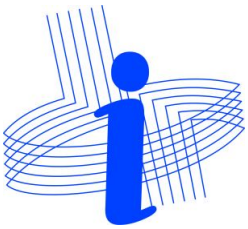

UNIVERSIDADE FEDERAL DE VIÇOSA
DEPARTAMENTO DE INFORMÁTICA
INF213 - Estrutura de Dados
Primeira prova -- 05/04/2018
Prof. Salles Magalhaes

Aluno:

Matricula:

Obs: nesta prova assuma que o uso de “includes” e de diretivas do tipo “using namespace std;” não seja necessário.

1) (40%) Nesta questão você deverá implementar uma classe Matriz para representar matrizes quadradas de tipos arbitrários.

Exemplo de uso da sua classe:

```
int main() {
	Matriz<int> x; //por padrao, x tera dimensoes 10x10
	Matriz<int> m(4);
	for(int i=0;i<4;i++)
		for(int j=0;j<4;j++)
			m.set(i,j,1);
	m.print();

	Matriz<int> m2(4);
	for(int i=0;i<4;i++)
		for(int j=0;j<4;j++)
			m2.set(i,j,3);

	(m+m2).print();

	Matriz<string> ms(2);
	ms.set(0,0,"abc");
	ms.set(0,1,"x");
	ms.set(1,0,"a");
	ms.set(1,1,"w");
	ms.print();
}
```

Saida esperada:

```
1 1 1 1
1 1 1 1
1 1 1 1
```

1 1 1 1
4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
abc x
a w

Por simplicidade, assuma que:
- Sua classe não precisa validar o acesso aos dados (i.e., você não precisa verificar se o usuário está tentando acessar uma posição inválida da matriz).
- Você não precisa se preocupar em inicializar os elementos da sua matriz ao construí-la.
- O usuário nunca tentará somar matrizes de tamanhos diferentes.

Sua classe deverá possuir pelo menos os seguintes métodos (adicione outros métodos que forem necessários):
- Método get: dadas as coordenadas l,c (linha, coluna) da matriz retorna o elemento nessa posição.
- Método set: dadas as coordenadas l,c (linha, coluna) da matriz e um valor, atribui o valor a posição l,c da matriz.
- O operador de adição: retorna a soma de duas matrizes.
- Um método print(): imprime os elementos da matriz na saída padrão. (observe que os operadores << e >> não precisam ser implementados na sua classe).

Observe que o construtor da sua classe deverá suportar a criação de matrizes com tamanho padrão (4x4) e matrizes com um tamanho definido pelo usuário (veja o exemplo acima).

Escreva a seguir a declaração completa da sua classe e a implementação de cada método, **exceto** os métodos print() e set() (implemente os métodos apenas após o final da declaração da classe). Ou seja, você deverá incluir aqui todo o código que estaria no arquivo de cabeçalho da sua classe. Lembre-se de utilizar boas práticas de engenharia de software (por exemplo, usar *const* quando adequado):

2 (10%) - Indique a ordem de complexidade de cada função abaixo (Obs: você não precisa justificar sua resposta, sempre use a ordem de complexidade “mais justa”):

```
double f1(int n) {
        double  x = 0;
        for(int i=0;i<n/2;i++)
          for(int j= n/2; j>=0;j--)
                x+= log(2*i*j);
        return x;
}
```

Resposta:

```
int f2(int n) {
    if(n==0) return 0;
    return 1+f2(n-1);
}
```

Resposta:

```
double f3(int n) {
        double ans = 0;
        for(int i=0;i<n;i++) {
                for(int j=0;j<n;j++)
                        ans -= log(i*j);
                ans += log(n+i);
                for(int j2=0;j2<n/2;j2++)
                        for(int k=n/2;k<n;k++)
                                ans += log(j2*k);
        }
        return ans;
}
```

Resposta:

```
double f4(int v[], int n, int x) {
        for(int i=0;i<n;i++) {
                if(v[i]==x) {
                        int ct = 0;
                        for(int j=i+1;j<n;j++)
                                if(v[j]>x) ct++;
                        return ct;
                }
        }
        return 0;
}
```

Resposta:

3 (26%, cada erro deduz 3 pontos da nota) - Indique se cada afirmação abaixo é verdadeira ou falsa.

