



Programação

C++

Aula 06
Jorgiano Vidal

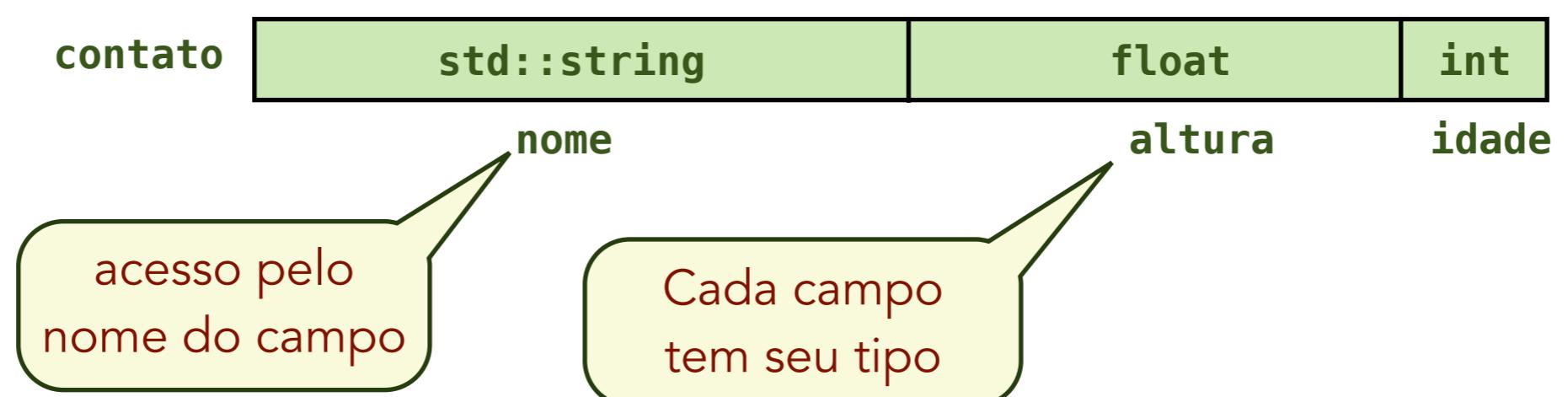


- ❖ Tipos compostos: struct/class
- ❖ Membros
- ❖ Acesso
- ❖ Visibilidade
- ❖ Construtor
- ❖ Métodos
- ❖ Destruitor



Struct / class

- ❖ Tipo de dados complexo
 - ❖ Diferentes dados
 - ❖ Campos/atributos
 - ❖ Também conhecido como registro
- ❖ Define novo **tipo**
 - ❖ Conta bancária, contato de uma agenda, aluno de uma instituição, etc
- ❖ Estrutura de dados **heterogêneas**
 - ❖ Diferentes tipos
 - ❖ Possui membros/campos
 - ❖ Dado é acessado pelo nome do campo/atributo





Definição

- ❖ Definição de novo tipo
 - ❖ Composição outros tipos
- ❖ Sintaxe:

```
struct NOME{  
    TIP01 membro01;  
    TIP02 membro02, membro03;  
};
```

```
class NOME{  
    TIP01 membro01;  
    TIP02 membro02, membro03;  
};
```

```
struct data{  
    int dia, mes, ano;  
};
```

```
class hora{  
    short hora, minuto;  
};
```

```
class aluno{  
    long long matricula;  
    double nota01;  
    double nota02;  
    double recuperacao;  
};
```



Definição

- ❖ Definição de novo tipo
 - ❖ Composição outros tipos
- ❖ Sintaxe:

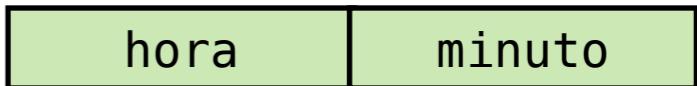
```
struct NOME{  
    TIP01 membro01;  
    TIP02 membro02, membro03;  
};
```

```
class NOME{  
    TIP01 membro01;  
    TIP02 membro02, membro03;  
};
```

```
struct data{  
    int dia, mes, ano;  
};
```



```
class hora{  
    short hora, minuto;  
};
```



```
class aluno{  
    long long matricula;  
    double nota01;  
    double nota02;  
    double recuperacao;  
};
```



Definição

- ❖ Definição de novo tipo
 - ❖ Composição outros tipos
- ❖ Sintaxe:

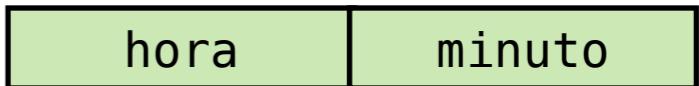
```
struct NOME{  
    TIP01 membro01;  
    TIP02 membro02, membro03;  
};
```

```
class NOME{  
    TIP01 membro01;  
    TIP02 membro02, membro03;  
};
```

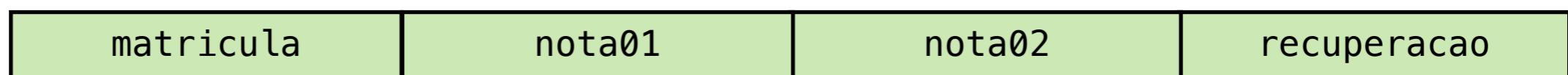
```
struct data{  
    int dia, mes, ano;  
};
```



```
class hora{  
    short hora, minuto;  
};
```



```
class aluno{  
    long long matricula;  
    double nota01;  
    double nota02;  
    double recuperacao;  
};
```





Variáveis e atribuição

* Declaração

```
struct data hoje, amanha;  
struct hora inicio, fim;  
struct aluno ana, jose;
```

hoje

amanha

inicio

fim

ana

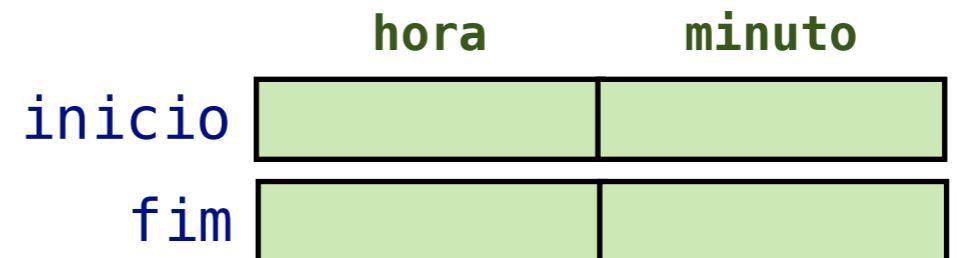
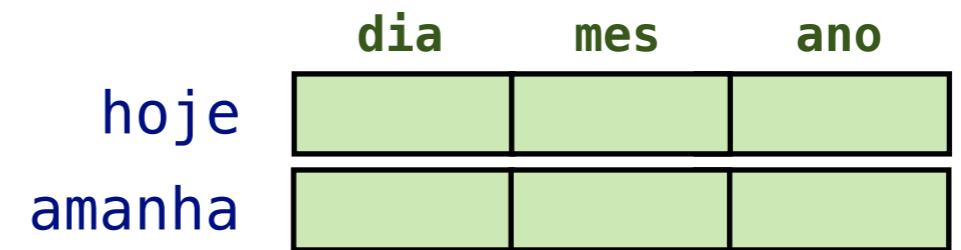
jose



Variáveis e atribuição

* Declaração

```
struct data hoje, amanha;  
struct hora inicio, fim;  
struct aluno ana, jose;
```



