Ponteiros

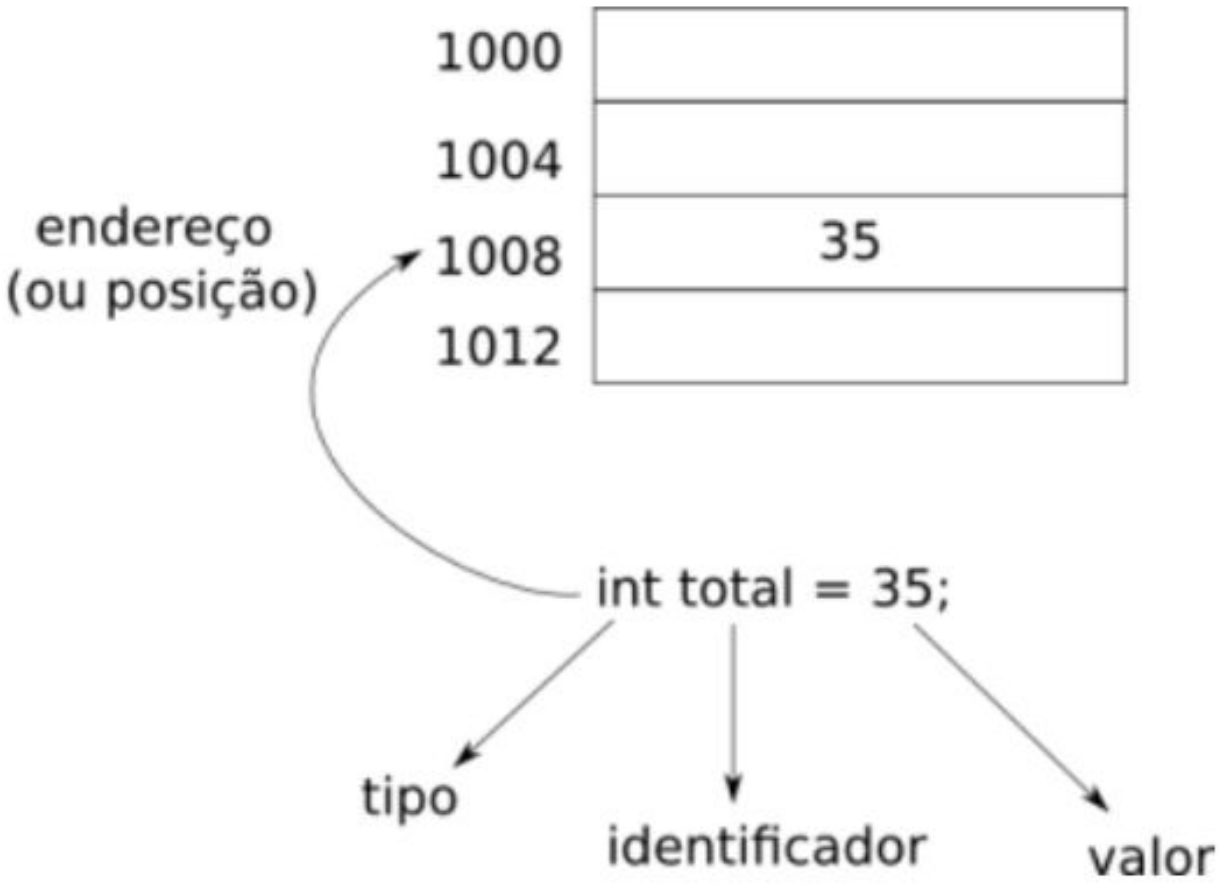

ufjf
Universidade
Federal de Juiz de Fora


GETCOMPUTAÇÃO


Departamento de
Ciência da Computação

Ponteiros

- char: 1 byte
- bool: 1 byte
- short: 2 bytes
- int: 4 bytes
- float: 4 bytes
- double: 8 byte

