{
       return(origem);
    }

    virtual void Desenha(){
       origem.Desenha();
       cout << "Desenha Figura" << endl;
    }

    virtual ~Figura(){
       cout << "Figura::~Figura()" << endl;
    }
};

void main(){
    Figura f1(8, 9);
    f1.Desenha();
    Ponto p2=f1.getOrigem();
    p2.Desenha();
}
```

a) Identifique o tipo de relação existente entre Figura e Ponto.

b) Apresente o resultado que será visualizado na saída standard, após a execução do programa.
c) Considere agora a seguinte substituição da função main anterior:

```
void main(){
    Ponto p1(5, 7);
    Figura f1(p1);
    f1.Desenha();
}
```

Existe um erro neste programa. Corrija-o sem alterar a função main.
d) Apresente o resultado que será visualizado na saída standard, após a execução do programa da alínea anterior, depois de resolvido o erro anterior.

34. Considere o seguinte código adicional ao código da questão anterior e a nova função main:

```
class Retangulo: public Figura{
    int comp, larg;
public:
    Retangulo(int x, int y, int c, int l): Figura(x, y){
       comp=c; larg=l;
    }
    void Desenha(){
       Figura::Desenha();
       cout << "Desenha Retangulo: " << comp << ", " << larg << endl;
    }
    void operator+=(const Figura &f){
       origem+=f.getOrigem();
    }
    virtual ~Retangulo(){
       cout << "Retangulo::~Retangulo()" << endl;
    }
};

class Circulo: public Figura{
    int raio;
public:
    Circulo(int x, int y, int r): Figura(x, y){raio=r;}
    void Desenha(){cout << "Desenha Circulo: " << raio << endl;}
    virtual ~Circulo(){ cout << "Circulo::~Circulo()" << endl;}
};


void main(){
    Figura *f1=new Retangulo(10, 10, 30, 40);
    Figura *f2=new Circulo(20, 20, 50);
    f1->Desenha();
    f2->Desenha();
    delete f1;
    delete f2;
}
```

a) Identifique o tipo de relação existente entre Retangulo e Figura.
b) Apresente o resultado que será visualizado na saída standard, após a execução do programa.
c) Considere a seguinte substituição da função main:

```
void main(){
    Retangulo f3(2, 3, 10, 15);
    Circulo f4(11, 13, 20);
    f3+=f3;
    f4+=f3;
}
```

Existe um erro neste programa. Corrija-o sem alterar a função main.

d) Apresente o resultado que será visualizado na saída standard, após a execução do programa da alínea anterior, depois de resolvido o erro anterior.

35. Considere o seguinte programa:

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

class Pessoa{
      protected:
            string _nome;
            int _nbi;
      public:
            Pessoa(){_nbi=0; cout << "1\n";}
            Pessoa(string n,int b):_nome(n){_nbi=b; cout << "2\n";}
            virtual ~Pessoa(){cout<< "4\n";}
            string getNome(){return _nome;}
            virtual int getID(){return _nbi;}
            friend ostream &operator<<(ostream&,const Pessoa&);
};

ostream &operator<<(ostream &o,const Pessoa &c){
      o << "Nome:" << c._nome << endl;
      o << "Nº BI:" << c._nbi << endl;
      return o;
}


class Condutor : public Pessoa{
      protected:
            int _ncarta, _idade;
      public:
            Condutor(){_ncarta=_idade=0; cout << "5\n";}
            Condutor(string n,int b,int nc, int i)
                  :Pessoa(n,b),_ncarta(nc),_idade(i){cout << "6\n";}
            virtual ~Condutor(){cout<< "8\n";}
```

```
            string getNome(){return "Condutor(a) " + _nome;}
            int getID(){return _ncarta;}
            friend ostream &operator<<(ostream&,const Condutor&);
};

ostream &operator<<(ostream &o,const Condutor &c){
      o << (Pessoa&)(c);
      o << "N. de carta:" << c._ncarta << endl;
      o << "Idade:" << c._idade << endl;
      return o;
}

void main(){
      Condutor ze("Ze",11122233,555666,25);
      Pessoa maria("Maria",222333444), *p=&ze;
      // Substituir esta linha pelo código adicional
}
```

a) Identifique o tipo de relação existente entre Pessoa e Condutor.

b) Diga se o programa é funcional, isto é, compila e executa. Justifique e, caso não seja funcional, efetue as alterações necessárias em conformidade com o restante código.
NOTA: As alterações que eventualmente sejam efetuadas devem ser levadas em conta na resolução das restantes alíneas deste grupo.

c) Apresente o resultado que será visualizado na saída standard, após a execução do programa.

d) Apresente o resultado que será visualizado na saída standard, mas apenas para as seguintes linhas de código:

```
/* ************************************************************** */
      // Código adicional
      cout << "ID:" << maria.getNome() << endl;
      cout << "ID:" << maria.getID() << endl;
      cout << "ID:" << ze.getNome() << endl;
      cout << "ID:" << ze.getID() << endl;
      cout << "ID:" <<  p->getNome() << endl;
      cout << "ID:" <<  p->getID() << endl;
/* ************************************************************** */
```

e) Apresente o resultado que será visualizado na saída standard, mas apenas para as seguintes linhas de código:

```
/* ************************************************************** */
      // Código adicional
      cout << maria << endl;
      cout << ze << endl;
      cout << *p << endl;
/* ************************************************************** */
```

