



Uma introdução à Orientação por Objetos

Ponteiro para objetos
Construtores e Destrutores
Membros Estáticos

Opções para codificar o corpo do método

Métodos

C++ permite que as *funções-membro* possam ser descritas fora do escopo da classe.

```
class Data
{
    private :
        int dia;
        int mes;
        int ano;

    public :
        char* mesExtenso();
        bool dataValida();
        void leData();
        void escreveData();
        bool setDia(int dia);
        int getDia();
        bool setMes(int mes);
        int getMes();
        void setAno(int ano);
        int getAno();
        bool setData(int dia, int mes, int ano);
        Data getData();
};
```

```

class Data
{
    private :
        int dia;
        int mes;
        int ano;

    public :
        char* mesExtenso();
        bool dataValida();
        void leData();
        void escreveData();
        bool setDia(int dia);
        int getDia();
        bool setMes(int mes);
        int getMes();
        void setAno(int ano);
        int getAno();
        bool setData(int dia, int mes, int ano);
        Data getData();
};

```

```

char* Data::mesExtenso()
{
    char* mes[]={ "janeiro", "fevereiro",
                  "marco", "abril", "maio", "junho",
                  "julho", "agosto", "setembro",
                  "outubro", "novembro", "dezembro" };
    return mes[ (this->mes-1) ];
}

bool Data::dataValida()
{
    bool valida=true;
    if(this->dia < 0 || this->dia > 31)valida=false;
    else if(this->mes < 0 || this->mes > 12)valida=false;
    return valida;
}

// Baseado nas discussões ao estudarmos Est. Seleção,
// amplie o escopo de validação de uma data pelo
// número de dias previstos para cada mês do ano,
// inclusive para os casos de ano bissexto

```

```

class Pessoa
{
    private :
        string nome;
        Data nascimento;
    public :
        void setNome(string nome);
        string getNome();
        bool setNascimento(int dia,int mes,int ano);
        Data getNascimento();
        void lePessoa();
        void escrevePessoa();
};

void Pessoa::setNome(string nome)
{
    this->nome= nome;
}

string Pessoa::getNome()
{
    return this->nome;
}

```

```

bool Pessoa::setNascimento(int dia, int mes, int ano){
    this-> nascimento.setData(dia,mes,ano);
}

Data Pessoa::getNascimento(){
    return this->nascimento;
}

void Pessoa::leiaPessoa(){
    string nome;
    cout << "\nNome: ";
    getline(cin, nome);
    setNome(nome);
    cout << "\nData de nascimento: ";
    this->nascimento.leData();
}

void Pessoa::escrevePessoa(){
    cout << "\nNome: " << getNome();
    cout << "\nData de Nascimento: ";
    nascimento.escreveData();
}

```

Ponteiros para objetos

Declarando um objeto

Vimos que em C++, a instrução abaixo cria um objeto diretamente:

```
Pessoa PESSOA;
```

PESSOA

nome

idade

Acessando propriedades do objeto

Vimos que isto permite escrever códigos como estes:

Pessoa PESSOA;

PESSOA

nome

idade

PESSOA.nome = "Pedro"; //se *nome* for público

PESSOA.setNome("Pedro"); //se *nome* for privado e *setNome()* público

Ponteiro para objetos

Entretanto, C++ nos permite criar variáveis que não identificam um objeto propriamente dito. Ao contrário, ao invés de serem objetos, são ponteiros para objetos.

```
Pessoa* PESSOA;
```

```
// Declara um ponteiro para Pessoa
```

Ponteiro para objetos

A criação de um novo objeto em si fica destinada a uma outra instrução, encarregada de alocar o espaço de memória necessário para representar os valores de uma nova instância.

A instrução **new** cumpre este papel. É ela similar à instrução *malloc()* no sentido de alocar o espaço de memória e devolver o endereço do novo objeto criado.

