# Programação genérica: templates de funções e templates de classes

Utilizando C++, o programador tem a capacidade de criar seus próprios tipos de dados, seja utilizando structs, assim como em C, como criando classes, adicionando comportamento aos objetos durante o ciclo de vida de um sistema. Além disso é possível que se crie classes com métodos e funções com o mesmo nome, mas assinaturas diferentes para cada conjunto de parâmetros que são passados, conseguindo-se uma forma um pouco mais homogênea de se programar.

Entretanto nota-se que, somente com esses recursos, muito código em um sistema ainda é repetido em vários lugares pois a sobrecarga de métodos e funções só existe para se resolver o seguinte problema: *"como uma função, com o mesmo, executa de forma diferente para tipos de entrada distintos."* e não *"como se executar o mesmo procedimento para diferentes tipos"*.

Só para dar um exemplo, imagine uma lista encadeada onde cada um dos seus nós armazenam um objeto da classe Musica. A implementação dessa lista não é muito diferente de uma outra que, por exemplo, cada nó guardaria um objeto da classe Pessoa, ou Veículo, por exemplo. Ela teria as mesmas funcionalidades, os mesmos métodos, que fariam as mesmas ações. Então fica fácil de se imaginar que seria muito útil se existisse uma maneira de se criar uma lista encadeada (por exemplo) genérica, que tivesses todos os recursos que já estudamos, porém implementados somente uma vez mas que pudesse armazenar qualquer tipo de dados nos seus nós. Essa funcionalidade de se criar código genérico existe existe em C++ com os **templates** que podem ser utilizados tanto em funções como em classes.

Templates são basicamente tipos genéricos, que podem ter seu uso indicado em funções e classes, que somente são resolvidos pelo compilador em um estágio mais avançado, quando se está utilizando a class/função. Você pode com templates criar, por exemplo, uma lista encadeada para um tipo de dados XX, e somente quando for utilizar a lista no seu código informar que XX é um **int**, ou uma **Musica** ou uma **Pessoa**. Isso dá uma flexibilidade muito grande para o seu código e garante que ele seis mais reusável e enxuto.

Veja o exemplo abaixo com sobrecarga:

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

As quatro funções acima tem a mesma implementação e podem ser substituídas por uma única função com **template**:

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

## Slides

- **Conjunto de slides do grupo de professores LP1 (material 2020.6)**
  - https://docs.google.com/presentation/d/1LQ5jox5YcGPGxPqg7dE0WOn3nzNrACF9bZzDtup-PFY/edit?usp=sharing
- Exemplo resolvido com array, ordenação e templates: Link

## Textos

- **Geek for Geeks (em ingles)**
  - Templates: Link

- **Wikibooks - Programar em C++ (em portugues)**
  - C++ Programming/Templates: Link

- **Livro C++ como programar (Capítulo 11)**
  - DEITEL, Harvey M.; DEITEL, Paul J. C++ como programar. 5. ed. São Paulo: Pearson Prentice Hall, 2006. 1163 p. ISBN: 9788576050568

## Vídeos

- **Aula sobre o tema referente aos slides do grupo de professores LP1 (material 2020.6)**
  - https://drive.google.com/file/d/1BpbPwwyJTPjF5UbxakF8hOmYIv0r5-4h/view?usp=sharing

- **The cherno (Ingles, mas podem usar as legendas do Youtube)**
  - Templates : https://www.youtube.com/watch?v=I-hZkUa9mIs