ana

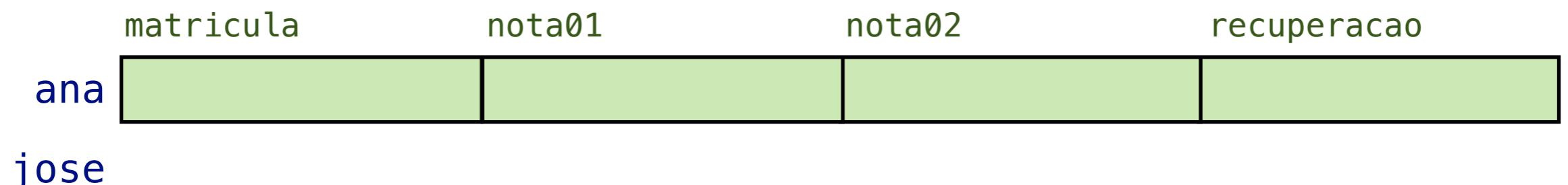
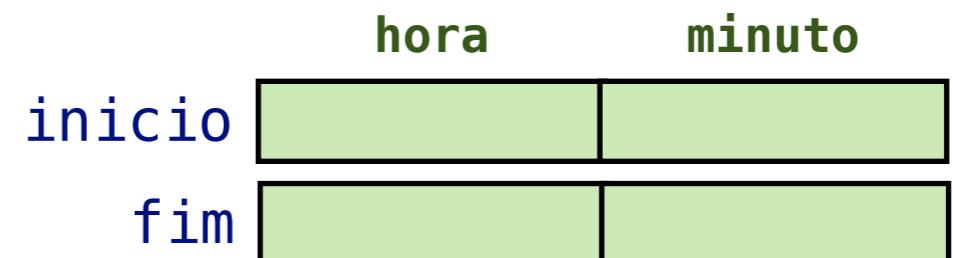
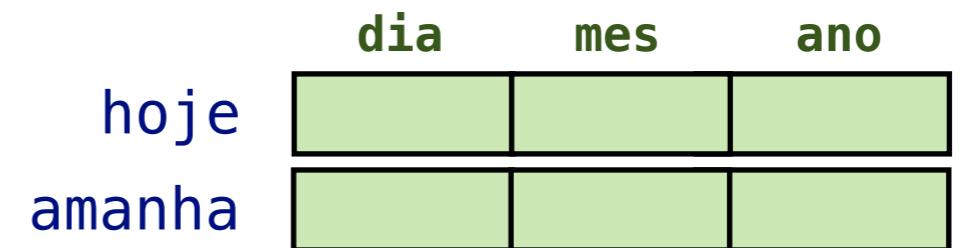
jose



Variáveis e atribuição

* Declaração

```
struct data hoje, amanha;  
struct hora inicio, fim;  
struct aluno ana, jose;
```

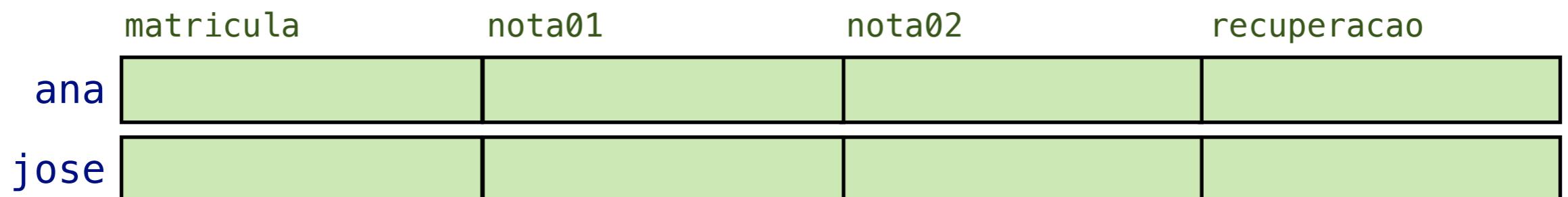
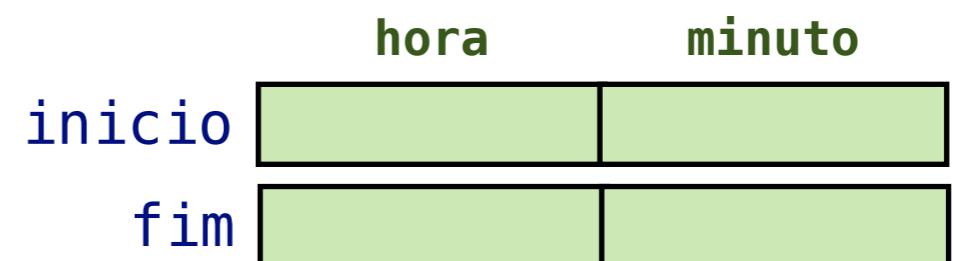
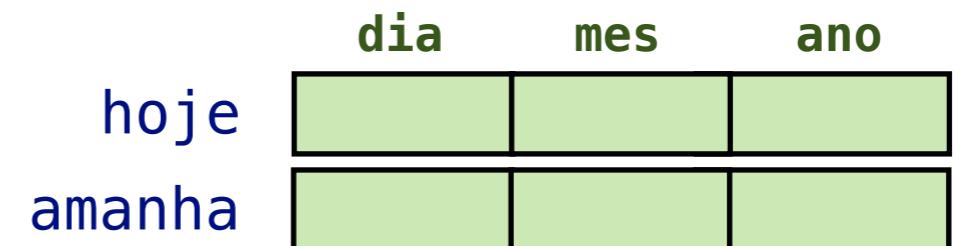




Variáveis e atribuição

* Declaração

```
struct data hoje, amanha;  
struct hora inicio, fim;  
struct aluno ana, jose;
```





Variáveis e atribuição

- * Acesso a membros

- * Operador ponto:

- var.campo

```
data hoje,amanha;  
...  
hoje.ano = 1889;  
hoje.mes = 11;  
hoje.dia = 15;  
amanha.ano = hoje.ano;  
amanha.mes = hoje.mes;  
amanha.dia = hoje.dia+1;
```

15 11 1889

16 11 1889



Variáveis e atribuição

- * Acesso a membros

- * Operador ponto:
var.campo

```
data hoje,amanha;  
...  
hoje.ano = 1889;  
hoje.mes = 11;  
hoje.dia = 15;  
amanha.ano = hoje.ano;  
amanha.mes = hoje.mes;  
amanha.dia = hoje.dia+1;
```

	dia	mes	ano
hoje	15	11	1889
	16	11	1889



Variáveis e atribuição

- * Acesso a membros

- * Operador ponto:

- var.campo

```
data hoje,amanha;  
...  
hoje.ano = 1889;  
hoje.mes = 11;  
hoje.dia = 15;  
amanha.ano = hoje.ano;  
amanha.mes = hoje.mes;  
amanha.dia = hoje.dia+1;
```

	dia	mes	ano
hoje	15	11	1889
amanha	16	11	1889



✿ Cópia de área de memória

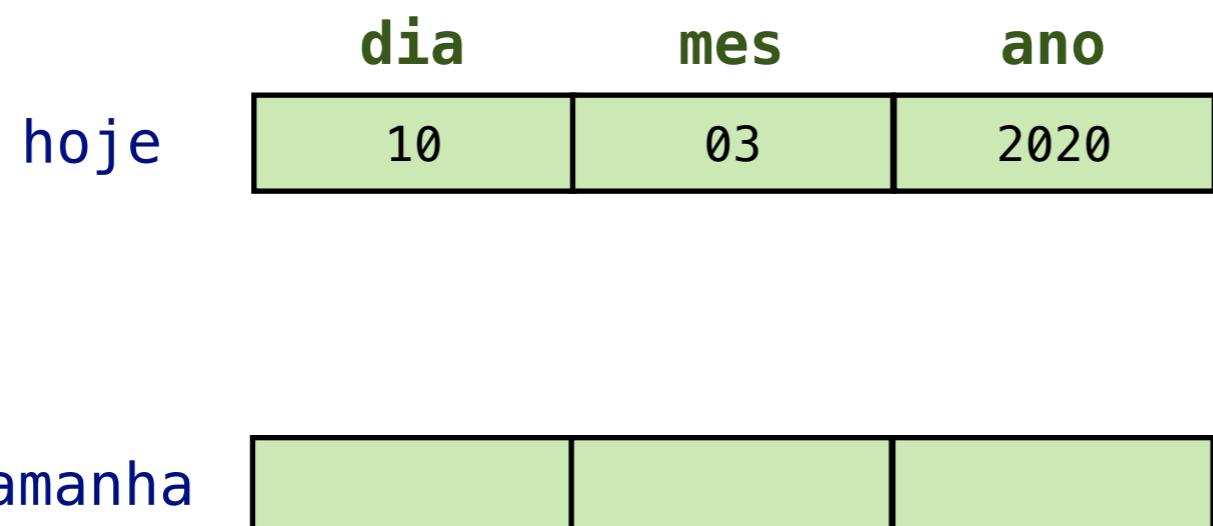
```
data hoje,amanha;  
...  
amanha = hoje;  
amanha.dia = amanha.dia + 1;
```



Variáveis e atribuição

✿ Cópia de área de memória

```
data hoje,amanha;  
...  
amanha = hoje;  
amanha.dia = amanha.dia + 1;
```