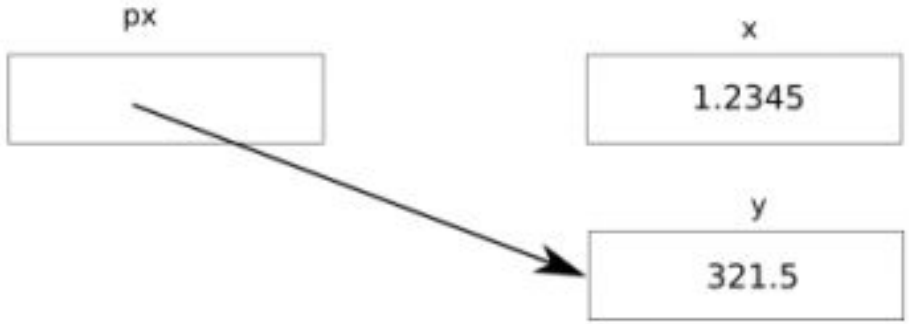


Ponteiros

```
double * px;
double x = 5.1234;
double y;
px = &x;
```


```
// altera o valor do objeto apontado por px
*px = 1.2345;
```


```
// faz px apontar para outro objeto
px = &y;
*px = 321.5;
```



Aritmética de ponteiros

```
int *p = NULL;
int vet[3] = {10, 20, 30};

p = &vet[0];

cout << "enderecos" << endl;
cout << p << endl;
cout << p+1 << endl;
cout << p+2 << endl;

cout << "valores" << endl;
cout << vet[0] << endl;
cout << vet[1] << endl
cout << vet[2] << endl;
```

Passagem por referência

```
void troca(int *a, int *b)
{
  int aux = *b;
  *b = *a;
  *a = aux;
}

int main()
{
  int x=2, y=30;
  troca(&x, &y);
  cout << "x = " << x << " , ";
  cout << "y = " << y << endl;
  return 0;
}
```

Alocação dinâmica

- Exemplo:

```
//alocando e deletando uma variável
dinamicamente
int * variavel = new int;
delete variavel;

//alocando e deletando um vetor dinamicamente
int * vetor = new int [tamanho];
delete [] vetor;
```

Recursividade

Recursividade

- Em um algoritmo recursivo, o problema original é dividido e uma ou mais versões simples de si mesmo.
- Uma solução recursiva tem as seguintes características:
  - Deve-se conhecer a solução direta para um valor pequeno de n //caso base
  - Um problema de um dado tamanho n pode ser dividido em uma ou mais versões menores do mesmo problema //caso recursivo