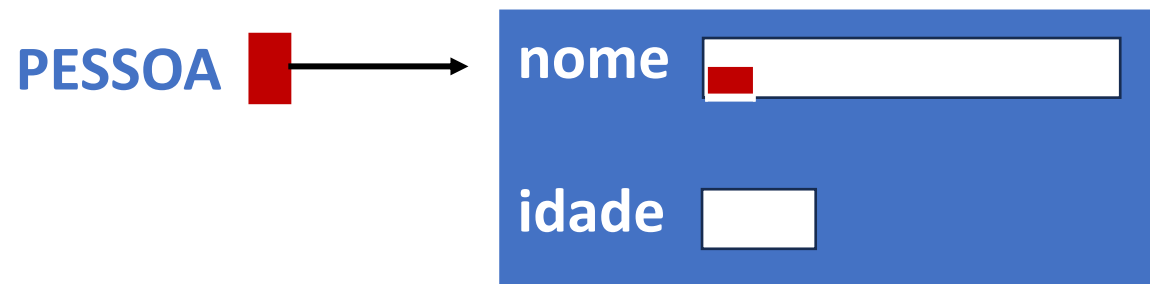
```
Pessoa* PESSOA;           // Declara um ponteiro para Pessoa
```

```
PESSOA = new Pessoa(); // Entrega ao ponteiro o endereço do novo objeto
```

Ponteiro para objetos

Como previsto na sintaxe da linguagem, declarando e atribuindo em uma mesma linha de código:

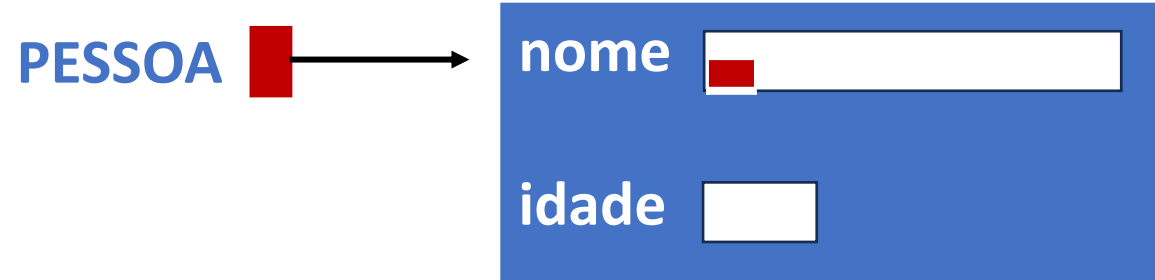
```
Pessoa* PESSOA = new Pessoa();
```



Manipulando um ponteiro para objeto

Um ponteiro para objetos nos permite escrever códigos como estes:

```
Pessoa* PESSOA = new Pessoa();
```



```
PESSOA->nome = "Pedro"; //se nome for público
```


```
PESSOA->setNome("Pedro"); //se nome for privado e setNome() público
```

