

INF1010 - Estruturas de dados avançadas

Introdução a C++

PUC-Rio

12 de março de 2018

Roteiro

Introdução

- Primeiro programa em C++
- Programação orientada a objetos

Classes em C++

- Atributos e métodos
- Construtores e destrutores
- Herança e polimorfismo

Instanciação e alocação dinâmica

Ponteiros e referência

Input/output

Referências

Primeiro programa em C++

- ▶ Imprimindo "Hello World!":

```
//Meu primeiro programa em C++
#include <iostream>
int main()
{
    std::cout << " Hello World!" << std::endl;
    return 0;
}
```

- ▶ Como compilar o seu programa?
 - ▶ Por linha de comando no Linux: `g++ example.cpp -o example`
 - ▶ Utilizando um IDE (Integrated Development Environment):
Dev-C++ (Somente Windows), Visual Studio (Somente Windows), Netbeans, Eclipse, Code::blocks, QtCreator...

Programação orientada a objetos

- ▶ Programação estruturada: orientada a procedimentos (subrotinas)
- ▶ Programação orientada a objetos: baseada em estruturas (objetos) que contém dados/propriedades (atributos) e comportamento (funções ou métodos).

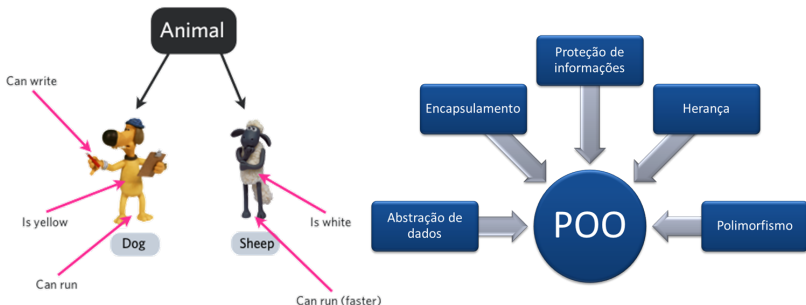


Figure 1: Programação orientada a objetos.

Programação orientada a objetos

- ▶ Classe vs. Objeto:
Classe é o modelo, a definição.
Objeto é a instanciação de uma classe.

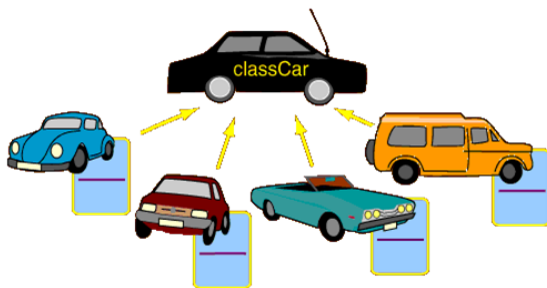


Figure 2: Exemplo de instaciações de objetos de uma classe Carro.

Classes em C++

- Definição de uma classe simples:

```
class Circle
{
public:
    Circle(float x, float y, float radius);

    ~Circle();

    float computeArea();
    void setCenter(float x, float y);
    void setRadius(float radius);

private:
    float _x;
    float _y;
    float _radius;
};
```

Atributos e métodos

- ▶ *Atributos* são elementos (variáveis) pertencentes à classe.
- ▶ *Métodos* são funções associadas a uma classe, e são chamadas pelos objetos pertencentes à classe ou por outros métodos da própria classe.

```
Circle circle1(0.0f, 0.0f, 1.0f);  
Circle* circle2 = new Circle(circle1);  
float area1 = circle1.computeArea();  
float area2 = circle2->computeArea();
```

Atributos e métodos

- ▶ Atributos e métodos podem ser de três modalidades, com relação à sua visibilidade: `private`, `protected` e `public`
 - ▶ **private**: Só podem ser acessados por membros da mesma classe.
 - ▶ **public**: Podem ser acessadas fora do objeto, onde ele estiver definido.
 - ▶ **protected**: Podem ser acessados somente pela classe ou por classes "filhas". (Herança)

Construtores e destrutores

- Construtores: Chamados quando um objeto é criado.

```
// Construtor padrao:  
Circle::Circle()  
{  
    _x = 0.0f;  
    _y = 0.0f;  
    _radius = 0.0f;  
}  
  
// Construtor com lista de inicializacao:  
Circle::Circle(float x, float y, float radius)  
    : _x(x), _y(y), _radius(radius)  
{  
}
```