Variáveis e atribuição

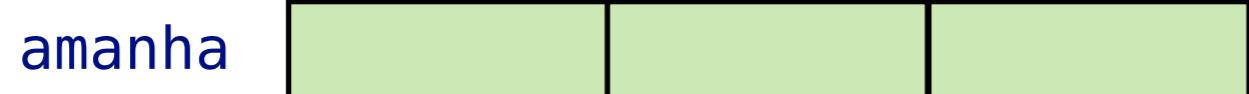
✿ Cópia de área de memória

```
data hoje,amanha;
```

```
...  
amanha = hoje;
```

```
amanha.dia = amanha.dia + 1;
```

	dia	mes	ano
hoje	10	03	2020





Variáveis e atribuição

✿ Cópia de área de memória

```
data hoje,amanha;
```

```
...  
amanha = hoje;
```

```
amanha.dia = amanha.dia + 1;
```

	dia	mes	ano
hoje	10	03	2020

amanha	10	03	2020
--------	----	----	------



Variáveis e atribuição

✿ Cópia de área de memória

```
data hoje,amanha;
```

```
...
amanha = hoje;
amanha.dia = amanha.dia + 1;
```

	dia	mes	ano
hoje	10	03	2020
amanha	10	03	2020

	dia	mes	ano
hoje	10	03	2020
amanha	10	03	2020



Variáveis e atribuição

✿ Cópia de área de memória

```
data hoje,amanha;
```

```
...  
amanha = hoje;
```

```
amanha.dia = amanha.dia + 1;
```

	dia	mes	ano
hoje	10	03	2020
amanha	11	03	2020

	dia	mes	ano
hoje	10	03	2020
amanha	11	03	2020



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};  
  
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};  
  
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

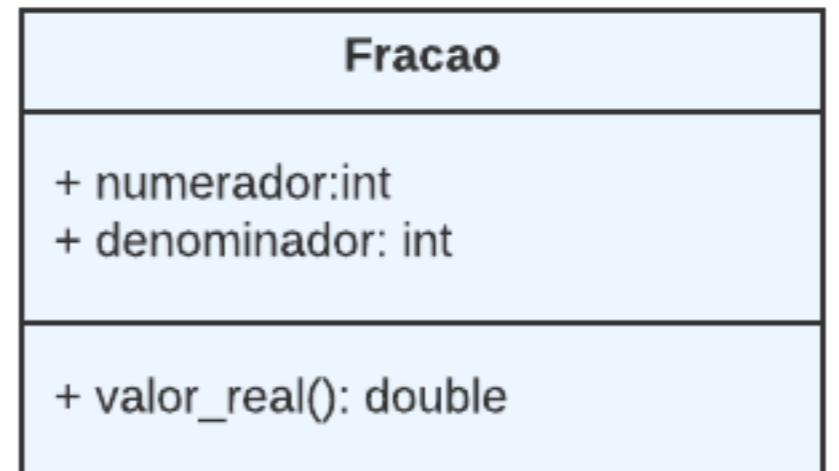
```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};  
  
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

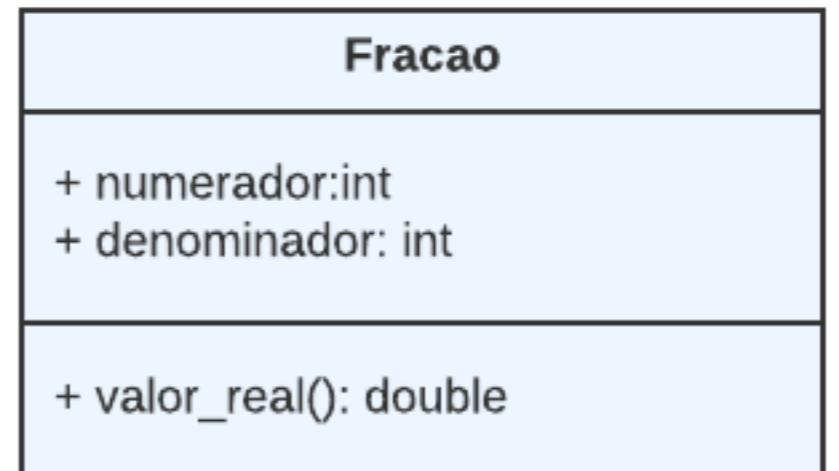
```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

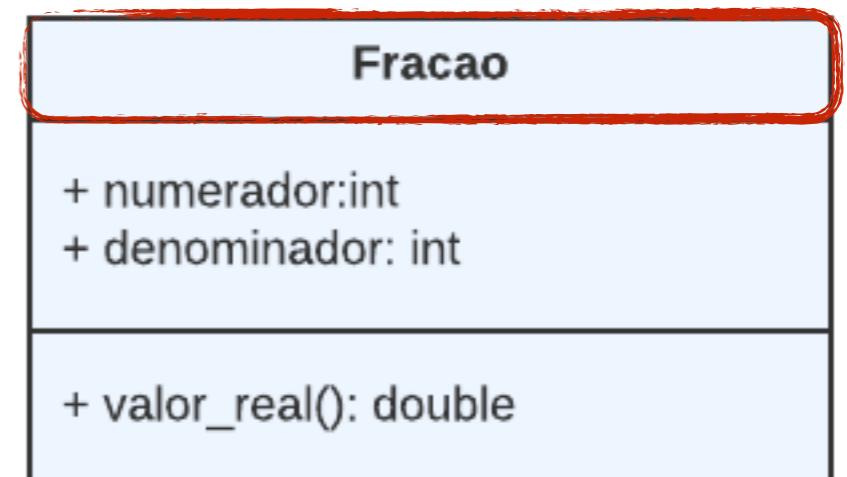
```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```

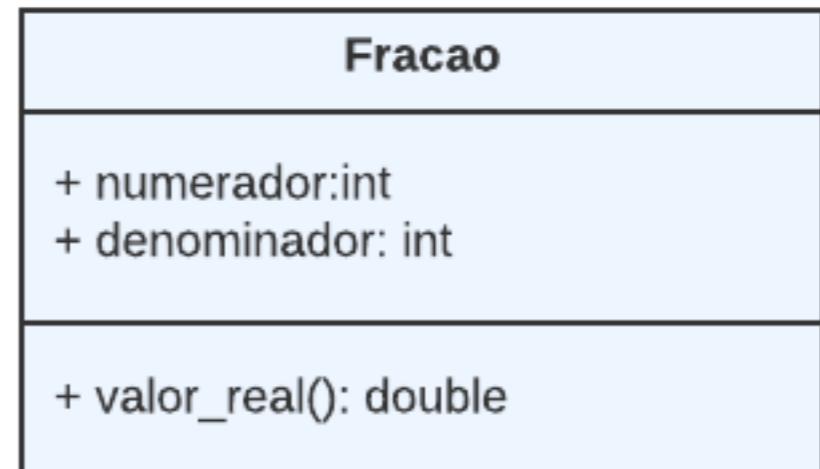




- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

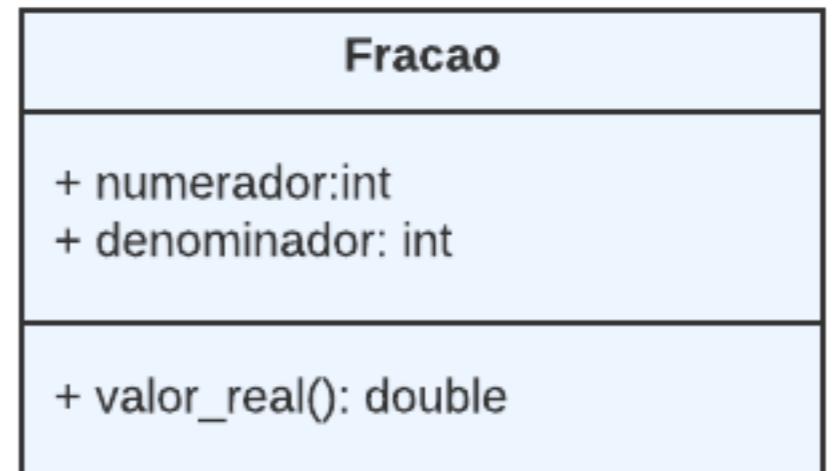
```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