Uma coleção de ponteiros para objetos

Um vetor de ponteiros para objetos

```
Pessoa* PESSOA[3];
```

PESSOA [0]  → NULL

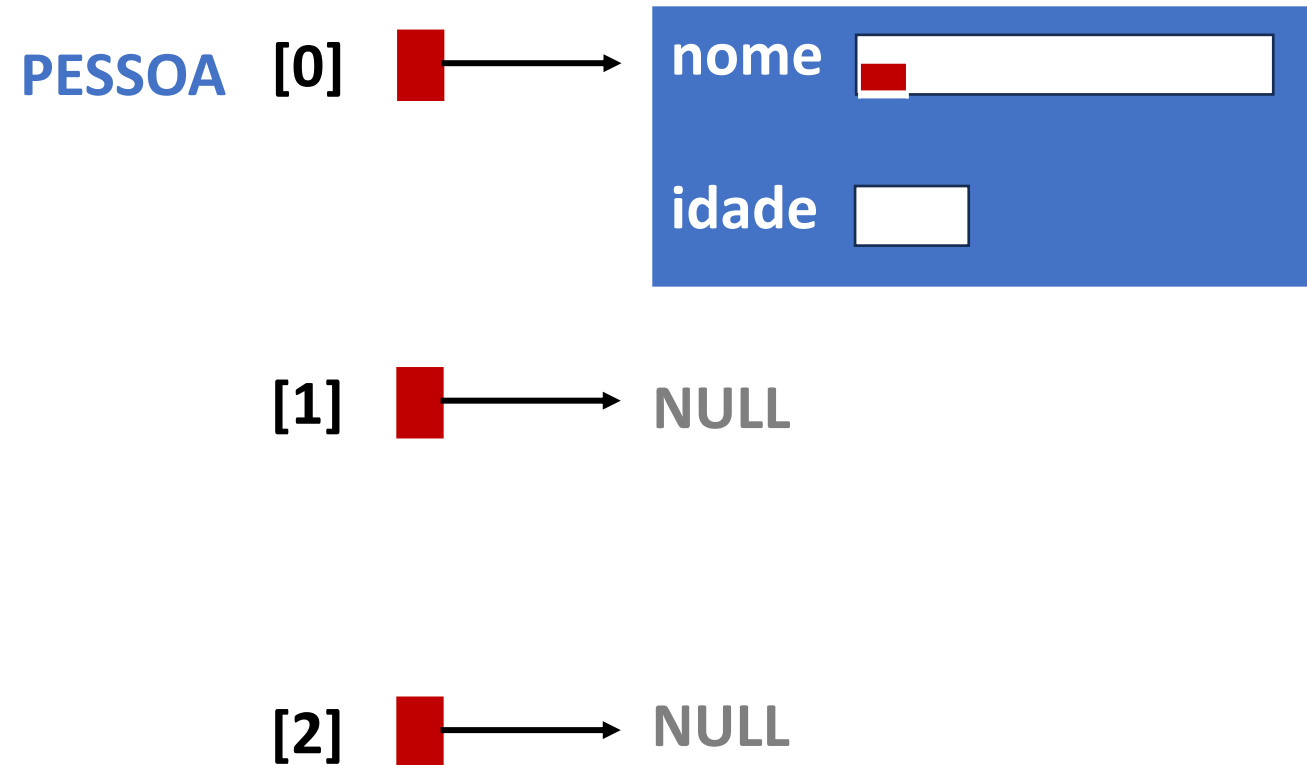
[1]  → NULL

[2]  → NULL

Um vetor de ponteiros para objetos

```
Pessoa* PESSOA[3];
```

```
PESSOA[0] = new Pessoa();
```