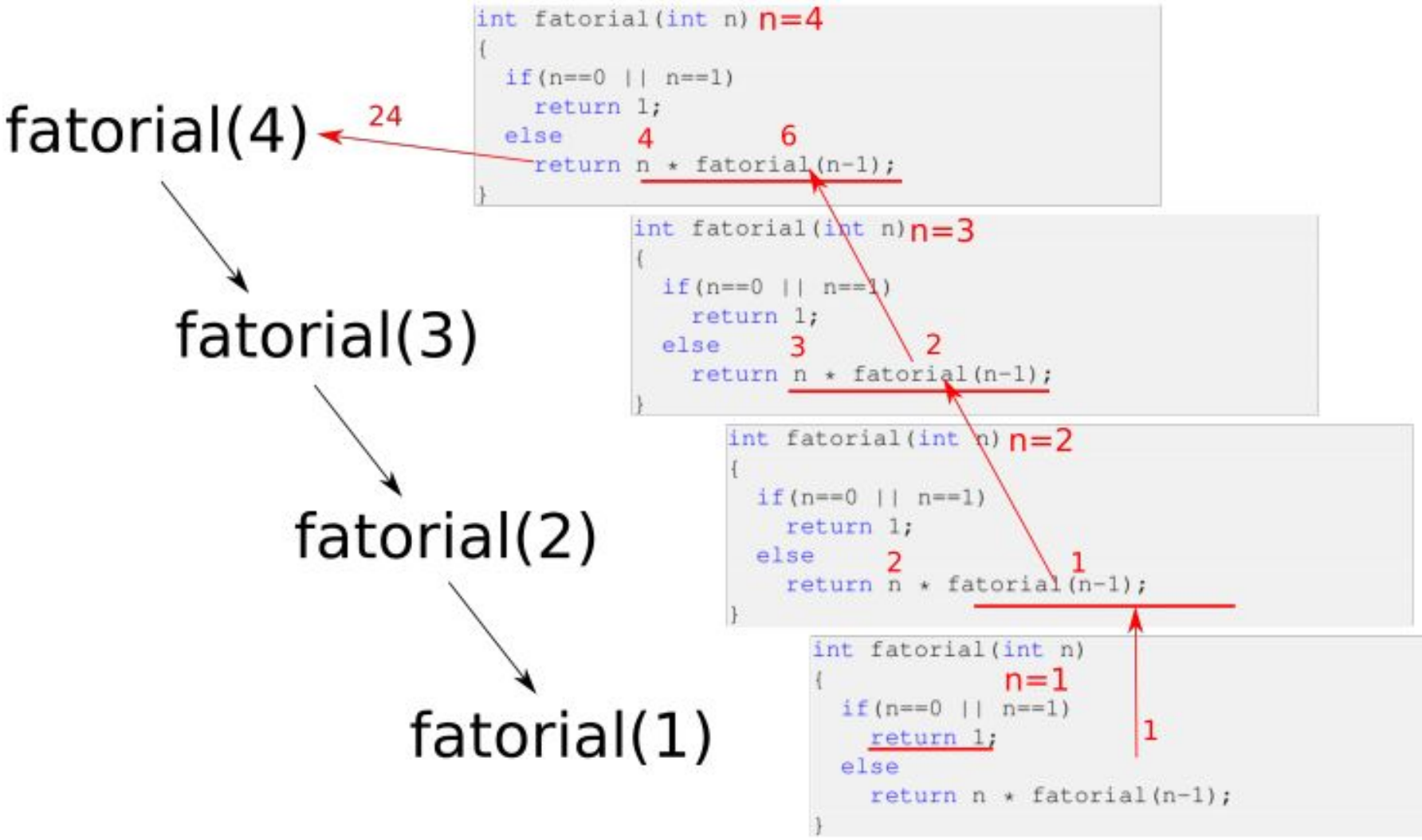
Recursividade

Exemplo: Fatorial de n = n! = 1*2 * 3 *...* (n-1) * n

Implementação recursiva:

```
int fatorial(int n)
{
if(n==0 || n==1)     //caso base
    return 1;
else
    return n * fatorial(n-1);   //caso recursivo
}
```

Recursividade



Complexidade

- Caso n > 1

```
int n, c, fat = 1;
cout << "Digite n" << endl;                    // 1
cin >> n;                                       // 1
if(n >= 0)                                      // 1
{
  for(c = 1; c<=n; c++)                         // n+1
    fat = fat * c;                              // n
  cout << "Fatorial = " << fat << endl;         // 1
}
else
  cout << "Valor negativo" << endl;
```

- Soma das frequências = 2n + 5

Complexidade

FUNÇÃO DE COMPLEXIDADE	n (tamanho do problema)		
	20	40	60
n	0.0002 s	0.0004 s	0.0006 s
$n \log_2 n$	0.0009 s	0.0021 s	0.0035 s
n^2	0.0040 s	0.0160 s	0.0360 s
n^3	0.0800 s	0.6400 s	2.1600 s
2^n	10.0000 s	27 dias	3660 séculos
3^n	580 minutos	38550 séculos	$1.3*10^{14}$ séculos