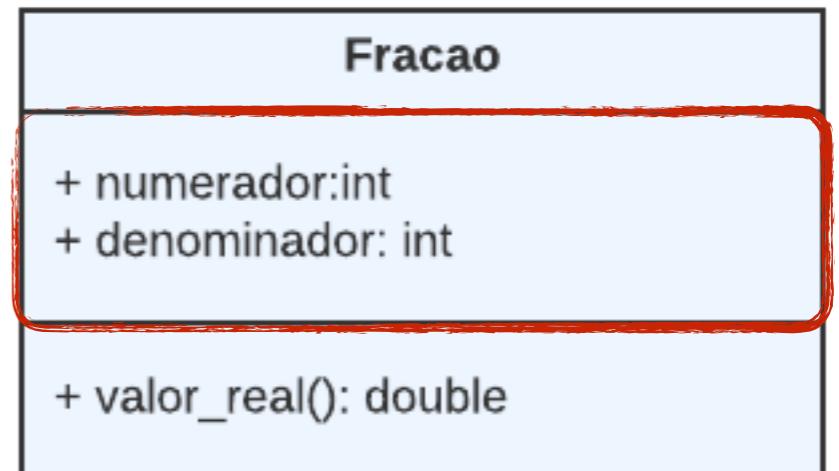
```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```

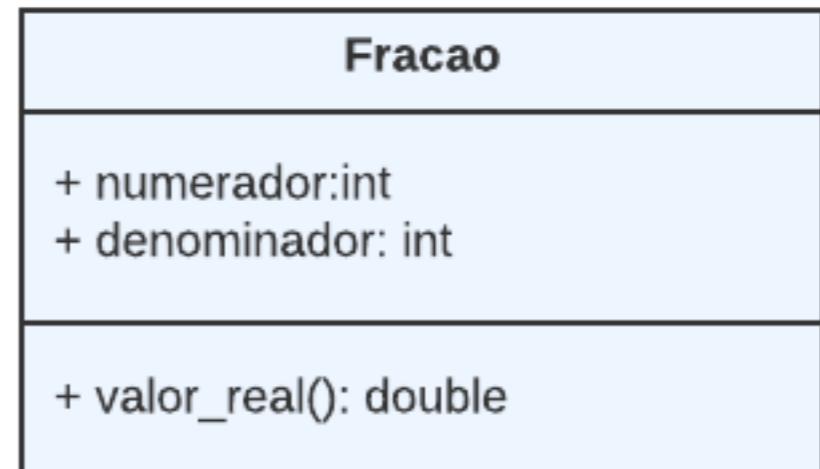




- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

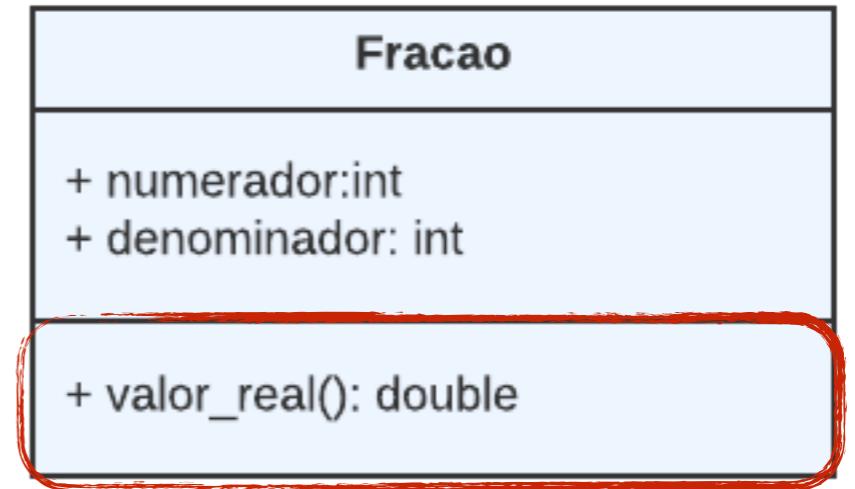
```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

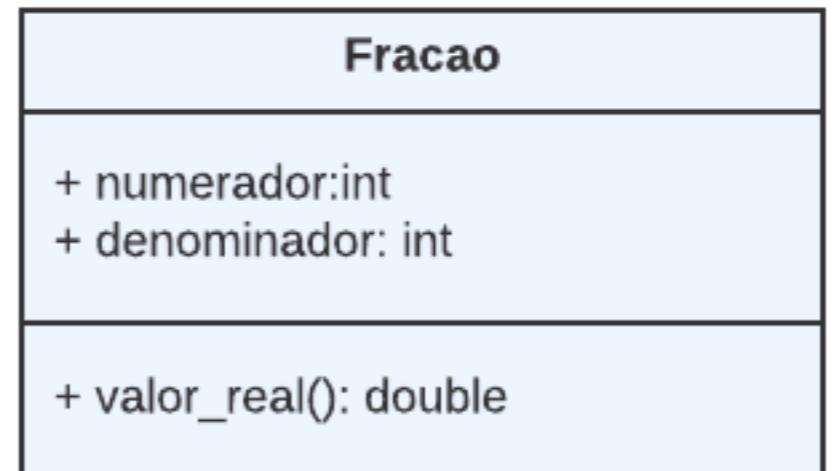
```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```



- ❖ Operações específicas de uma struct/class
 - ❖ Acessa membros/atributos da struct/class

```
struct Fracao {  
    int numerador, denominador;  
    double valor_real() {  
        double valor = static_cast<double>(numerador)/static_cast<double>(denominador);  
        return valor;  
    }  
};
```

```
int main() {  
    Fracao f1, f2;  
    std::cin >> f1.numerador >> f1.denominador;  
    std::cin >> f2.numerador >> f2.denominador;  
    std::cout << f1.valor_real() << std::endl;  
    std::cout << f2.valor_real() << std::endl;  
    return 0;  
}
```





✿ Podem ser operações complexas

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {
        double valor = static_cast<double>(numerador) / static_cast<double>(denominador);
        return valor;
    }
    void simplificar() {
        int mdc = std::gcd(numerador, denominador); // Máximo divisor comum em <numeric>
        numerador = numerador / mdc;
        denominador = denominador / mdc;
    }
    void adicionar(Fracao outra) {
        int novo_denominador = std::lcm(denominador, outra.denominador); // Mínimo múltiplo comum
        int novo_numerador = (novo_denominador / denominador) * numerador +
                            (novo_denominador / outra.denominador) * outra.numerador;
        numerador = novo_denominador;
        denominador = novo_denominador;
    }
    Fracao soma(Fracao outra){
        Fracao resultado;
        resultado.denominador = std::lcm(denominador, outra.denominador);
        resultado.numerador = (resultado.denominador / denominador) * numerador +
                            (resultado.denominador / outra.denominador) * outra.numerador;
        return resultado;
    }
};
```



✿ Podem ser operações complexas

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao outra) {...}
    Fracao soma(Fracao outra){...}
};

int main() {
    Fracao f1, f2;
    std::cin >> f1.numerador >> f1.denominador;
    std::cin >> f2.numerador >> f2.denominador;
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    f1.simplificar();
    std::cout << f1.valor_real() << std::endl;
    f2.adicionar(f1);
    Fracao f3 = f1.soma(f2);
    return 0;
}
```

Fracao
+ numerador:int + denominador: int
+ valor_real(): double + simplificar(): void + adicionar(Fracao outra): void + soma(Fracao outra): Fracao



✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```



✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```

✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

Passagem por referência!
Reduz uso de memória e
tempo de cópia

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```



✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

Passagem por referência!
Reduz uso de memória e
tempo de cópia

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```



✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

Passagem por referência!
Reduz uso de memória e
tempo de cópia

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```



✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

Passagem por referência!
Reduz uso de memória e
tempo de cópia

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```

Valor constante!
Não pode ser
alterado no método



✿ Passagem por referência

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(Fracao &outra) {...}
    Fracao soma(Fracao &outra){...}
};
```

Passagem por referência!
Reduz uso de memória e
tempo de cópia

```
#include <iostream>
#include <numeric>
struct Fracao {
    int numerador, denominador;
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};
```

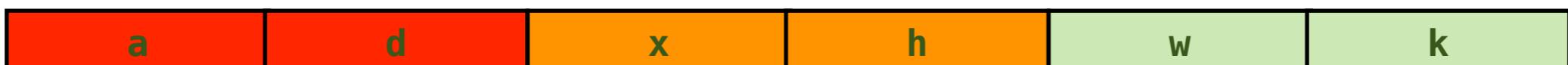
Valor constante!
Não pode ser
alterado no método



- ✿ Membros podem ser

- ✿ Públicos
- ✿ Protegidos
- ✿ Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```



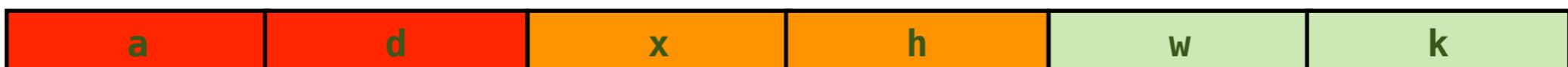


Visibilidade

- ✿ Membros podem ser

- ✿ Públicos
- ✿ Protegidos
- ✿ Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```



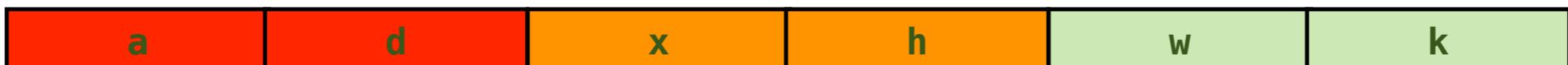


Visibilidade

- * Membros podem ser

- * Públicos
- * Protegidos
- * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```