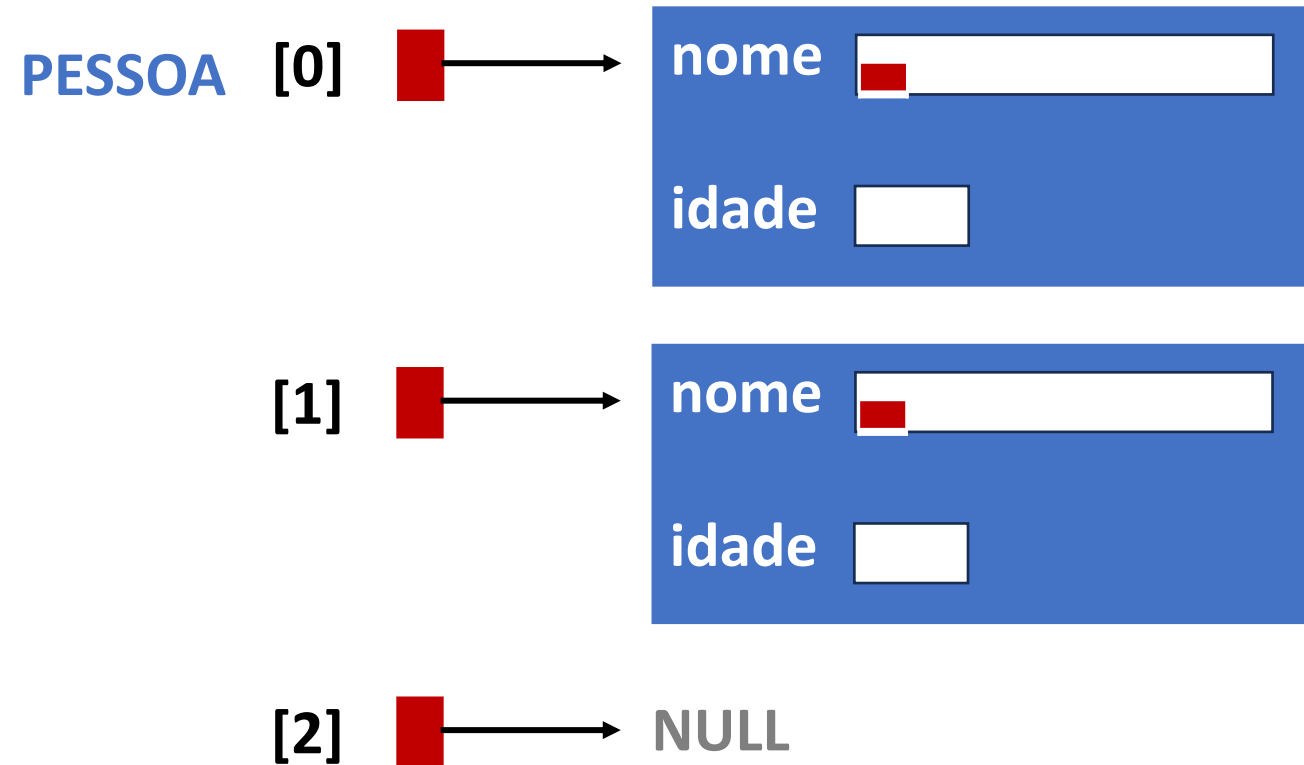


Um vetor de ponteiros para objetos

```
Pessoa* PESSOA[3];
```

```
PESSOA[0] = new Pessoa();
```

```
PESSOA[1] = new Pessoa();
```



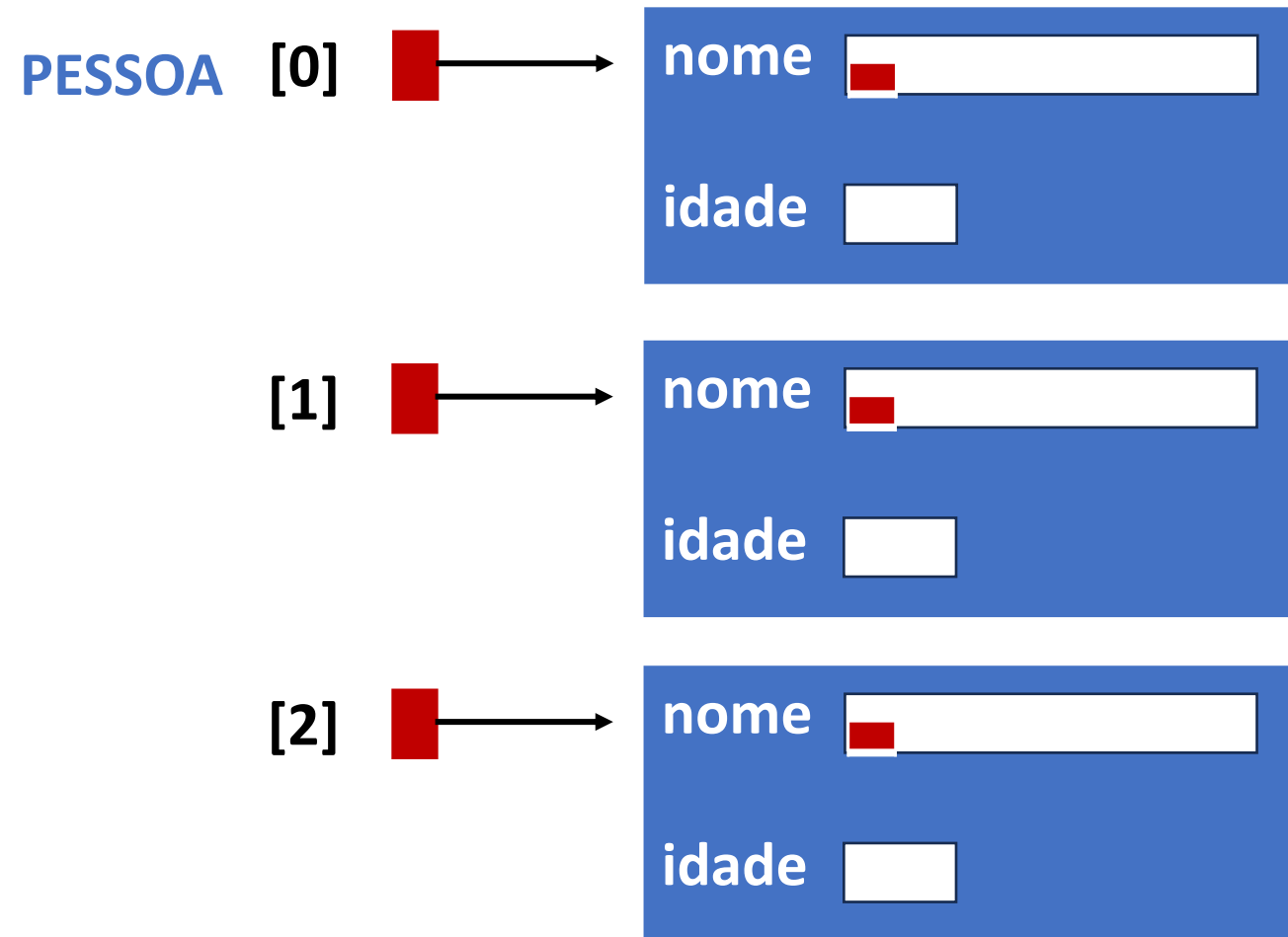
Um vetor de ponteiros para objetos

```
Pessoa* PESSOA[3];
```

```
PESSOA[0] = new Pessoa();
```

```
PESSOA[1] = new Pessoa();
```

```
PESSOA[2] = new Pessoa();
```



