# Linguagem de Programação II

**Prof.Antonio Carlos Sobieranski**

DEC7532 | ENC | DEC | CTS

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

# Orientação à Objetos

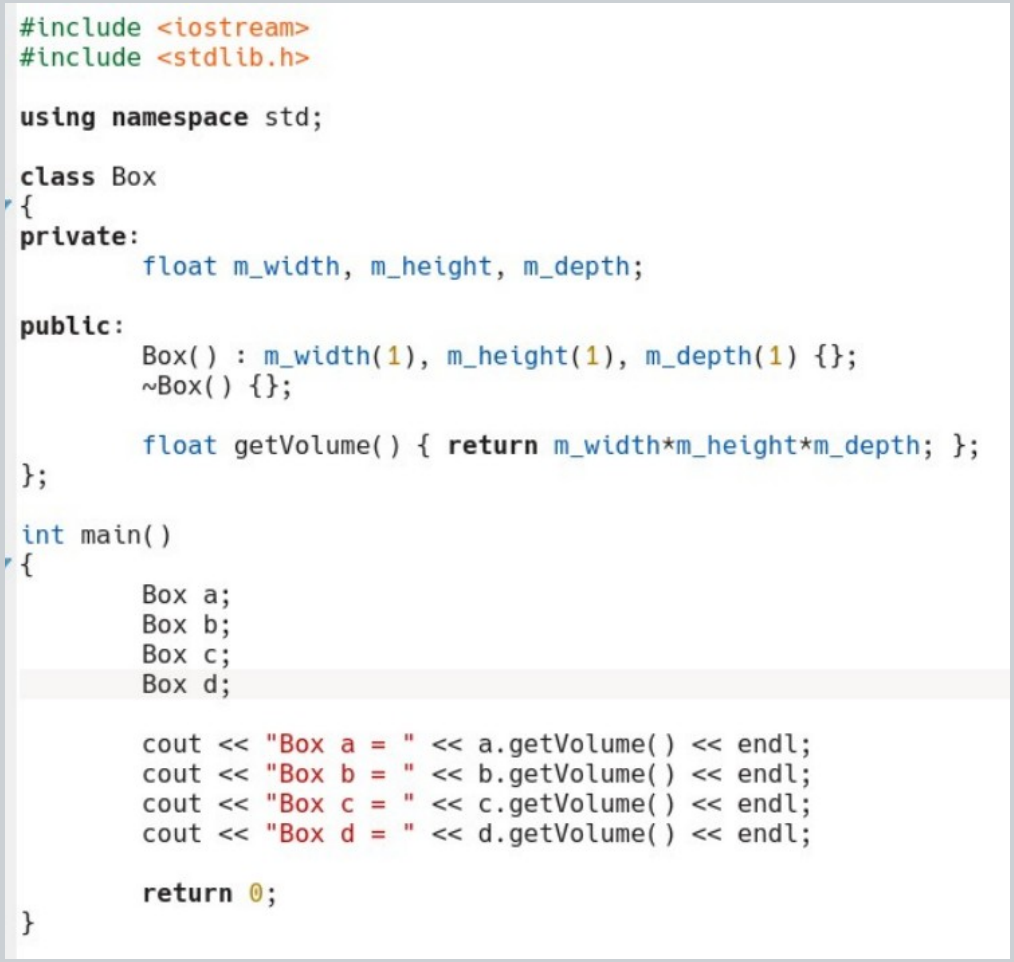
## Construtores e Destrutores

### Construtores

- Funções membro especiais chamadas pelo sistema no momento de CRIACAO de um objeto
- Propriedades:
  - Não possuem valor de retorno
  - Permitem fazer sobre-carga (vários construtores != argumentos)
  - Inicializacao do objeto de forma organizada
    - Imagina esquecer de inicializar o construtor ou chamar 2 vezes !
  - Construtor tem sempre o MESMO NOME da classe
  - Construtor não declarado, existe virtualmente

