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

```
#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;
}
```

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() {};

        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

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

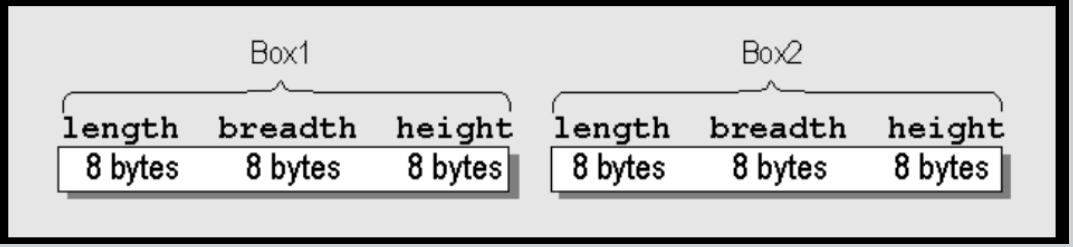
Uso de Operadores

```
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;
}
```

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;
```

Orientação à Objetos

Construtores e Destrutores

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

//operators methods
bool isBigger(const Box& b)   { return this->getVolume() > b.getVolume(); };
bool operator> (const Box& b) { return this->getVolume() > b.getVolume(); };

Box operator+ (const Box& b)
{
        Box result;
        result.m_width =m_width +b.m_width;
        result.m_height=m_height+b.m_height;
        result.m_depth =m_depth +b.m_depth;
        return result;
}
Box operator+ (float scalar)
{
        Box result;
        result.m_width =m_width +scalar;
        result.m_height=m_height+scalar;
        result.m_depth =m_depth +scalar;
        return result;
}
```

Orientação à Objetos

Construtores e Destrutores

```
bool operator== (const Box& b)
{
    return (m_width==b.m_width && m_height==b.m_height && m_depth==b.m_depth);
}
void operator++ (int)
{
    m_width++;
    m_height++;
    m_depth++;
}
void operator++ ()
{
    m_width++;
    m_height++;
    m_depth++;
}
```

Orientação à Objetos

Construtores e Destrutores – Exercício – “Triangles !”

1. Criar uma classe **Triangle**
2. Classe **Triangle** deve ter como atributos:
   – *3 lados, 3 angulos, Area*, todos em ponto flutuante.

3. Classe **Triangle** deve ter diferentes construtores, sendo que quando utilizados a **area de um triangulo qualquer** seja calculada (não necessariamente equilátero). Deve possuir também métodos de acesso e demais métodos que se fizerem necessários.

4. Deve também possuir um construtor que inicialize o objeto com 3 coordenadas de pontos 2D. Não é necessário armazenar os 3 pontos como variável membro.

4. Implementar todas as 05 variáveis membros do **Triangle** com **alocação dinamica** de memória. Alocar no construtor, desalocar no destrutor (*new* e *delete*).

5. Implementar operadores para a classe **Triangle**

6. No **main.cpp,** instanciar objetos e armazenar em arrays ou vetores

se vector<>, usar vector< Triangle* > → “move constructors”

Contato

Prof.Antonio Carlos Sobieranski – DEC | A316

E-mail: a.sobieranski@ufsc.br

Inst: @antonio.sobieranski

UNIVERSIDADE FEDERAL
DE SANTA CATARINA