Manipulando um vetor de ponteiros para objetos

Um vetor de ponteiros para objetos nos permite escrever códigos como estes:

```
Pessoa* PESSOA[3];
```

```
PESSOA[0] = new Pessoa();
```

```
PESSOA[1] = new Pessoa();
```

```
PESSOA[2] = new Pessoa();
```

```
PESSOA [0] -> setNome("Pedro");
```

```
PESSOA [1] -> setNome("Ana");
```

```
PESSOA [2] -> setNome("Lucas");
```

PESSOA [0]  →

nome	<input type="text"/>
idade	<input type="text"/>

[1]  →

nome	<input type="text"/>
idade	<input type="text"/>

[2]  →

nome	<input type="text"/>
idade	<input type="text"/>

Um método muito especial:
*ele é executado sem ser chamado – ele entra
em cena assim que é criada uma nova
instância da classe*

Construtor

Método executado sempre que uma nova instância for criada.

Não se aplica o conceito de tipo do método por não haver chamada explícita.

Efeito colateral: Implementa uma regra para criação dos objetos daquela classe.

...

```
Data* data1 = new Data(21, 11, 2024);
```

```
Data* data2 = new Data(); // Erro!
```

...

```
class Data
```

```
{
```

```
    private :
```

```
        int dia;
```

```
        int mes;
```

```
        int ano;
```

```
    public :
```

```
        Data(int dia, int mes, int ano) {
```

```
            setData(dia,mes,ano);
```

```
        }
```

```
        void setData(int dia, int mes, int ano) {
```

```
            setData(dia);
```

```
            setMes(mes);
```

```
            setAno(ano);
```

```
        }
```

```
        ...
```

```
};
```