# Orientação à Objetos

## Construtores e Destrutores

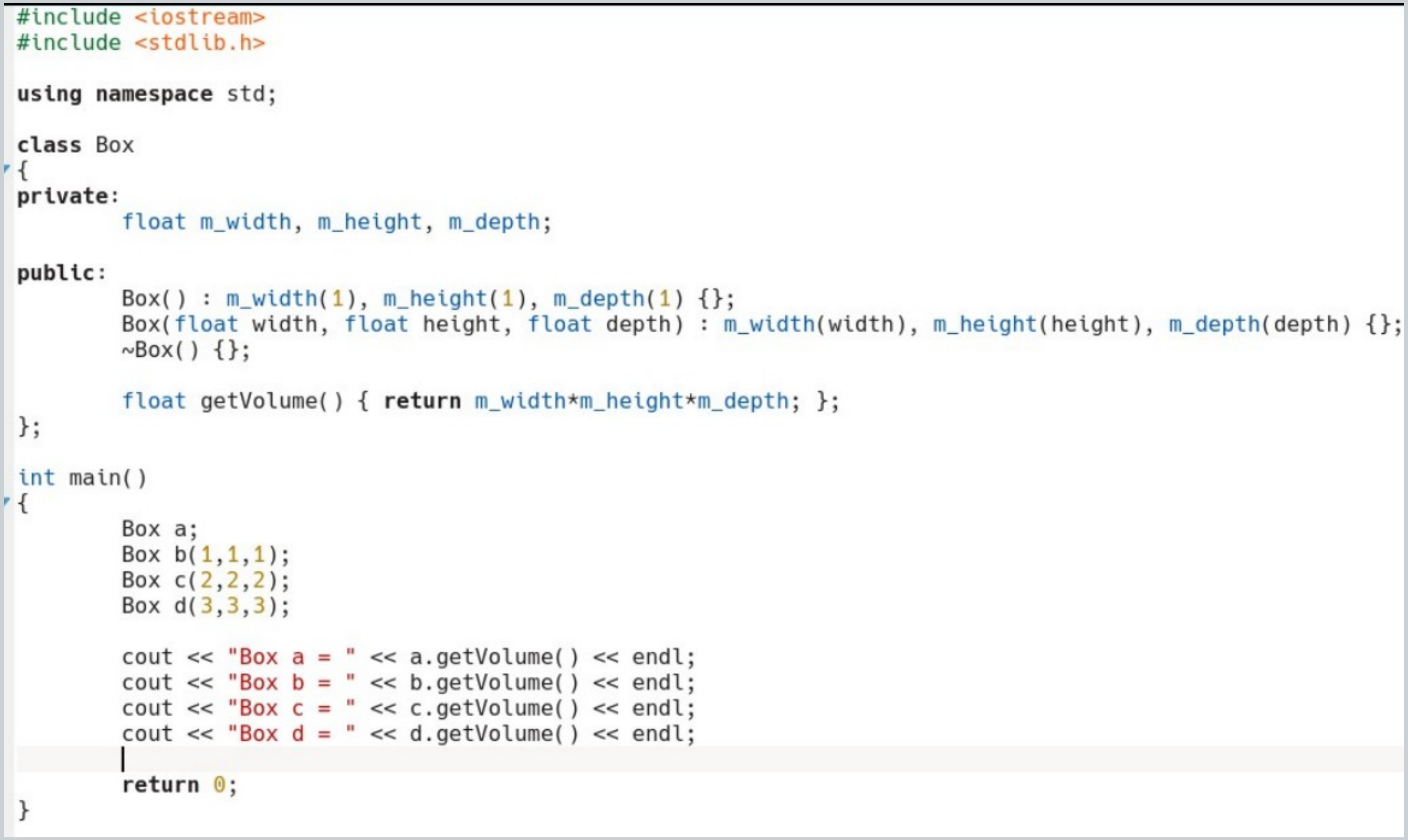


```cpp
#include <iostream>
#include <stdlib.h>

using namespace std;

class Box
{
private:
	float m_width, m_height, m_depth;

public:
	Box() : m_width(1), m_height(1), m_depth(1) {};
	~Box() {};

	float getVolume() { return m_width*m_height*m_depth; };
};

int main()
{
	Box a;
	Box b;
	Box c;
	Box d;

	cout << "Box a = " << a.getVolume() << endl;
	cout << "Box b = " << b.getVolume() << endl;
	cout << "Box c = " << c.getVolume() << endl;
	cout << "Box d = " << d.getVolume() << endl;

	return 0;
}
```

