UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Fundamentos de Programação

ET43I

Orientada a Objetos

Templates, metaprogramação

Aula 8

Prof. Daniel Cavalcanti Jeronymo

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia Eletrônica – 3º Período
2016.1

Plano de Aula

- Templates
- Templates de funções
- Templates de classes
- Metaprogramação

Templates

- Padrões que permitem *reutilização de classes e objetos* na linguagem C++
  - Herança
  - Composição
- Padrão que permite *reutilização de código* na linguagem C++

Templates

- Templates
  - Utilizado para implementar **programação genérica**
  - Funções e classes trabalham com **tipos genéricos**
  - Um mesmo pedaço de código é escrito uma única vez e aplicado para diferentes **tipos de dados**

Conventional wisdom reveres complexity.

Templates

- Sintaxe:

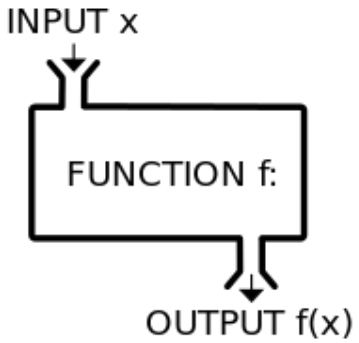
```
template < class/typename Tipo >

template < class/typename Tipo1, class/typename Tipo2, ... >
```

- Indica ao compilador que o código que segue irá trabalhar com um **tipo genérico** (não especificado)

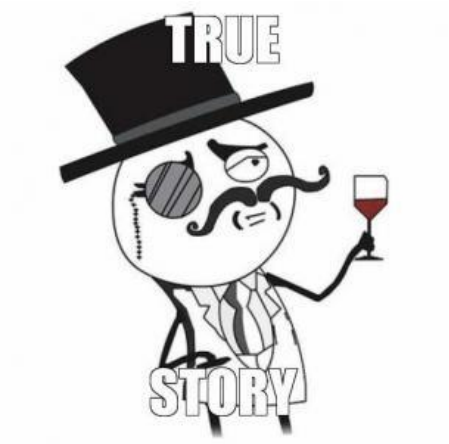
- Ao ser utilizado, o compilador irá gerar o código real a partir do código modelo (*template)* substituindo **Tipo** pelo tipo de dado desejado

Templates

- Duas aplicações principais
  - Funções



  - Classes



Templates de funções

```
#include <iostream>

using std::cout;
using std::endl;

template <class Tipo>
Tipo max(Tipo a, Tipo b)
{
   return a > b ? a : b;
}

int main()
{
   // O compilador identifica o tipo implicitamente (char)
   cout << max('a', 'b') << endl;

   // O programador define o tipo explicitamente
   cout << max<int>('a', 'b') << endl;

   // O compilador identifica o tipo implicitamente (int)
   cout << max(97, 98) << endl;

   return 0;
}
```

Templates de funções

- Ao identificar uma chamada para a função **max**, o compilador identifica o tipo e cria uma cópia da função com aquele tipo

```
int max(int a, int b)
{
   return a > b ? a : b;
}

char max(char a, char b)
{
   return a > b ? a : b;
}
```

Templates de classes

- Mesma sintaxe do uso em funções

```
#include <iostream>

using std::cout;
using std::endl;

template <class T>
class Calculadora
{
  public:
    T multiplica(T x, T y)
    {
       return x * y;
    }

    T soma(T x, T y)
    {
       return x + y;
    }
 };
```

Templates de classes

- Necessário instanciar o objeto com o **Tipo**

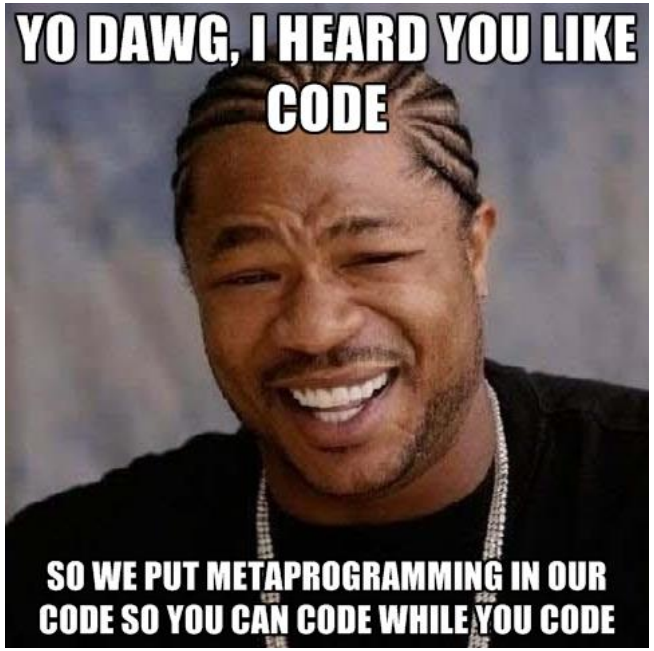
```
int main()
{
   Calculadora<int> calc;

   cout << calc.multiplica(2,3) << endl;

   return 0;
}
```

- calc é um objeto de **tipo parametrizado**

Metaprogramação

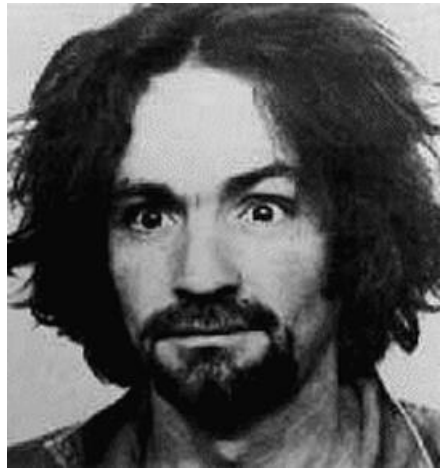
- Habilidade de um código de **modificar a si mesmo**

- Permite que um trecho de código crie, ou altere, código



Metaprogramação

```
#include <iostream>

// Template para uma "classe" Fatorial
template <int N>
struct Fatorial
{
   // variáveis constantes podem ser inicializadas diretamente
   static const int value = N * Fatorial<N - 1>::value;
};

// Template especializado para o "tipo" zero
template <>
struct Fatorial<0>
{
   static const int value = 1;
};

int main(void)
{
  std::cout << "Fatorial de 5: "  << Fatorial<5>::value  <<
std::endl;

  return 0;
}
```



Exercício

- Utilize templates para definir uma lista encadeada para tipos genéricos

- Basear-se no código **listaEncadeada.cpp**