Solução:
Construtor sobrecarregado

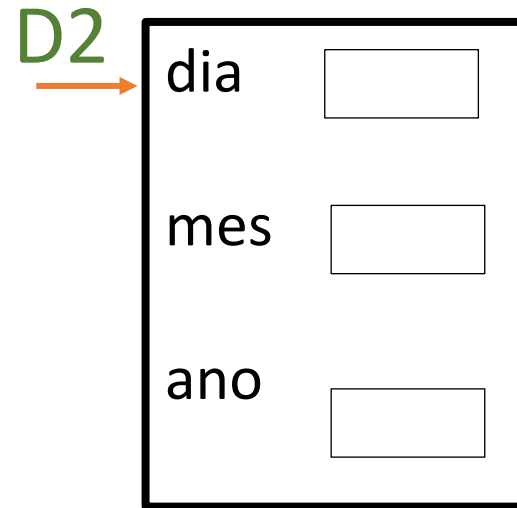
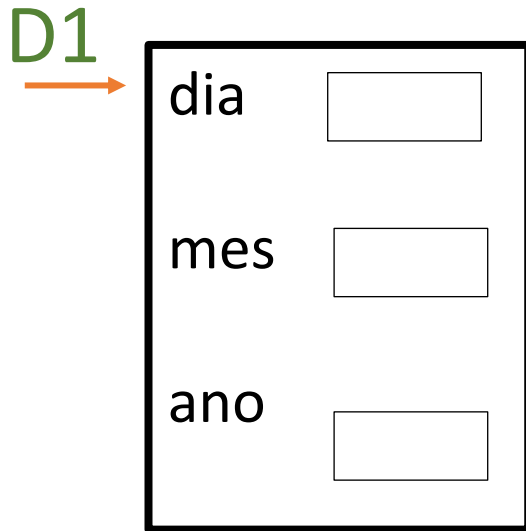
```
...  
Data* data[2];  
  
data[0] = new Data();  
  
data[1] = new Data(21, 11, 2024);  
  
...
```

```
class Data {  
    private :  
        int dia;  
        int mes;  
        int ano;  
  
    public :  
        Data() {  
            setData(0, 0, 0);  
        }  
        Data(int dia, int mes, int ano) {  
            setData(dia,mes,ano);  
        }  
        void setData(int dia, int mes, int ano) {  
            setDia(dia);  
            setMes(mes);  
            setAno(ano);  
        }  
        ...  
};
```

Construtores

```
Data* D1 = new Data();
```

```
Data* D2 = new Data(21,11,2024);
```



```
class Data
{
    private :
        int dia;
        int mes;
        int ano;

    public :
        Data()
        {
            dia=mes=ano= 0;
        }
        Data(int dia, int mes, int ano)
        {
            this->setData(dia, mes, ano);
        }
        ...
};
```

Destrutor

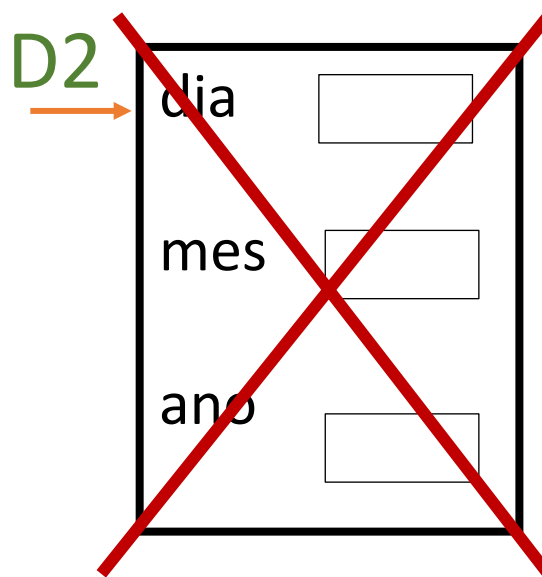
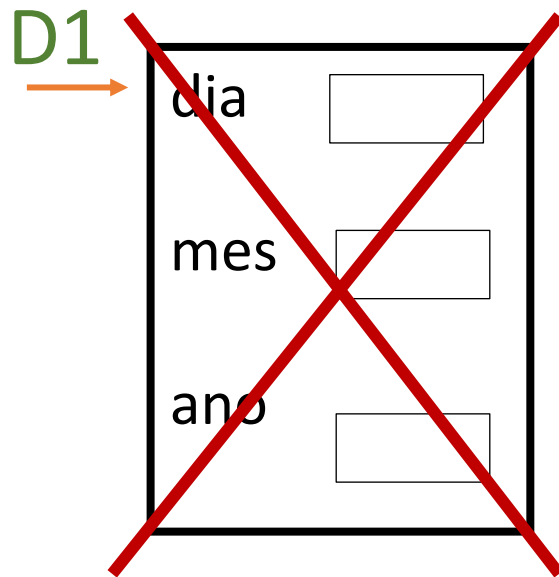
Método executado sempre que uma instância for destruída.