( ) $100n+20 \in \Omega(n^2)$

( ) $n^2 \log n \in O(n^3)$

( ) $4 \times 2^n \in O(2^n)$

( ) Se $f(n) \in O(g(n))$ então $f(n)+g(n) \in O(g(n))$

( ) $2^{n+1} \in O(2^n)$

( ) $2^{2n} \in O(2^n)$

( ) Atribuir o endereço de um objeto de uma classe Base a um ponteiro da classe derivada é um erro de compilação.

( ) Atribuir o endereço de um objeto de uma classe Base a um ponteiro da classe derivada usando o dynamic_cast faz com que o programa falhe em tempo de execução.

( ) Atribuir o endereço de um objeto de uma classe derivada a um ponteiro de sua classe base gera um erro de compilação.

( ) Atribuir o endereço de um objeto de uma classe derivada a um ponteiro de sua classe base gera um erro em tempo de execução.

( ) Para sobrecarregar o operador << e' preciso faze-lo ser friend da classe.

( ) Se uma classe base não possui construtor padrão, a classe derivada deve chamar o construtor da classe base de forma explícita em seu construtor.

( ) Os membros de dados protegidos (protected) de uma classe só podem ser alterados por funções implementadas na classe X ou por funções friend da classe X.

4 - (valor: 5%) (Poscomp 2015) O conceito de encapsulamento de programação orientada a objetos pode ser implementado na linguagem Java por meio de: (Obs: C++ e' similar)

(A) métodos estáticos (static) e públicos (public).

(B) métodos públicos (public), privados (private) e protegidos (protected).

(C) classes abstratas (abstract) e métodos protegidos (protected).

(D) interfaces (interface), métodos públicos (public) e métodos protegidos (protected).

(E) herança (extends) e métodos estáticos (static).

5 - (valor: 5%) (Poscomp 2012) O encapsulamento dos dados tem como objetivo ocultar os detalhes da implementação de um determinado módulo. Em linguagens orientadas a objeto, o ocultamento de informação é tornado explícito requerendo-se que todos os métodos e atributos em uma classe tenham um nível particular de visibilidade com relação às suas subclasses e às classes clientes. Em relação aos atributos de visibilidade, assinale a alternativa correta.

a) Um atributo ou método público é visível a qualquer classe cliente e subclasse da classe a que ele pertence.

b) Um atributo ou método protegido é visível somente à classe a que ele pertence, mas não às suas subclasses ou aos seus clientes.

c) Um atributo ou método privado é visível somente às subclasses da classe a que ele pertence.

d) Um método protegido não pode acessar os atributos privados declarados na classe a que ele pertence, sendo necessária a chamada de outro método privado da classe.
e) Um método público pode acessar somente atributos públicos declarados na classe a que ele pertence.

6 - (valor: 7%) (POSCOMP 2011) Em linguagens orientadas a objetos, o polimorfismo refere-se à ligação tardia de uma chamada a uma ou várias implementações diferentes de um método em uma hierarquia de herança. Neste contexto, considere as seguintes classes descritas na Linguagem C++.

```
class PosComp1
{
public:
	int Calcula() { return 1; };
};

class PosComp2 : public PosComp1 {
public:
	virtual int Calcula() { return 2; }
};

class PosComp3 : public PosComp2 {
public:
	int Calcula() { return 3; }
};
```

Se estas classes forem utilizadas a partir do programa a seguir

```
int main() {
	int Result=0;
	PosComp1 *Objs[3];
	Objs[0] = new PosComp1();
	Objs[1] = new PosComp2();
	Objs[2] = new PosComp3();
	for (int i=0; i<3; i++)
		Result += Objs[i]->Calcula();
	cout << Result << endl;
	return 0;
}
```

A saída desse programa será:
a) 0
b) 3
c) 5
d) 6
e) 9

7 - (valor: 7%) (POSCOMP 2010) O mecanismo de herança, no paradigma da programação orientada a objetos, é uma forma de reutilização de software na qual uma nova classe é criada, absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas. Considere as seguintes classes descritas na linguagem C++.

```
class A {
protected:
        int v;
public:
        A() { v = 0; };
        void m1() { v += 10; m2(); };
        void m2() { v += 20; };
        int getv() { return v; };
};

class B : public A {
public:
        void m2() { v += 30; };
};
```

Se essas classes forem utilizadas a partir do programa a seguir,

```
int main() {
        B *Obj = new B();
        Obj->m1();
        Obj->m2();
        cout << Obj->getv() << endl;
        return 0;
}
```

a saída do código computacional acima será:

a) 30
b) 40
c) 50
d) 60
e) 70