UFSC UNIVERSIDADE FEDERAL DE SANTA CATARINA

# Orientação à Objetos

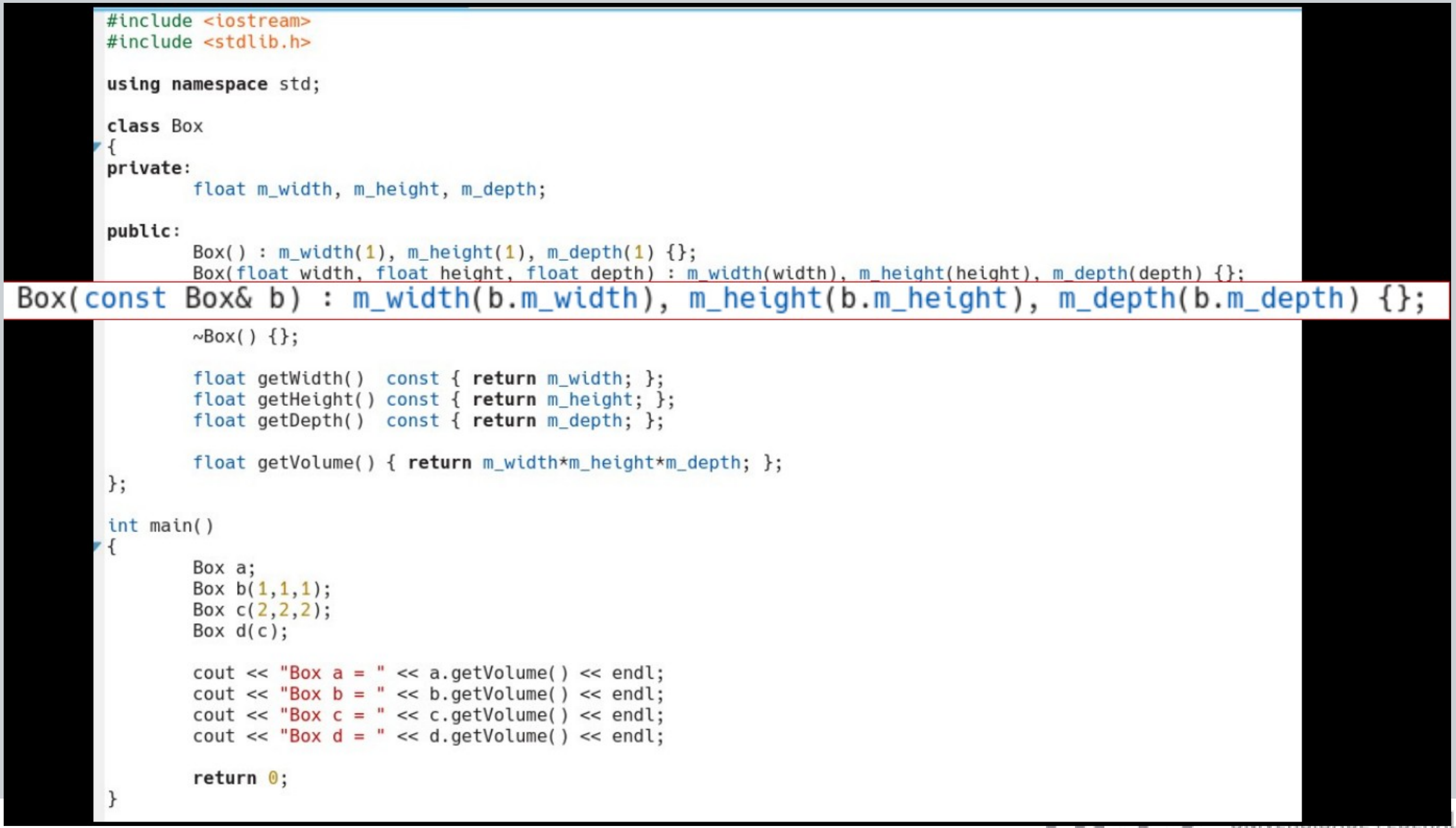
## Construtores e Destrutores



```cpp
#include <iostream>
#include <stdlib.h>

using namespace std;

class Box
{
private:
        float m_width, m_height, m_depth;

public:
        Box() : m_width(1), m_height(1), m_depth(1) {};
        Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
        ~Box() {};

        float getVolume() { return m_width*m_height*m_depth; };
};

int main()
{
        Box a;
        Box b(1,1,1);
        Box c(2,2,2);
        Box d(3,3,3);

        cout << "Box a = " << a.getVolume() << endl;
        cout << "Box b = " << b.getVolume() << endl;
        cout << "Box c = " << c.getVolume() << endl;
        cout << "Box d = " << d.getVolume() << endl;

        return 0;
}
```