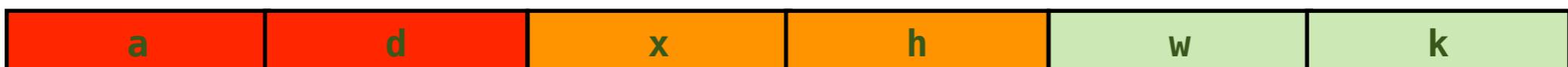


Visibilidade

- ✿ Membros podem ser

- ✿ Públicos
- ✿ Protegidos
- ✿ Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```





Visibilidade

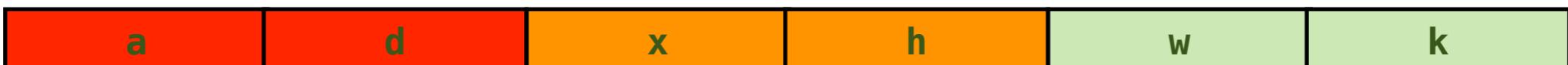
- * Membros podem ser

- * Públicos
- * Protegidos
- * Privados

```
struct exemplo{  
    private: ←  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public: ←  
        int w;  
        double k;  
};
```

Padrão de class é privado

Padrão de struct é público





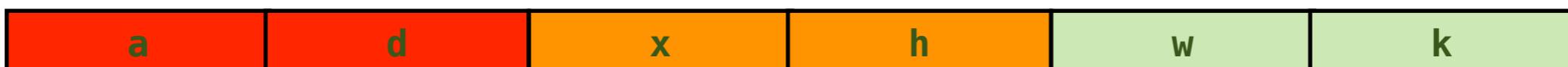
- ✿ Membros podem ser
 - ✿ Públicos
 - ✿ Protegidos
 - ✿ Privados

```
struct exemplo{  
    private: ←  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public: ←  
        int w;  
        double k;  
};
```

Encapsulamento

Padrão de class é privado

Padrão de struct é público



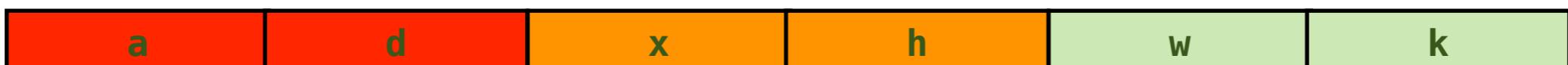


- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```

Encapsulamento

```
int main(){  
    exemplo e1;  
    e1.a = 10;  
    e1.d = 20.1;  
    e1.x = 20;  
    e1.h = 1.1;  
    e1.w = 30;  
    e1.k = 21.1;  
    return 0;  
}
```



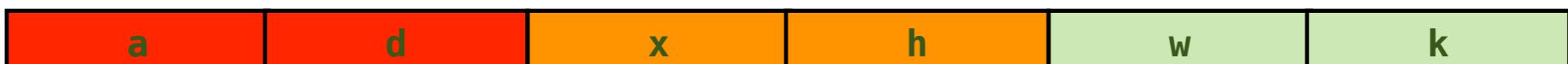


- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```

Encapsulamento

```
int main(){  
    exemplo e1;  
    e1.a = 10;  
    e1.d = 20.1;  
    e1.x = 20;  
    e1.h = 1.1;  
    e1.w = 30;  
    e1.k = 21.1;  
    return 0;  
}
```





✿ Encapsular dados

✿ Numerador e denominador privados

```
#include <iostream>
#include <numeric>
struct Fracao {
private:
    int numerador, denominador;
public:
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};

int main() {
    Fracao f1, f2;
    std::cin >> f1.numerador >> f1.denominador;
    std::cin >> f2.numerador >> f2.denominador;
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    f1.simplificar();
    std::cout << f1.valor_real() << std::endl;
    f2.adicionar(f1);
    Fracao f3 = f1.soma(f2);
    return 0;
}
```



✿ Encapsular dados

✿ Numerador e denominador privados

```
#include <iostream>
#include <numeric>
struct Fracao {
private:
    int numerador, denominador;
public:
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};

int main() {
    Fracao f1, f2;
    std::cin >> f1.numerador >> f1.denominador;
    std::cin >> f2.numerador >> f2.denominador;
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    f1.simplificar();
    std::cout << f1.valor_real() << std::endl;
    f2.adicionar(f1);
    Fracao f3 = f1.soma(f2);
    return 0;
}
```



✿ Encapsular dados

✿ Numerador e denominador privados

```
#include <iostream>
#include <numeric>
struct Fracao {
private:
    int numerador, denominador;
public:
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};

int main() {
    Fracao f1, f2;
    std::cin >> f1.numerador >> f1.denominador;
    std::cin >> f2.numerador >> f2.denominador;
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    f1.simplificar();
    std::cout << f1.valor_real() << std::endl;
    f2.adicionar(f1);
    Fracao f3 = f1.soma(f2);
    return 0;
}
```



✿ Encapsular dados

✿ Numerador e denominador privados

```
#include <iostream>
#include <numeric>
struct Fracao {
private:
    int numerador, denominador;
public:
    double valor_real() {...}
    void simplificar() {...}
    void adicionar(const Fracao &outra) {...}
    Fracao soma(const Fracao &outra){...}
};

int main() {
    Fracao f1, f2;
    std::cin >> f1.numerador >> f1.denominador;
    std::cin >> f2.numerador >> f2.denominador;
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    f1.simplificar();
    std::cout << f1.valor_real() << std::endl;
    f2.adicionar(f1);
    Fracao f3 = f1.soma(f2);
    return 0;
}
```

Acesso negado



✿ Criar métodos para definir valores

```
...
void set_numerador(int value){
    numerador = value;
}
void set_denominador(int value){
    denominador = value;
}

int main() {
    Fracao f1, f2;
    int n,d;
    std::cin >> n >> d;
    f1.set_numerador(n);
    f1.set_denominador(d);
    std::cin >> n >> d;
    f2.set_numerador(n);
    f2.set_denominador(d);
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    ...
    return 0;
}
```



✿ Criar métodos para definir valores

```
...
void set_numerador(int value){
    numerador = value;
}
void set_denominador(int value){
    denominador = value;
}

int main() {
    Fracao f1, f2;
    int n,d;
    std::cin >> n >> d;
    f1.set_numerador(n);
    f1.set_denominador(d);
    std::cin >> n >> d;
    f2.set_numerador(n);
    f2.set_denominador(d);
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    ...
    return 0;
}
```



✿ Criar métodos para definir valores

```
...
void set_numerador(int value){
    numerador = value;
}
void set_denominador(int value){
    denominador = value;
}

int main() {
    Fracao f1, f2;
    int n,d;
    std::cin >> n >> d;
    f1.set_numerador(n);
    f1.set_denominador(d);
    std::cin >> n >> d;
    f2.set_numerador(n);
    f2.set_denominador(d);
    std::cout << f1.valor_real() << std::endl;
    std::cout << f2.valor_real() << std::endl;
    ...
    return 0;
}
```



- ✿ Método especial chamado na alocação
 - ✿ Mesmo nome da struct
 - ✿ Se for definido um padrão é definidos

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    Fracao f1, f2;  
    ...  
    return 0;  
}
```



- ✿ Método especial chamado na alocação
 - ✿ Mesmo nome da struct
 - ✿ Se for definido um padrão é definidos

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main(){  
    Fracao f1, f2;  
    ...  
    return 0;  
}
```



- ✿ Método especial chamado na alocação
 - ✿ Mesmo nome da struct
 - ✿ Se for definido um padrão é definidos

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main(){  
    Fracao f1, f2;  
    ...  
    return 0;  
}
```

- ✿ Método especial chamado na alocação
 - ✿ Mesmo nome da struct
 - ✿ Se for definido um padrão é definidos

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main(){  
    Fracao f1, f2;  
    ...  
    return 0;  
}
```

ERRO!
Construtor precisa
de parâmetros



Construtor



✿ Usa construtor definido

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f1(n, d);  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```



✿ Usa construtor definido

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f1(n, d);  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```



- ⌘ Usa construtor definido

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f1(n, d);  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```

- ⌘ Pode ter mais de um



* Usa construtor definido

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f1(n, d);  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```

* Pode ter mais de um

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(){  
        // Fração padrão: 1/1  
        numerador = denominador = 1;  
    }  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    Fracao f1; // Cria 1/1  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```



* Usa construtor definido

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f1(n, d);  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```

* Pode ter mais de um

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(){  
        // Fração padrão: 1/1  
        numerador = denominador = 1;  
    }  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    Fracao f1; // Cria 1/1  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```



* Usa construtor definido

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f1(n, d);  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```