Tipos Abstratos de Dados

Tipos Abstratos de Dados

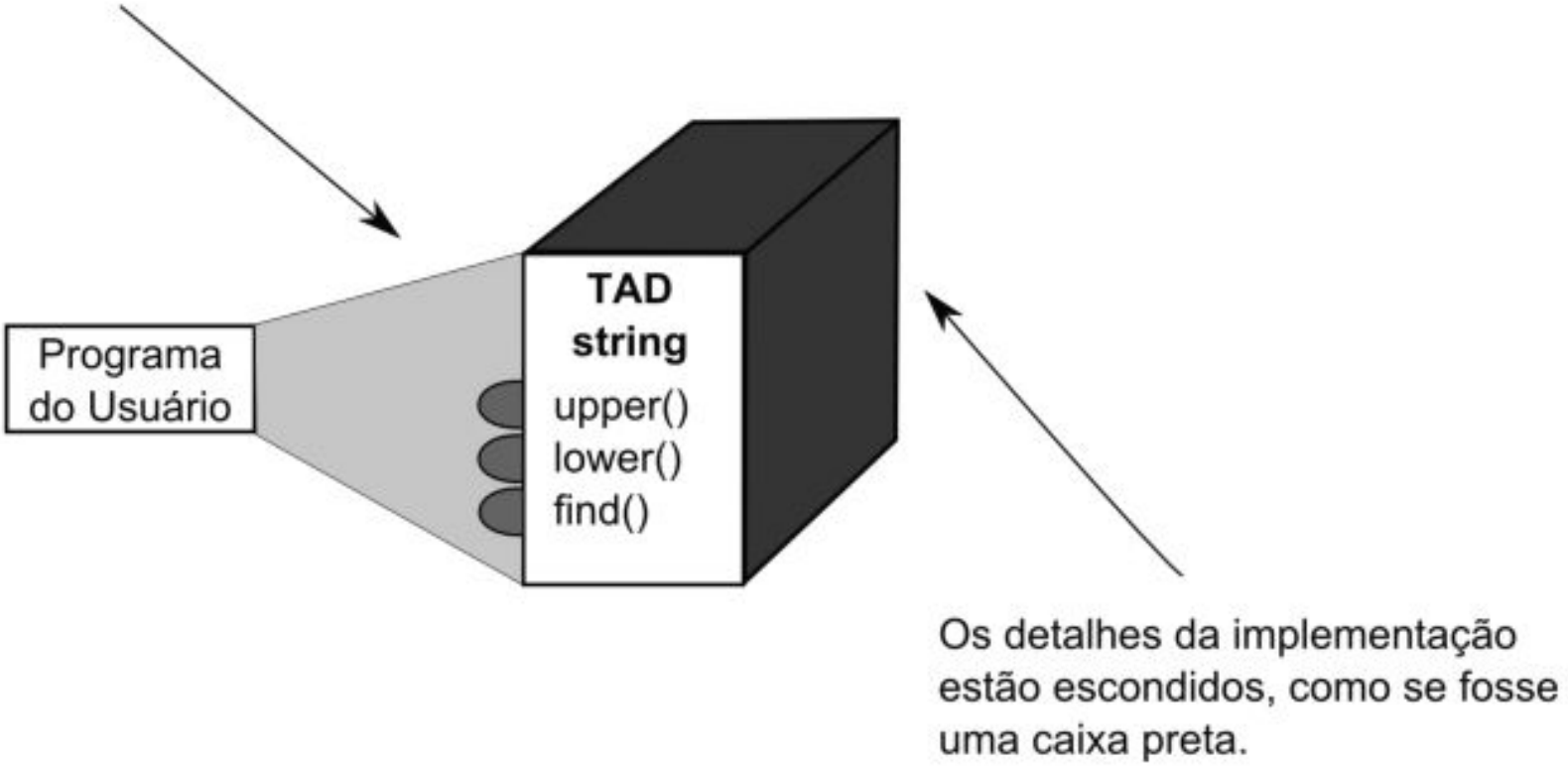
- Tipos de dados definidos pelo usuário que satisfazem as propriedades de invisibilidade, proteção e encapsulamento.