# Orientação à Objetos

## Construtores e Destrutores

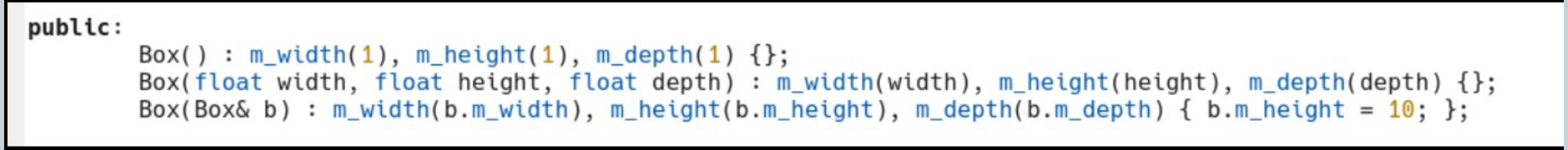


```
#include <iostream>
#include <stdlib.h>

using namespace std;

class Box
{
private:
        float m_width, m_height, m_depth;

public:
        Box() : m_width(1), m_height(1), m_depth(1) {};
        Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) {};
        ~Box() {};

        float getWidth()  const { return m_width; };
        float getHeight() const { return m_height; };
        float getDepth()  const { return m_depth; };

        float getVolume() { return m_width*m_height*m_depth; };
};

int main()
{
        Box a;
        Box b(1,1,1);
        Box c(2,2,2);
        Box d(c);

        cout << "Box a = " << a.getVolume() << endl;
        cout << "Box b = " << b.getVolume() << endl;
        cout << "Box c = " << c.getVolume() << endl;
        cout << "Box d = " << d.getVolume() << endl;

        return 0;
}
```

# Orientação à Objetos

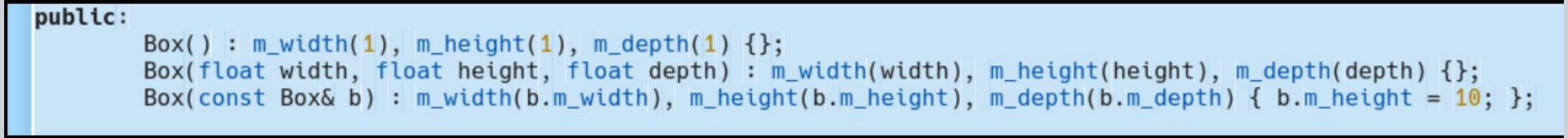
## Construtores e Destrutores

- Sem *const*

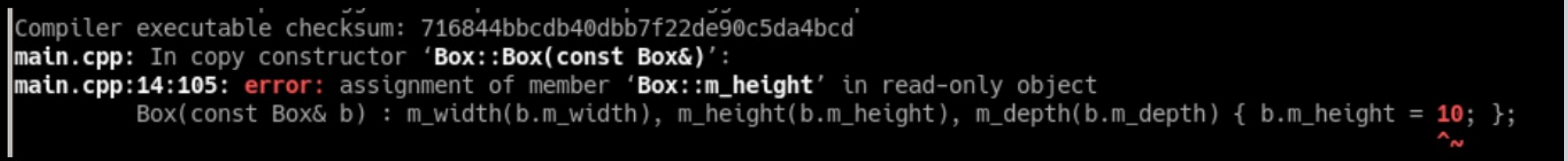


```
public:
        Box() : m_width(1), m_height(1), m_depth(1) {};
        Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
        Box(Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) { b.m_height = 10; };
```