Construtores e destrutores

► Construtores: (cont.)

```
// Construtor de copia:
Circle::Circle( const Circle& other )
    : _x(other._x),
      _y(other._y),
      _radius(other._radius)
{
}
```

► Destrutor: Chamado quando uma classe é deletada.

```
Circle::~~Circle()
{
    // Desaloca objetos de responsabilidade
    // da classe.
}
```

Herança e polimorfismo

- ▶ Herança: Classes filhas herdam os atributos e métodos da classe pai.
 - ▶ Uma classe filha só consegue acessar atributos e métodos declarados como public ou protected, na classe pai.
- ▶ Polimorfismo em C++:
 - ▶ Métodos com mesmo nome, porém com retornos ou parâmetros diferentes.
 - ▶ Métodos declarados como virtuais podem ser sobrescritos.
 - ▶ Métodos virtuais puros devem ser sobrescritos.

Herança e polimorfismo

```
class Shape
{
public:
    Shape( Color color );
    virtual void draw() = 0;
protected:
    Color _color;
};

class Circle : public Shape
{
    {...}
};

Circle::Circle(float x, float y, float radius,
               Color someColor)
    : Shape(someColor),
      _x(x), _y(y), _radius(radius)
{
}
```

Instanciação e alocação dinâmica

- ▶ A instanciação de objetos é o processo de criar a estrutura lógica dos mesmos na memória.
- ▶ Dois tipos de instanciação são possíveis:
 - ▶ Alocação estática:

```
Circle circle(5.0f, 8.0f, 2.0f);
```

A variável `circle` é destruída quando termina o escopo em que ela foi definida (uma função, uma outra classe, um laço, etc), e seu construtor é chamado.

- ▶ Alocação dinâmica: `new` retorna um ponteiro para a área de memória em que a variável foi criada.

```
Circle* circle = new Circle(5.0f, 8.0f, 2.0f);
```

A variável `circle` nesse caso não é destruída quando o escopo termina! É necessário liberar a memória alocada:

```
delete circle;
```

Ponteiros e referência

- ▶ São ambos mecanismos de indireção.
- ▶ *Ponteiros* guardam endereços de memória de uma variável.

```
int x = 10;  
int* xPointer = &x; //Obtem o endereço de x
```

- ▶ Derreferência: Acesso ao valor armazenado, com o operador *

```
std::cout << *xPointer; //Imprime 10
```

- ▶ Principais usos:
 - ▶ Manipulação e gerenciamento de memória dinâmica;
 - ▶ Criação e destruição de objetos de formas e em momentos especiais.

Ponteiros e referência

- ▶ *Referências* são como "apelidos" para variáveis.

```
int x = 10;  
int& rx = x;  
rx = 5; //A mudança eh aplicada em x.
```

- ▶ São utilizadas como variáveis comuns (maior facilidade de uso e transparência).
- ▶ Tem obrigatoriamente que ser inicializadas na sua declaração.
- ▶ Principal uso: passagem de parâmetros.

```
void somefunction(int& par);  
  
int x = 10;  
somefunction(x);
```

Input/output

- Saída padrão: imprimindo na tela.

```
#include <iostream>
int main()
{
    int year = 2017;
    std::cout << "Hello world!" << "\nIt's "
               << year << "!" << std::endl;
    return 0;
}
```

- Entrada padrão: lendo do teclado.

```
int number;
std::cin >> number;
```

Referências

- ▶ Tutoriais, referências de funções, fórum, etc:
<http://www.cplusplus.com>
- ▶ Livros:
 - ▶ Drozdek, Adam. Data Structures and Algorithms in C++. Pacific Grove, CA: Brooks/Cole, 2001.
 - ▶ Paul J. Deitel. 2010. C++ how to Program. P.J. Deitel, H.M. Deitel (7th ed.). Pearson Education.
 - ▶ Scott Meyers. 1998. Effective C++ (2nd Ed.): 50 Specific Ways to Improve Your Programs and Designs. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
 - ▶ Scott Meyers. 2014. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14 (1st ed.). O'Reilly Media, Inc.
- ▶ E muito material na Internet...