UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA E TELECOMUNICAÇÕES

Exame Prático de Paradigmas de Programação I

16/Janeiro/2002 (duração uma hora).

1. **Construtor de cópia**.

Marcar todas as linhas onde o construtor de cópia da classe `my_class` é chamado

```
my_class& my_function(my_class* mcp)
{    /*...*/ mcp = new my_class;
     // .....
     return *mcp;   }
void main(int argc, char* argv[])
{    my_class my_object;
     my_class &my_object_copy = my_object;
     my_class *my_pointer=0;
     my_object_copy = my_function(my_pointer);
     my_class my_ob = my_object_copy;
// ......
}
```

2. **Destrutor virtual**.

A classe `derived_mc` é derivada da classe `my_class` (`class derived_mc : public my_class`). Que destrutores serão chamados nas linhas seguintes se **A)** destrutor da classe `my_class` for virtual; **B)** destrutor da classe `my_class` não for virtual.

A)
```
my_class *dmc = new derived_mc;
delete dmc;
```

B)
```
my_class *dmc = new derived_mc;
delete dmc;
```

3. **Redefinição de operadores** (2 perguntas).

3.1. O operador `operator=` é definido só para a classe `my_class`. Verificar as seguintes linhas. Se existirem alguns erros marque as respectivas linhas e explique o(s) erro(s). Marque também a(s) linha(s) onde a função `operator=` é chamada

```
derived_mc derived_object;
my_class *dmc = new derived_mc;
derived_object = *dmc;
*dmc = derived_object;
```

3.2. Apresente um exemplo que demonstra como redefinir o operador `operator()`. Por exemplo, para aceder ao elemento i,j de um array bidimensional usando apenas um índice.

4. **Classes bases e derivadas**.

Assume-se que temos as seguintes classes bases e derivadas:

```
class my_class
{public:  void display() { cout << "my_class\n";  }
//...
};
class derived_mc : public my_class
{public:  void display() { cout << "derived_mc\n"; }
```

```
//...
};
class derived1_mc : public my_class
{    /* ...  */          };
class derived_derived :  public derived_mc,
                         public derived1_mc
{    /* ... */      };
```

Para o código seguinte remova a(s) linha(s) que têm erros e mostre os resultados que vão aparecer no ecrã para todas as linhas correctas.

```
derived_mc derived_object;
derived1_mc derived_object1;
derived_derived dd_object;
derived_object.display();
derived_object1.display();
dd_object.display();
```

5. **Classes abstractas**.

Assume-se que temos as seguintes classes bases e derivadas:

```
class my_class
{public:  virtual void f() = 0;
//...
};
class derived_mc : public my_class
{public:  void f(){/*...*/};
//...
};
class derived_derived :  public derived_mc
{    /* ... */      };
```

O seguinte código está correcto ou não? Explique porquê.

```
derived_derived dd;
dd.f();
```

6. **Templates**.

Crie o código para a classe template `pilha`. Assume-se que a `pilha` pode ser usada para guardar só os seguintes tipos: `int`, `float`, `double`, `char`.

7. **Controlo de excepções**.

Utilizar o mecanismo de controlo de excepções para encontrar os erros que podem aparecer nas funções `push` (colocar na pilha) e `pop` (remover da pilha). Ver o exemplo do template no ponto 6.

8. **Entrada e saída de dados**.

Apresentar um exemplo que demonstra como redefinir manipuladores de formato sem argumentos. Apresentar um exemplo que demonstra como escrever/ler dados para os ficheiros.

9. **Herança e agregação**.

Apresentar um exemplo de herança e um exemplo de agregação.

Peso *2 valores* para cada pergunta.