* Pode ter mais de um

```
struct Fracao {  
private:  
    int numerador, denominador;  
public:  
    Fracao(){  
        // Fração padrão: 1/1  
        numerador = denominador = 1;  
    }  
    Fracao(int n, int d){  
        numerador = n;  
        denominador = d;  
    }  
    ...  
};  
  
int main() {  
    Fracao f1; // Cria 1/1  
    int n, d;  
    std::cin >> n >>d;  
    Fracao f2(n, d);  
    ...  
    return 0;  
}
```



Arrays em struct/class

* Exemplo: uma struct/class List

```
struct aluno{  
    long long matricula;  
    int notas[10];  
    int pesos[10];  
    int qtd_notas;  
};
```

```
aluno a01;  
/*...*/  
a01.qtd_notas = 3;  
a01.notas[0] = 66;  
int x = a01.pesos[0];
```

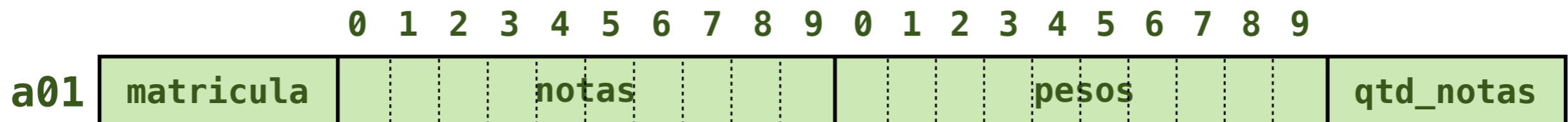


Arrays em struct/class

* Exemplo: uma struct/class List

```
struct aluno{  
    long long matricula;  
    int notas[10];  
    int pesos[10];  
    int qtd_notas;  
};
```

```
aluno a01;  
/*...*/  
a01.qtd_notas = 3;  
a01.notas[0] = 66;  
int x = a01.pesos[0];
```





✿ Mesma definição do Python list

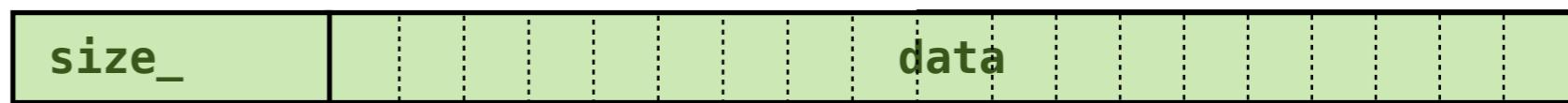
```
struct list {  
    private:  
        int data[100];  
        unsigned int size_ = 0;  
    public:  
        void append(int value) {  
            if (size_ == 100) // erro  
                return;          // Usar throw para gerar erro  
            data[size_] = value;  
            size_++;  
        }  
        int get_at(unsigned int index) {  
            if (index >= size_) // erro  
                return 0;          // Usar throw para gerar erro  
            return data[index];  
        }  
        unsigned int size() { return size_; }  
        void clear() {}  
        unsigned int count(int value) {}  
        unsigned int index(int value) {}  
        /* ... */  
};  
int main() {  
    list l1;  
    for (int i = 0; i < 10; ++i)  
        l1.append(i + 1);  
    std::cout << "l1 = [";  
    for (int i = 0; i < 10; ++i)  
        std::cout << l1.get_at(i) << " ";  
    std::cout << "]" << std::endl;  
    return 0;  
}
```

✿ Mesma definição do Python list

```

struct list {
    private:
        int data[100];
        unsigned int size_ = 0;
    public:
        void append(int value) {
            if (size_ == 100) // erro
                return;          // Usar throw para gerar erro
            data[size_] = value;
            size_++;
        }
        int get_at(unsigned int index) {
            if (index >= size_) // erro
                return 0;          // Usar throw para gerar erro
            return data[index];
        }
        unsigned int size() { return size_; }
        void clear() {}
        unsigned int count(int value) {}
        unsigned int index(int value) {}
        /* ... */
    };
int main() {
    list l1;
    for (int i = 0; i < 10; ++i)
        l1.append(i + 1);
    std::cout << "l1 = [";
    for (int i = 0; i < 10; ++i)
        std::cout << l1.get_at(i) << " ";
    std::cout << "]" << std::endl;
    return 0;
}

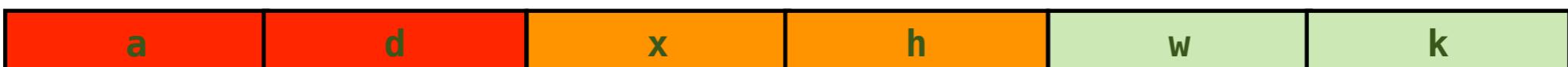
```





- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

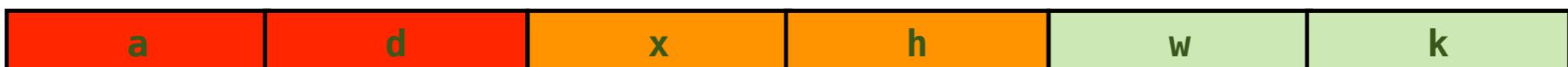
```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```





- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

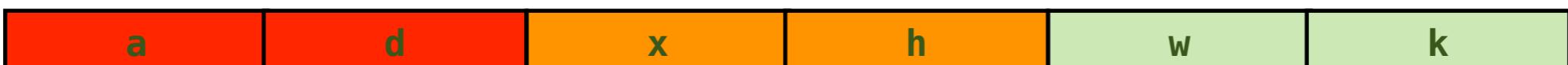
```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```





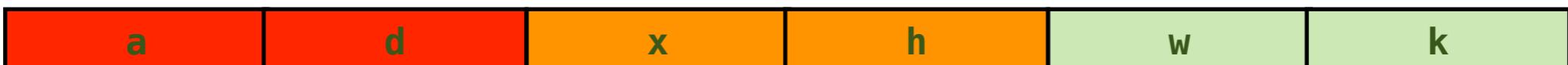
- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```



- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```

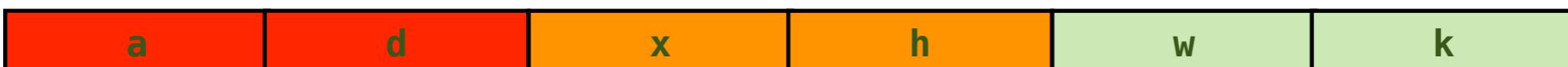


- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```

Padrão de class é privado

Padrão de struct é público



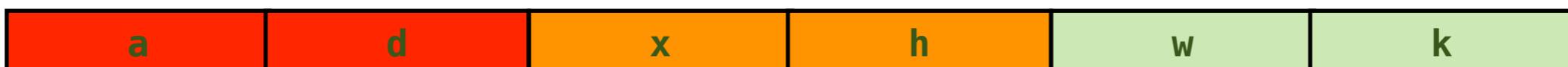
- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
struct exemplo{  
    private:  
        int a;  
        double d;  
    protected:  
        int x;  
        float h;  
    public:  
        int w;  
        double k;  
};
```

Encapsulamento

Padrão de class é privado

Padrão de struct é público



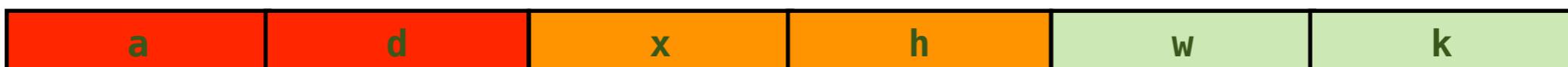
- * Em C++ struct é equivalente a classe
- * Membros podem ser
 - * Públicos
 - * Protegidos
 - * Privados

