Introdução a Orientação a Objetos[1][2]

Parte II: Polimorfismo – Vector

A orientação a objetos (OO) é um paradigma de programação que se baseia em 03 (três) pilares. Neste roteiro, você estudará de forma prática sobre o segundo pilar – o polimorfismo. Para isto, utilizaremos a biblioteca padrão do `C++`, a `STL` (*Standand Template Library*). Especificamente, neste roteiro utilizaremos a classe `vector`, através de exemplos, exercícios de implementação e da referência fornecida pelo site `cplusplus.com`.

1 Introdução conceitual

O polimorfismo é uma propriedade de linguagens de programação que permite que uma mesma entidade apresente diferentes comportamentos. Em `C++`, há quatro diferentes tipos de polimorfismo, dentre os quais dois serão estudados neste roteiro.

Ad-hoc – corresponde ao polimorfismo pontual e específico, em que há um baixo nível de generalização e nenhum reuso de código. Em `C++`, o polimorfismo ad-hoc é implementado através de sobrecarga de funções e operadores. Note que, neste tipo de polimorfismo, é necessário implementar cada um dos diferentes comportamentos que se deseja obter, descartando assim o benefício do reuso de código provido pelo conceito de polimorfismo.

Paramétrico – corresponde ao polimorfismo obtido por meio da *programação genérica*, isto é, a capacidade de escrever código reutilizável independente do tipo de dado usado como entrada/saída. Em `C`, este tipo de polimorfismo é obtido através de ponteiros para o tipo `void`. No entanto, esta abordagem força o uso de ponteiros e não permite verificação de tipos.

Em `C++`, o polimorfismo paramétrico é implementado através de *templates*, que podem ser aplicados a funções ou classes. Especificamente, pode-se elencar um (ou mais) tipos genéricos ao definir uma função ou classe, criando assim um modelo de função/classe (função/classe genérica). Para invocar uma função genérica (ou instanciar um objeto de uma classe genérica), deve-se especificar o(s) tipo(s) a ser(em) usado(s) no lugar do(s) tipo(s) genérico(s) definido(s) no modelo. Caso hajam múltiplos modelos para uma mesma função, é papel do compilador identificar qual modelo utilizar. Também é possível definir especializações de um modelo, isto é, casos específicos em que se deseja definir o comportamento de uma função/classe genérica para um tipo particular de dados.

2 Prática STL: vector

A classe `vector` pode ser utilizada com a inclusão da header `<vector>`. Em códigos `C++`, objetos `vector` tipicamente substituem os arrays tradicionais usados no `C`. Para aprender a utilizar a classe `vector`, acesse a referência

[1] Roteiro de estudo fornecido na disciplina de Linguagem de Programação 1 (LP1) da Universidade Federal do Rio Grande do Norte (UFRN). Disponível em `https://leobezerra.github.io/LP1-2017-1-T05`.

[2] Autor: Leonardo Bezerra (`leobezerra@imd.ufrn.br`).

disponibilizada no GitHub Pages da disciplina e faça os exercícios sugeridos abaixo. Note que você deverá criar dois novos projetos no GitLab para estes exercícios (`lab07` e `lab08`).

(*lab07*) Vetores unidimensionais – na primeira parte deste exercício, você deve escrever uma aplicação que receba um vetor de inteiros (com quantidade de elementos não especificada) e o imprima de forma ordenada. Como algoritmo de ordenação, implemente o *insertion sort*. Em seguida, generalize o seu código para trabalhar com um tipo de dados genérico `T`. Os testes de aceitação se encontram no SIGAA (`aceitacao.zip`).

Extra! – apresente implementações do *insertion sort* e do *quicksort* com *iterators*.

(*lab08*) Vetores bidimensionais – Reescreva a aplicação do exercício *lab02* utilizando *vectors* de tipo genérico.