- Será usado o conceito de classe para criar o TAD.

```
class nomeDaClasse
{
    // ... corpo da classe ...
};
```

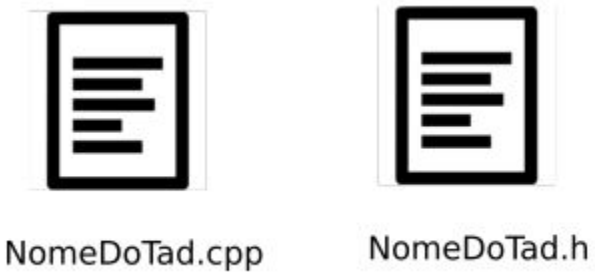
Tipos Abstratos de Dados

Os programas do usuário interagem com um TAD através da sua interface



Tipos Abstratos de Dados

- Por padrão, as operações são implementadas fora da classe no módulo de implementação (.cpp)



nomeDoTad**.h** : definição da classe
NomeDoTad**.cpp:** implementação das funções da classe

Tipos Abstratos de Dados

```
class Aluno
{
  public:
    Aluno(string n, string m);
    ~Aluno();

    void info();
    float getNota();
    string getNome();
    void setNota(float valor);
    bool verificaAprovado();

  private:
    string nome;
    string matricula;
    float nota;
};
```

Invisibilidade e proteção

Tipos Abstratos de Dados

```
Aluno::Aluno(string n, string m) {
  nome = n;
  matricula = m;
}

Aluno::~Aluno() {
  cout << "Destruindo aluno: " << nome << endl;
}

float Aluno::getNota() {
  return nota;
}

void Aluno::setNota(float valor) {
  cout << "Alterando nota do aluno" << endl;
  nota = valor;
}
```

Matrizes

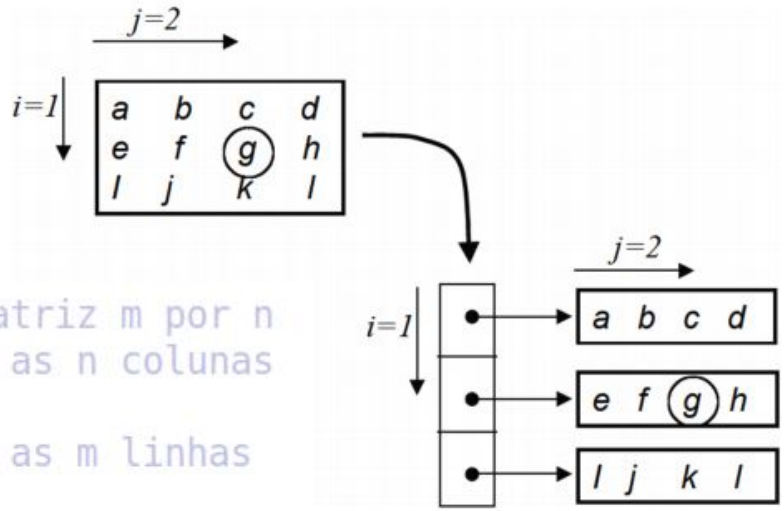

ufjf
UNIVERSIDADE
FEDERAL DE JUIZ DE FORA


GETCOMPUTAÇÃO


Departamento de
Ciência da Computação

Matrizes

- Representando matrizes usando ponteiro de ponteiros:



- Alocando uma nova matriz:

```
float **matriz;                 //criando matriz m por n
matriz = new float*[n];         //definindo as n colunas
for(int i = 0; i < m; i++)
    matriz[i] = new float[m];   //definindo as m linhas
```

- Desalocando:

```
for(int i = 0; i < m; i++)
    delete [] matriz[i];        //deletando as m linhas
delete [] matriz;
```

