INF1010 - Estruturas de dados avançadas
Introdução a C++

Templates

PUC-Rio

25 de abril de 2018

Polimorfismo em C++

Funções e classes com:

- Mesmo nome
- Mesma semântica
- Diferentes tipos de dados

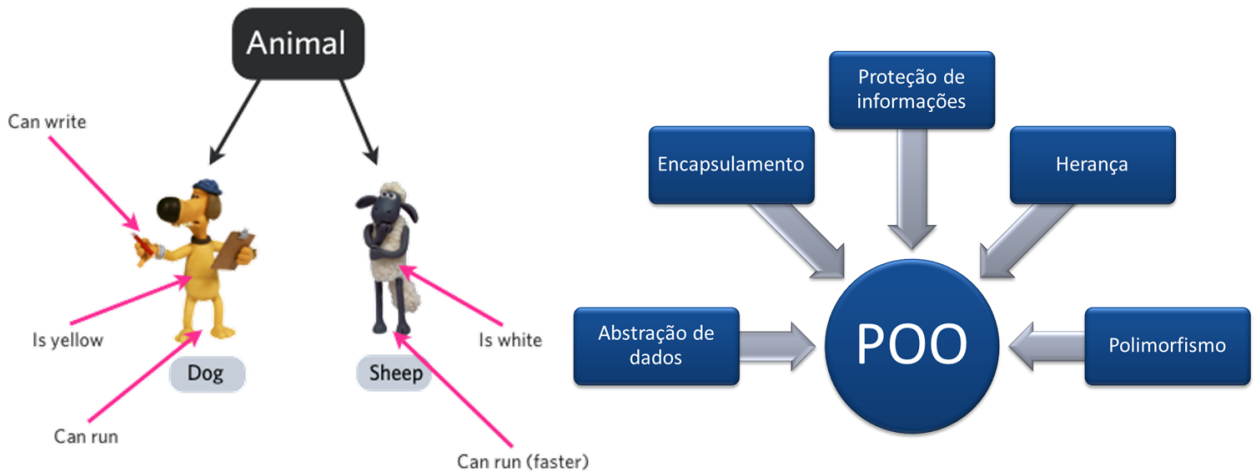

Figure 1: Programação orientada a objetos.

Polimorfismo em C++

Às vezes com a mesma sintaxe:

```
int sum( int a, int b )
{
  return a + b;
}

float sum( float a, float b )
{
  return a + b;
}

Point sum( Point a, Point b )
{
  return a + b;
}
```

Uma solução mais elegante: *template*

```
template <typename T>
T sum( T a, T b )
{
   return a + b;
}
```

- T pode ser um tipo de dados qualquer: int, float, classes definidas pelo usuário...
- Funciona com todos os tipos para os quais o operador + é definido.

Tipos de template

- Função template

```
template <typename identifier>
function_declaration;

template <class identifier>
function_declaration;
```

class e *typename* são ambas válidas no uso básico e mais comum de templates.
Uso:

```
int x, y;
int result = sum(x, y);

Point p1, p2;
Point p3 = sum(p1, p2);
```

Tipos de template

- Classe template
  Exemplo:

```
template <typename T>
class ClassName
{
public:
  T getVariable();
  void setVariable( T value );

private:
  T variable;
  T container[100];
};
```

Tipos de template

- Classe template
  Uso:

```
ClassName<int> myIntObject;
myIntObject.setVariable(10);

ClassName<double> myDoubleObject;
double value = myDoubleObject.getVariable();
```

Outros casos:

- Templates de valores e não de tipos:

```
template <int N>
void foo()
{
  float items[N];
  ...
}

//Uso:
foo<100>();
```

- Lista de templates, e não apenas um:

```
template <typename T, int N>
void foo()
{
  T items[N];
  ...
}
```

Outros casos:

- Valores default:

```
template <typename T = int>
class Foo
{
   ...
};
```

E, neste caso, o uso pode ser:

```
Foo<int> f1;
Foo<double> f2;
Foo<> f3;
```

Mais sobre templates

- Um template não é uma classe ou uma função. Um template é um **padrão** que o compilador usa para gerar uma familia de classes ou funções.
- Eu escrevo menos, mas não gero menos código. O compilador faz isso por mim.
- Para que o compilador possa gerar o código, ele precisa conhecer tanto a definição do template (não só a sua declaração), quanto os tipos específicos utilizados para "preencher" o template.
- Não é possível manter a separação de declaração e definição em .h e .cpp (exceto com alguns "truques") :-(

Um caso de template da STL

std::vector

- Container de alocação contígua de elementos.
- Acesso em O(1).
- Inserção e remoção em O(n).
- Seu tamanho é modificável dinamicamente e isso é gerenciado automaticamente.
- Internamente, usam arrays alocados dinamicamente. Quando novos elementos são inseridos, isso pode significar alocar um novo array e mover todos os elementos. Por isso, normalmente utilizam armazenamento extra (capacity) para acomodar um possível crescimento.

Um caso de template da STL

std::vector

- Exemplo de uso:

```
#include <vector>

std::vector<int> container;

for( int i = 1; i <= 10; i++ )
{
  container.push_back(i);
}

std::cout <<
  "Terceiro elemento de " <<
  container.size() << ": " <<
  container[2] << std::endl;
```

Saída: Terceiro elemento de 10: 3

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...