- Com *const*



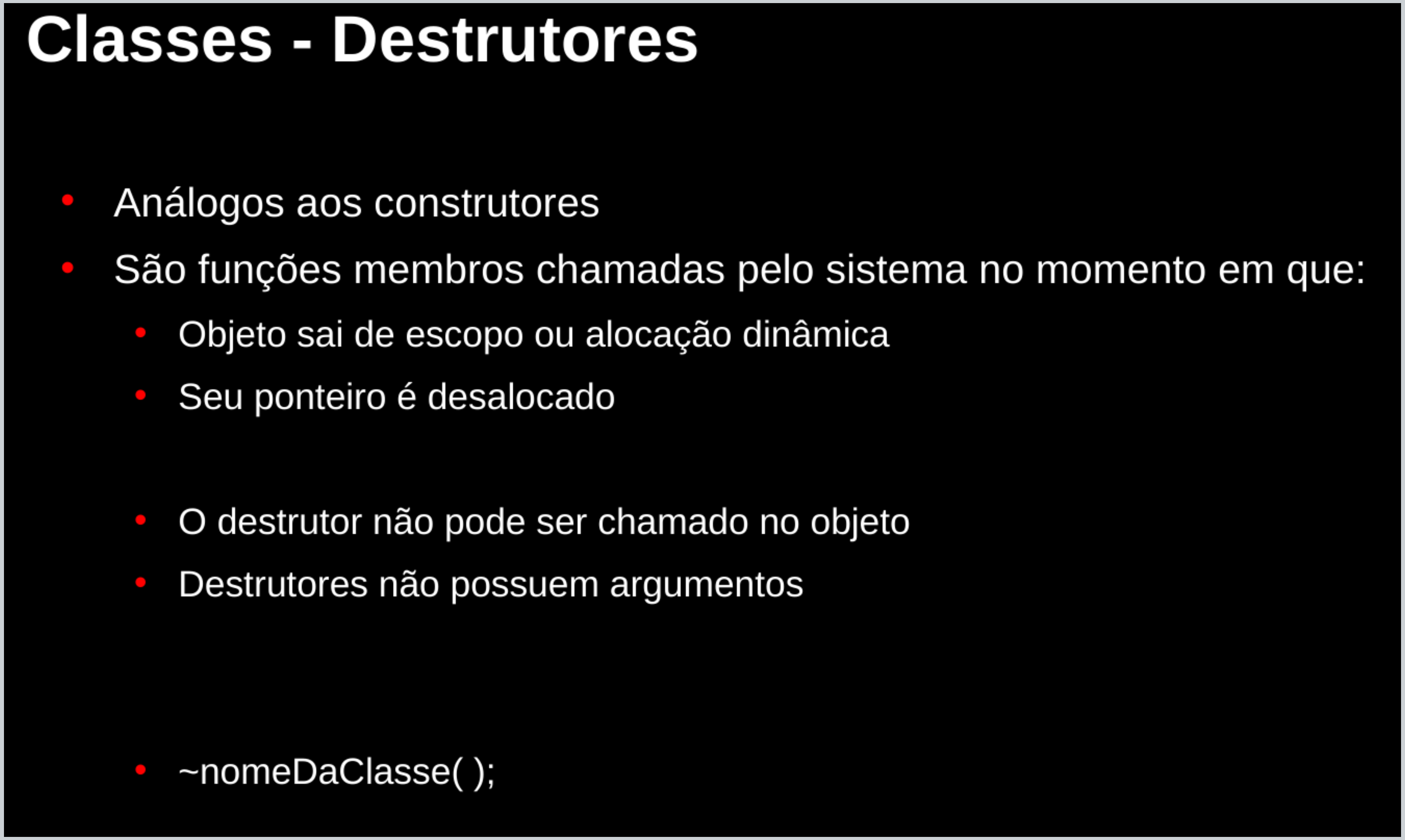
```
public:
        Box() : m_width(1), m_height(1), m_depth(1) {};
        Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
        Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) { b.m_height = 10; };
```



```
Compiler executable checksum: 716844bbcdb40dbb7f22de90c5da4bcd
main.cpp: In copy constructor 'Box::Box(const Box&)':
main.cpp:14:105: error: assignment of member 'Box::m_height' in read-only object
         Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) { b.m_height = 10; };
                                                                                                         ^~
```

# Orientação à Objetos

## Construtores e Destrutores



### Classes - Destrutores

- Análogos aos construtores
- São funções membros chamadas pelo sistema no momento em que:
  - Objeto sai de escopo ou alocação dinâmica
  - Seu ponteiro é desalocado