O operador delete é utilizado para destruir o objeto – note que a variável continua

```
class Data {  
    private :  
        int dia;  
        int mes;  
        int ano;  
    public :  
        Data() {  
            dia=mes=ano= 0;  
        }  
        Data(int dia, int mes, int ano) {  
            setData(dia, mes, ano);  
        }  
        ~Data(){  
            ...  
        }  
};  
  
void funcao()  
{  
    Data* D1 = new Data();  
    Data* D2 = new Data(21,11,2024);  
    ...  
    delete D1;  
    delete D2;  
}
```

Destruutores

delete D1;
delete D2;



```
class Data
{
    private :
        int dia;
        int mes;
        int ano;

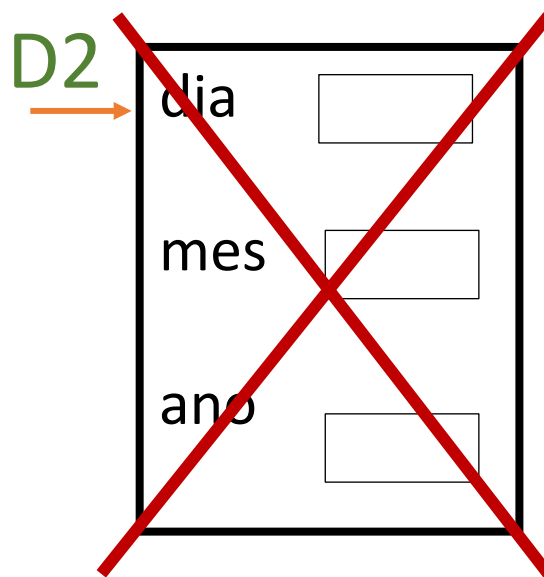
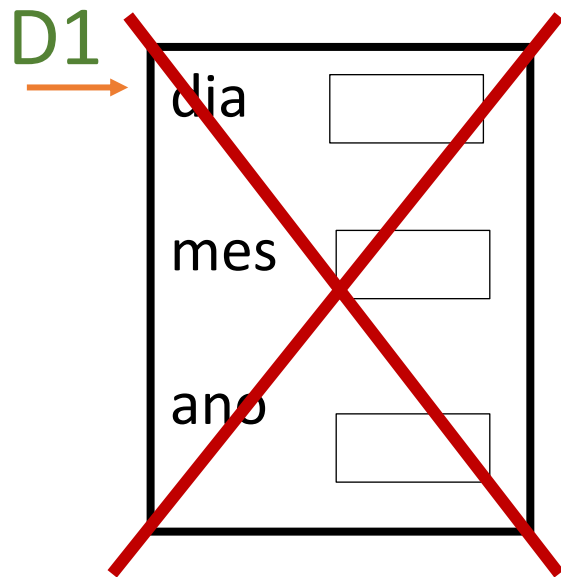
    public :
        ~Data()
        {
            printf("\nMorri!\n");
        }
        Data()
        {
            dia=mes=ano= 0;
        }

        ...

};
```


Destrutores

delete D1;
delete D2;



```
class Data
{
    private :
        int dia;
        int mes;
        int ano;

    public :
        ~Data()
        {
            printf("\n\aMorri em %p", this);
        }
        Data()
        {
            dia=mes=ano= 0;
        }

        ...

};
```

Postar no Verde
Concurso – 1 ponto
Lista 38