36. Considere as seguintes alterações ao código inicial das classes *Pessoa* e *Condutor* do exercício anterior:

```
/* ************************************************************** */
class Pessoa{
            ...
      public:...
            // Devolve a idade do condutor
            virtual int getIdade() const =0;
            ...
};

class Condutor : public Pessoa{
            ...
      public:...
            // Devolve a idade mínima de um qualquer condutor
            static int getIdadeMinima(){return 18;}
            ...
};

void main(){
      Condutor ze("Ze",11122233,555666,25);
      Pessoa *p=&ze;
      // Colocar aqui o código adicional
}
/* ************************************************************** */
```

a) Diga se o programa conforme está é funcional, isto é, compila e executa. Justifique e, caso não seja funcional, efetue as alterações necessárias em conformidade com o restante código.
NOTA: As alterações que eventualmente sejam efetuadas devem ser levadas em conta na resolução das restantes alíneas deste grupo.

b) Para cada uma das linhas de código que segue, diga:
   i) Se é ou não funcional;
   ii) Em caso afirmativo, apresente o valor visualizado na saída;
   iii) Em caso negativo, justifique porquê.

```
/* ************************************************************** */
      // Código adicional
      cout << "Idade:" <<  ze.getIdade() << endl;
      cout << "Idade mínima:" << ze.getIdadeMinima() << endl;
      cout << "Idade mínima:" << ze::getIdadeMinima() << endl;
      cout << "Idade mínima:" << Condutor::getIdadeMinima() << endl;
      cout << "Idade:" <<  p->getIdade() << endl;
      cout << "Idade mínima:" << p->getIdadeMinima() << endl;
/* ************************************************************** */
```

c) Efetue os acréscimos/alterações necessários nas classes apresentadas para que o seguinte código seja executável:

```
/* ************************************************************** */
      // Código adicional
      int idade=ze;
      cout << idade << endl;
```

```
	if(ze>=18) cout << "O Zé é maior de idade\n";
	cout << "O Zé tem " << *ze << "anos\n";
	cin >> ze;
	cout << ze << endl;
/* ************************************************************** */
```

37. O programa que se segue implementa um sistema de gestão (muito simples) de um prédio habitacional formado por habitações e garagens.

```
class Fracao{
protected:
	char id;
	int permilagem;
public:
	Fracao(char i, int p): id(i), permilagem(p){
		cout << "Criada Fracao " << id;
	}
	virtual void mostrar(){
		cout << id << " com permilagem "<< permilagem << endl;
	}
	virtual ~Fracao(){}
};

class Habitacao: public Fracao{
public:
	Habitacao(char i): Fracao(i, 40){ cout << " do tipo Habitacao" << endl;}
	void mostrar(){
		cout << "Habitacao ";
		Fracao::mostrar();
	}
	~Habitacao(){ cout << "Removida Habitacao "<< id << endl;}
};

class Garagem: public Fracao{
public:
	Garagem(char i): Fracao(i, 12){ cout << " do tipo Garagem" << endl;}
	void mostrar(){
		cout << "Garagem ";
		Fracao::mostrar();
	}
	~Garagem(){ cout << "Removida Garagem "<< id <<endl;}
};

class Predio{
	static const int MAXFRAC=20;
	Fracao *fracoes[MAXFRAC];
	int nfrac;
public:
	Predio():nfrac(0){
```

```
            cout << "Criado Predio." << endl;
      }
      void addGaragem(char i){
            if(nfrac<MAXFRAC) fracoes[nfrac++]=new Garagem(i);
      }
      void addHabitacao(char i){
            if(nfrac<MAXFRAC) fracoes[nfrac++]=new Habitacao(i);
      }
      void mostrarFracoes(){
            cout << endl << "Fracoes do Predio:" << endl;
            for(int i=0; i<nfrac; i++) fracoes[i]->mostrar();
      }
      ~Predio(){}
};

void main(){
      Predio p;
      p.addHabitacao('A');
      p.addGaragem('K');
      p.addHabitacao('B');
      p.mostrarFracoes();
}
```

a) Identifique o tipo de relação existente entre Predio e Habitacao, entre Fracao e Garagem, e entre Predio e Fracao.
b) Alguma das classes é abstrata?
c) Numa das classes o destrutor não tem a implementação mais adequada. Diga qual é esse destrutor e dê-lhe a implementação desejada.
d) Apresente o resultado que será visualizado na saída standard após a execução do programa e depois de resolvida a inconformidade referenciada na alínea anterior.
e) E o que seria visualizado se o método Fracao::Mostrar() não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior.
f) Admita agora que o prédio passou a poder incorporar, para além de habitações e garagens, lojas comerciais com diferentes permilagens. Acrescente ao programa os métodos e/ou classes necessárias, para que passe a refletir esta nova realidade.