  - O destrutor não pode ser chamado no objeto
  - Destrutores não possuem argumentos

  - ~nomeDaClasse( );

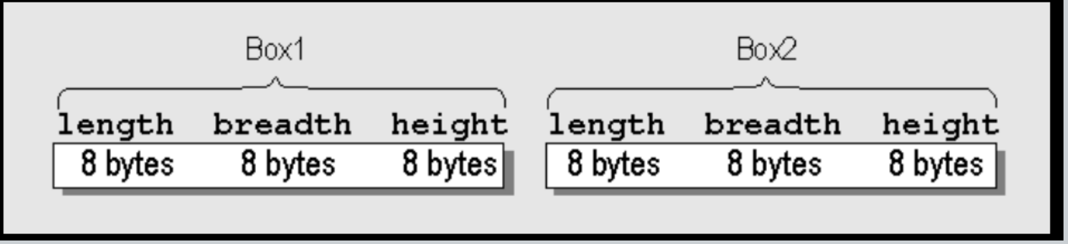
# Orientação à Objetos

## Construtores e Destrutores

### Uso de Operadores

```
class Box {
public:
        double m_width;
        double m_height;
        double m_depth;
};

Box myBox;
```

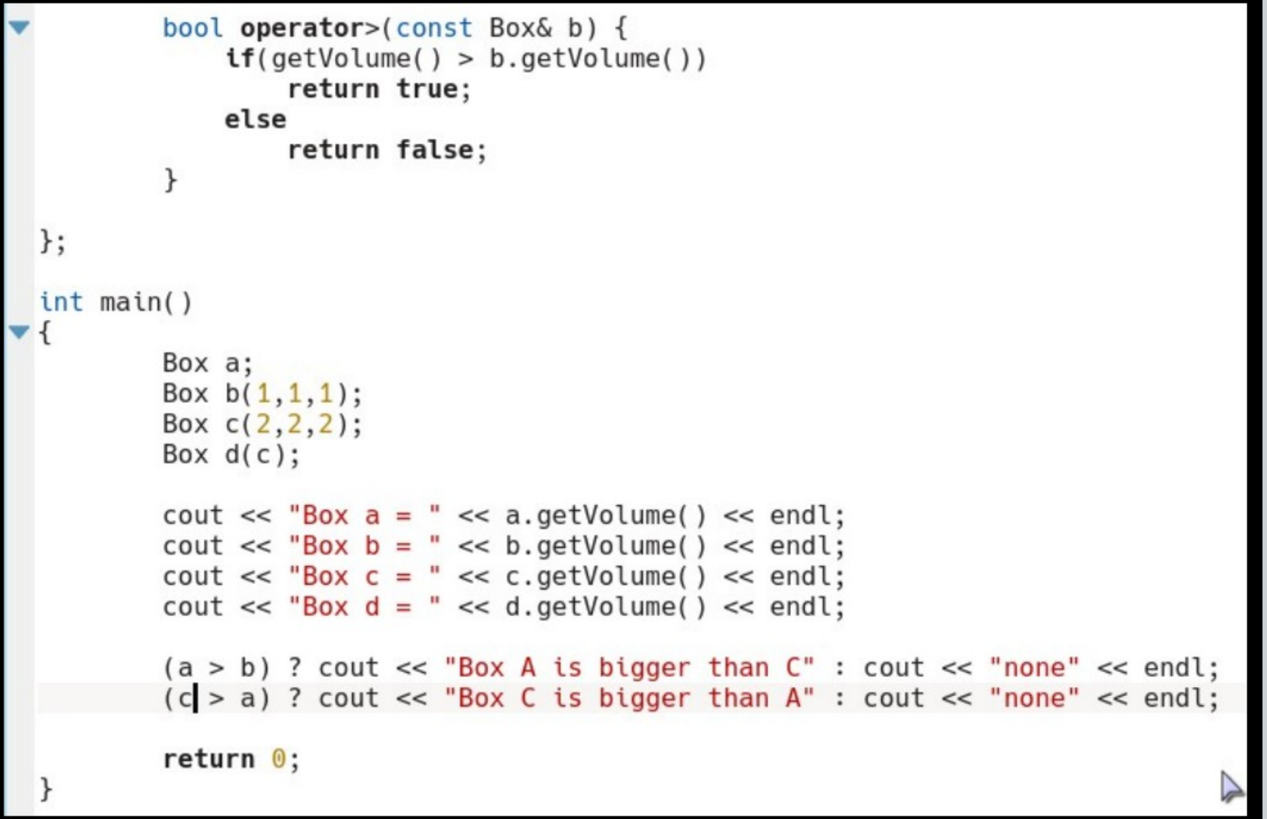
```
Box Box1(1,1,1);
Box Box2(2,2,2);

if(Box1 > Box2)
    Box1.Fill();
else
    Box2.Fill();
```