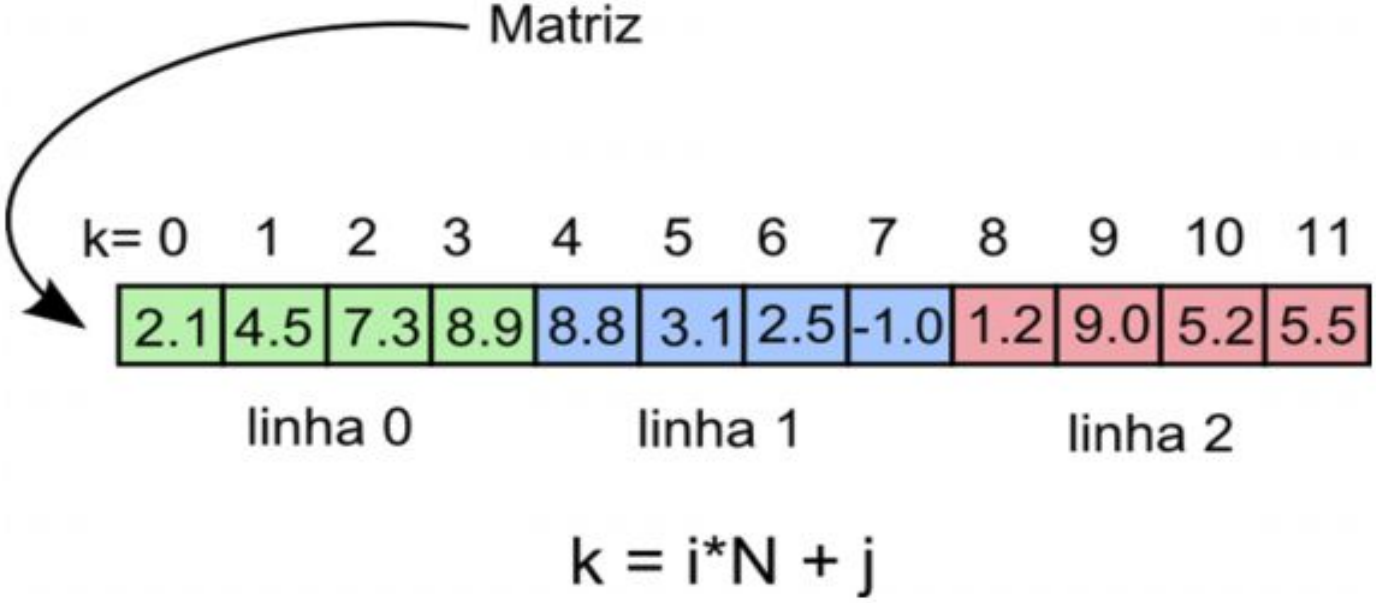
Matrizes

- Representando matrizes de forma linear

	j=0	j=1	j=2	j=3
i=0	2.1	4.5	7.3	8.9
i=1	8.8	3.1	2.5	-1.0
i=2	1.2	9.0	5.2	5.5

M=3 linhas
N=4 colunas

Matriz



$$k = i*N + j$$

Matrizes

```
class MatrizLin
{
    public:
        MatrizLin(int m, int n);
        ~MatrizLin();
        float get(int i, int j);
        void set(int i, int j, float val);

    private:
        int nl, nc;
        float *vet;
        bool verificaInd(int linha, int coluna);
};
```

Matrizes

```
MatrizLin::MatrizLin(int m, int n)
{
    nl = m;
    nc = n;
    vet = new float[nl*nc];
}
MatrizLin::~MatrizLin()
{
    delete [] vet;
}

bool MatrizLin::verificaInd(int i, int j)
{
    if(i >= 0 && i < nl && j >= 0 && j < nc)
        return true;
    else
        return false;
}
```

Matrizes

```
float MatrizLin::get(int i, int j)
{
    if(verificaInd(i, j)){
        int k = i*nc + j;
        return vet[k];
    }
    else {
        cout << "Erro: get" << endl;
        exit(1);
    }
}
void MatrizLin::set(int i, int j, float valor)
{
    if(verificaInd(i, j)){
        int k = i*nc + j;
        vet[k] = valor;
    }
    else {
        cout << "Erro: set" << endl;
        exit(1);
    }
}
```