```
class exemplo {  
private: ←  
    int a;  
    double d;  
protected:  
    int x;  
    float h;  
public: ←  
    int w;  
    double k;  
};
```

Encapsulamento

Padrão de class é privado

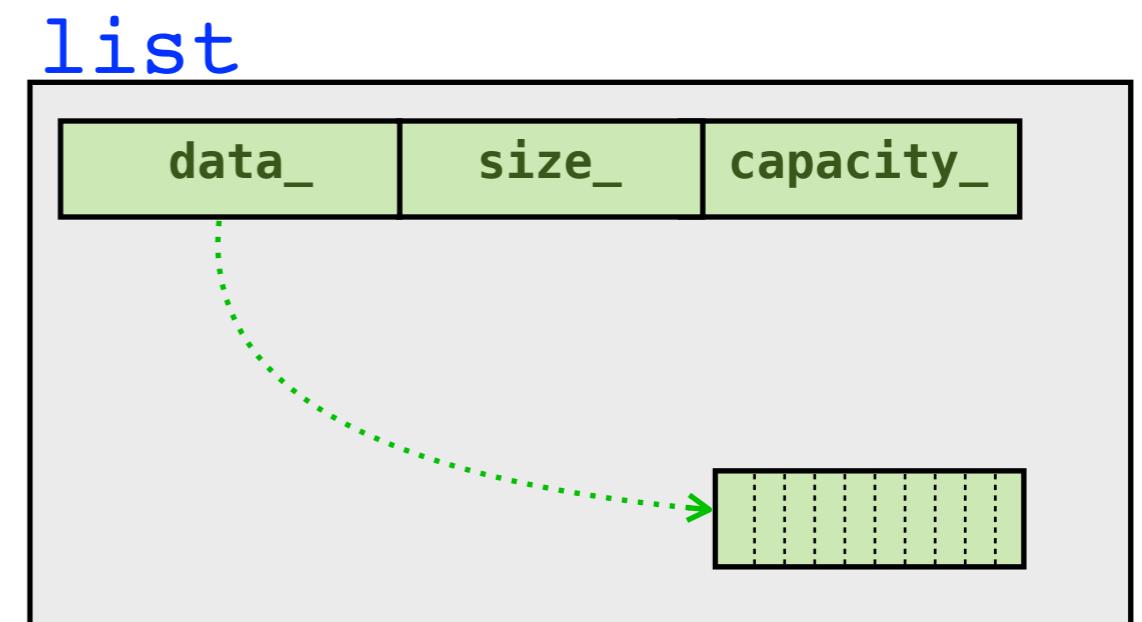
Padrão de struct é público





- ❖ Ideal é que espaço reservado seja pequeno
 - ❖ Aumenta de acordo com a necessidade
 - ❖ Memória alocada dinamicamente
 - ❖ Ponteiros

- ❖ Ideal é que espaço reservado seja pequeno
 - ❖ Aumenta de acordo com a necessidade
 - ❖ Memória alocada dinamicamente
 - ❖ Ponteiros





```
#include <iostream>

class list {
private:
    int *data_;
    unsigned int size_;
    unsigned int capacity_;
    void increase_capacity() { /* TODO */}

public:
    list() {
        data_ = new int[100];
        capacity_ = 100;
        size_ = 0;
    }
    void append(int value) {
        if (size_ == capacity_) {
            increase_capacity();
        }
        data_[size_] = value;
        size_++;
    }
    /* ... */
};
```

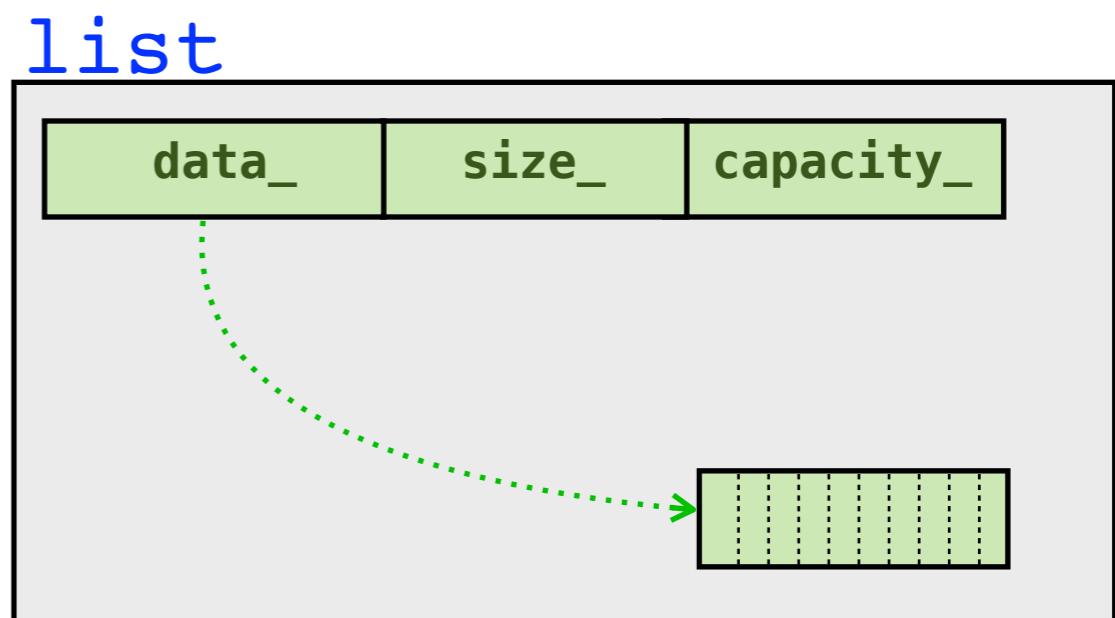
```
int main() {
    list l1;
    for (int i = 0; i < 300; ++i)
        l1.append(i + 1);
    std::cout << "l1 = [";
    for (int i = 0; i < 10; ++i)
        std::cout << l1.get_at(i) << " ";
    std::cout << "]" << std::endl;
    return 0;
}
```

```
#include <iostream>

class list {
private:
    int *data_;
    unsigned int size_;
    unsigned int capacity_;
    void increase_capacity() { /* TODO */}

public:
    list() {
        data_ = new int[100];
        capacity_ = 100;
        size_ = 0;
    }
    void append(int value) {
        if (size_ == capacity_) {
            increase_capacity();
        }
        data_[size_] = value;
        size_++;
    }
    /* ... */
};
```