# Orientação à Objetos

## Construtores e Destrutores

### Uso de Operadores

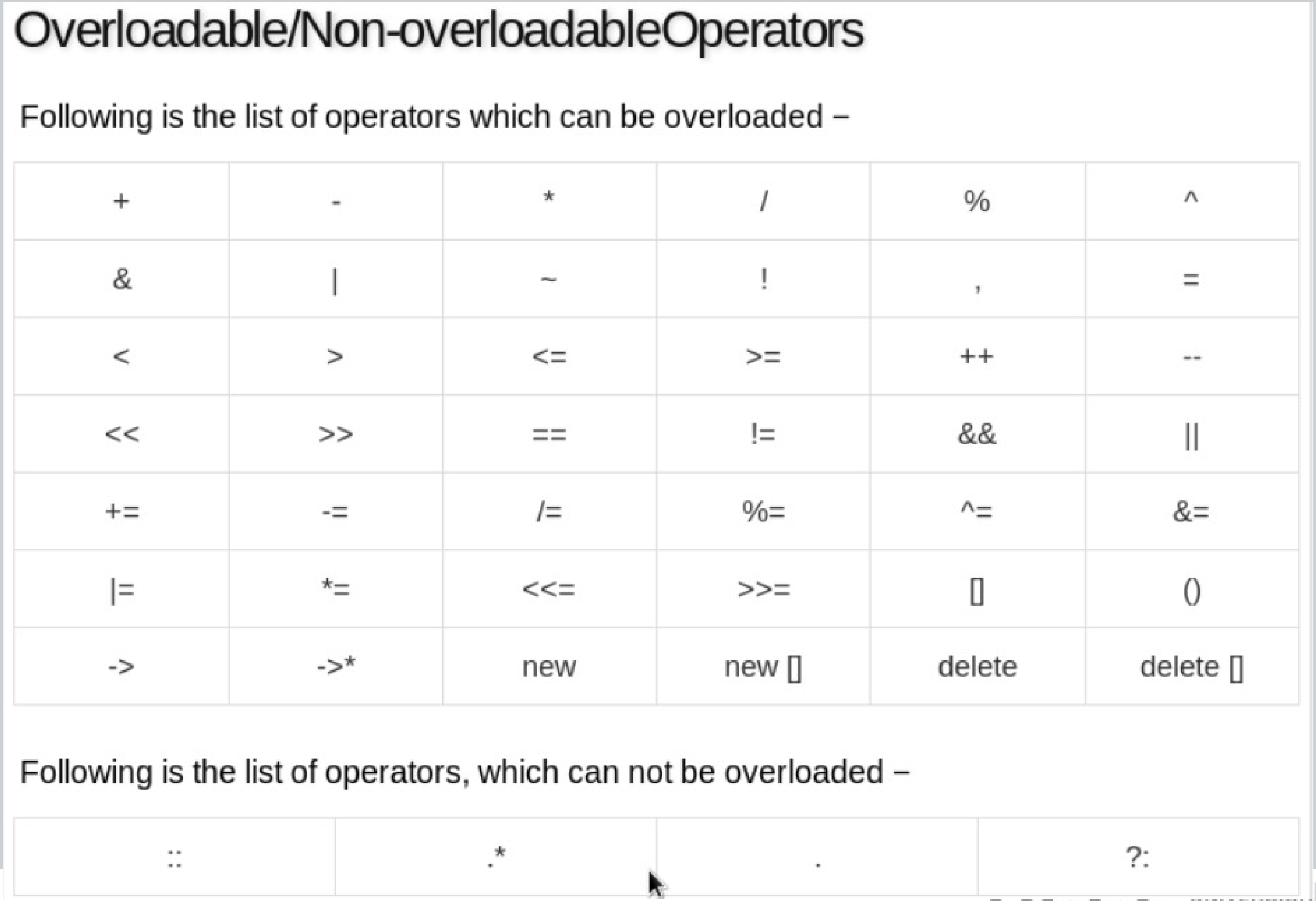


```
        bool operator>(const Box& b) {
            if(getVolume() > b.getVolume())
                return true;
            else
                return false;
        }

};

int main()
{
        Box a;
        Box b(1,1,1);
        Box c(2,2,2);
        Box d(c);

        cout << "Box a = " << a.getVolume() << endl;
        cout << "Box b = " << b.getVolume() << endl;
        cout << "Box c = " << c.getVolume() << endl;
        cout << "Box d = " << d.getVolume() << endl;

        (a > b) ? cout << "Box A is bigger than C" : cout << "none" << endl;
        (c > a) ? cout << "Box C is bigger than A" : cout << "none" << endl;

        return 0;
}
```

UFSC UNIVERSIDADE FEDERAL DE SANTA CATARINA

# Orientação à Objetos

## Construtores e Destrutores



### Overloadable/Non-overloadableOperators

Following is the list of operators which can be overloaded –

| | | | | | |
|---|---|---|---|---|---|
| + | - | * | / | % | ^ |
| & | \| | ~ | ! | , | = |
| < | > | <= | >= | ++ | -- |
| << | >> | == | != | && | \|\| |
| += | -= | /= | %= | ^= | &= |
| \|= | *= | <<= | >>= | [] | () |
| -> | ->* | new | new [] | delete | delete [] |

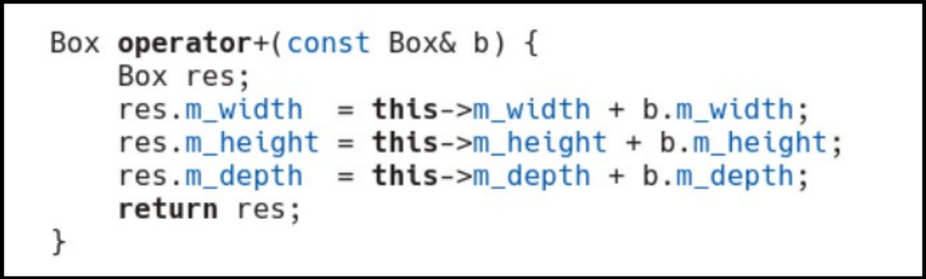
Following is the list of operators, which can not be overloaded –

| | | | |
|---|---|---|---|
| :: | .* | . | ?: |

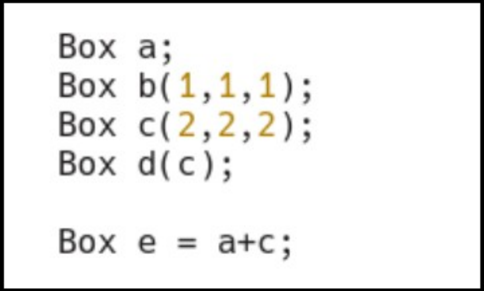
# Orientação à Objetos

## Construtores e Destrutores

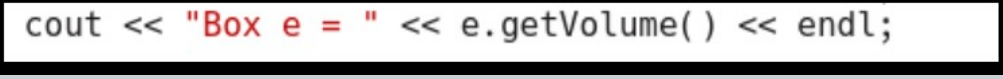
### Uso de Operadores – somando 2 caixas



```
Box operator+(const Box& b) {
    Box res;
    res.m_width  = this->m_width + b.m_width;
    res.m_height = this->m_height + b.m_height;
    res.m_depth  = this->m_depth + b.m_depth;
    return res;
}
```



```
Box a;
Box b(1,1,1);
Box c(2,2,2);
Box d(c);

Box e = a+c;
```



```
cout << "Box e = " << e.getVolume() << endl;
```

## Contato

Prof.Antonio Carlos Sobieranski – DEC | A316

E-mail: a.sobieranski@ufsc.br

Inst: @antonio.sobieranski

UNIVERSIDADE FEDERAL
DE SANTA CATARINA