```
int main() {
    list l1;
    for (int i = 0; i < 300; ++i)
        l1.append(i + 1);
    std::cout << "l1 = [";
    for (int i = 0; i < 10; ++i)
        std::cout << l1.get_at(i) << " ";
    std::cout << "]" << std::endl;
    return 0;
}
```

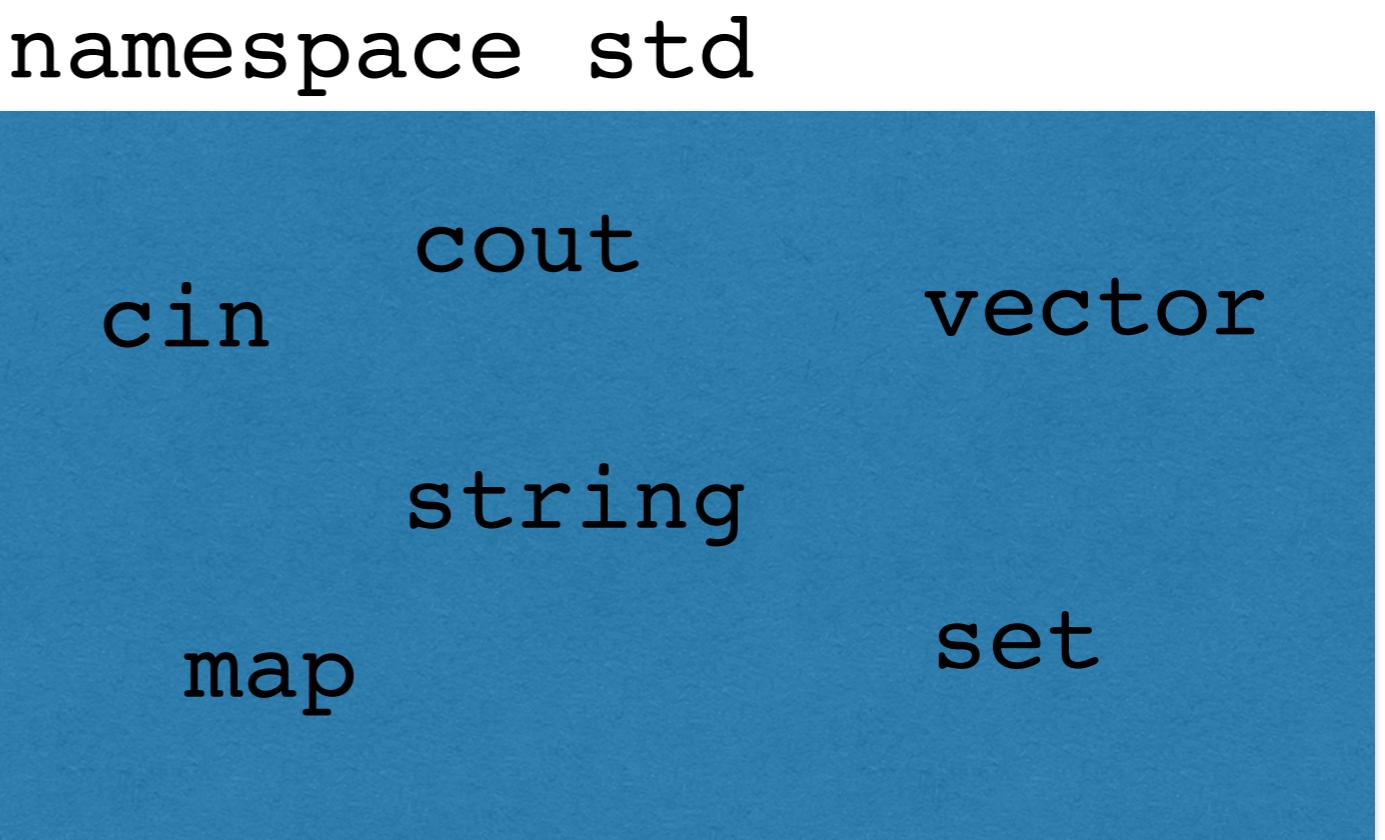




- ❖ Agrupar elementos relacionados em um mesmo espaço
 - ❖ Nome identifica espaço
 - ❖ :: para acessar elementos
 - ❖ std::cin, std::cout, std::end, std::string, std::vector, etc
- ❖ Para denir namespace



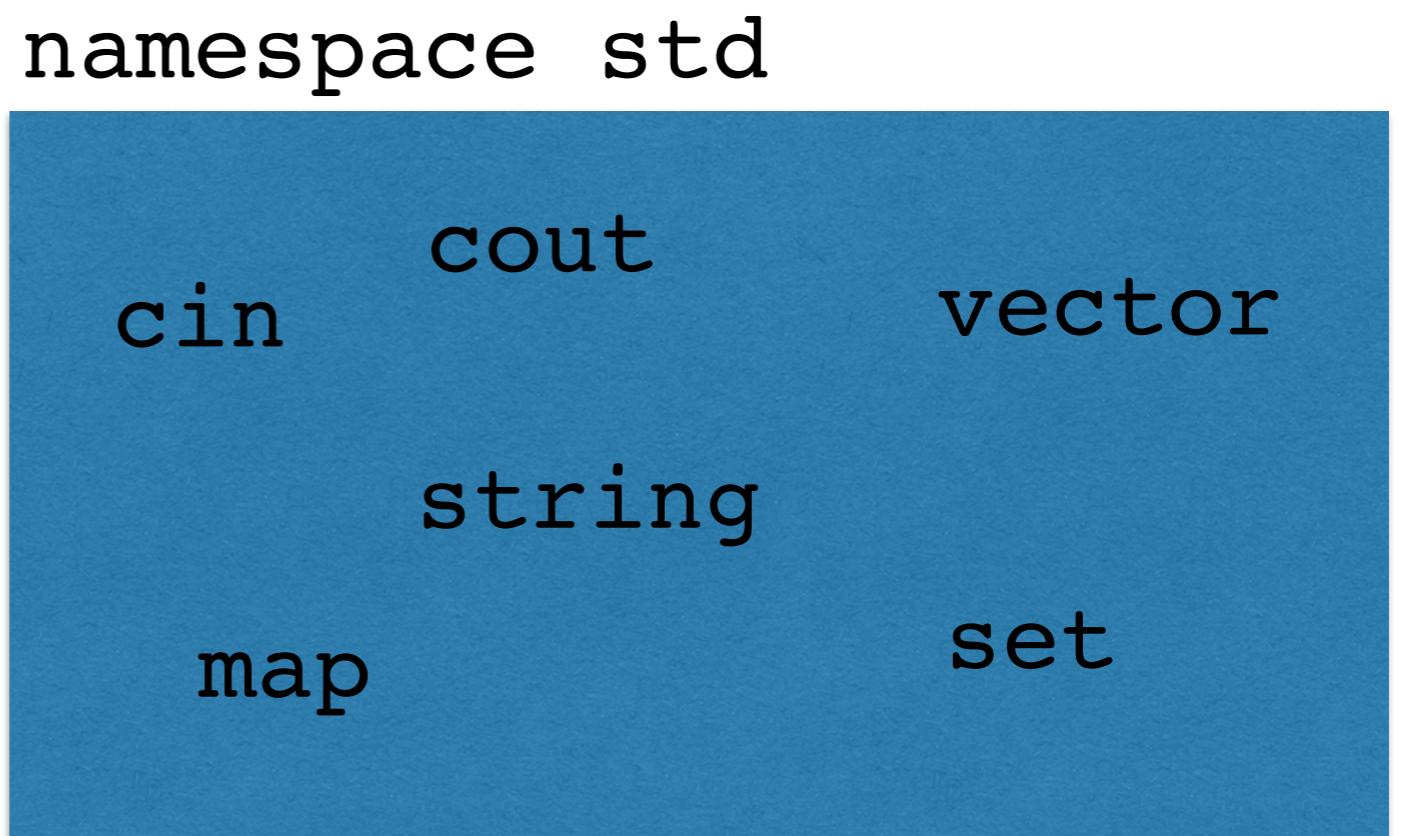
- ✿ Agrupar elementos relacionados em um mesmo espaço
 - ✿ Nome identifica espaço
 - ✿ :: para acessar elementos
 - ✿ std::cin, std::cout, std::end, std::string, std::vector, etc
- ✿ Para definir namespace





- ✿ Agrupar elementos relacionados em um mesmo espaço
 - ✿ Nome identifica espaço
 - ✿ :: para acessar elementos
 - ✿ std::cin, std::cout, std::end, std::string, std::vector, etc
- ✿ Para definir namespace

```
namespace NOME {  
    /* Definição dos  
       elementos  
       pertencentes ao  
       namespace NOME  
    */  
}
```





✿ Definir nossas classes no namespace ifrn

```
namespace ifrn {
    class list {
        private:
            int *data_;
            /* ... */
        public:
            list() {
                data_ = new int[100];
                capacity_ = 100;
                size_ = 0;
                /* ... */
            };
            class Fracao {
                /* ... */
            };
    } // namespace ifrn
```



* Definir nossas classes no namespace ifrn

```
namespace ifrn {
    class list {
        private:
            int *data_;
            /* ... */
        public:
            list() {
                data_ = new int[100];
                capacity_ = 100;
                size_ = 0;
                /* ... */
            };
            class Fracao {
                /* ... */
            };
    } // namespace ifrn
```



✿ Definir nossas classes no namespace ifrn

```
namespace ifrn {
    class list {
        private:
            int *data_;
            /* ... */
        public:
            list() {
                data_ = new int[100];
                capacity_ = 100;
                size_ = 0;
                /* ... */
            };
            class Fracao {
                /* ... */
            };
    } // namespace ifrn
```



* Definir nossas classes no namespace ifrn

```
namespace ifrn {  
    class list {  
        private:  
            int *data_;  
            /* ... */  
        public:  
            list() {  
                data_ = new int[100];  
                capacity_ = 100;  
                size_ = 0;  
                /* ... */  
            };  
  
            class Fracao {  
                /* ... */  
            };  
    };  
}  
} // namespace ifrn
```

```
int main() {  
    ifrn::list l1;  
    for (int i = 0; i < 300; ++i)  
        l1.append(i + 1);  
    std::cout << "l1 = [";  
    ifrn::Fracao f1, f2;  
    for (int i = 0; i < 10; ++i)  
        std::cout << l1.get_at(i) << " ";  
    std::cout << "]" << std::endl;  
    return 0;  
}
```



* Definir nossas classes no namespace ifrn

```
namespace ifrn {  
    class list {  
        private:  
            int *data_;  
            /* ... */  
        public:  
            list() {  
                data_ = new int[100];  
                capacity_ = 100;  
                size_ = 0;  
                /* ... */  
            };  
  
            class Fracao {  
                /* ... */  
            };  
    };  
}  
} // namespace ifrn
```

```
int main() {  
    ifrn::list l1;  
    for (int i = 0; i < 300; ++i)  
        l1.append(i + 1);  
    std::cout << "l1 = [";  
    ifrn::Fracao f1, f2;  
    for (int i = 0; i < 10; ++i)  
        std::cout << l1.get_at(i) << " ";  
    std::cout << "]" << std::endl;  
    return 0;  
}
```



* Definir nossas classes no namespace ifrn

```
namespace ifrn {  
    class list {  
        private:  
            int *data_;  
            /* ... */  
        public:  
            list() {  
                data_ = new int[100];  
                capacity_ = 100;  
                size_ = 0;  
                /* ... */  
            };  
  
            class Fracao {  
                /* ... */  
            };  
    };  
}  
} // namespace ifrn
```

```
int main() {  
    ifrn::list l1;  
    for (int i = 0; i < 300; ++i)  
        l1.append(i + 1);  
    std::cout << "l1 = [";  
    ifrn::Fracao f1, f2;  
    for (int i = 0; i < 10; ++i)  
        std::cout << l1.get_at(i) << " ";  
    std::cout << "]" << std::endl;  
    return 0;  
}
```



✿ Dúvidas?



Programação

C++

Aula 06
Jorgiano Vidal