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.

Sobrecarga – um tipo de polimorfismo pontual e específico, em que há um baixo nível de generalização e nenhum reuso de código. Em `C++`, o polimorfismo por sobrecarga 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 do tipo `vector` tipicamente substituem os arrays tradicionais usados no `C`. Para aprender a utilizar a classe `vector`, acesse a

[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/LP1-2017-2-T04`.

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

referência 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 (`lab02` e `lab03`).

(*lab02*) Vetores unidimensionais – implemente os procedimentos genéricos abaixo, considerando que cada algoritmo recebe como parâmetros objetos do tipo vector:

- concat: concatena dois objetos do tipo vector. Recebe como parâmetros três referências para objetos do tipo vector. O primeiro e segundo parâmetros são as referências para os vector de origem. O terceiro parâmetro é a referência para o objeto vector de destino.
- split: divide um vector em dois. Recebe como parâmetros três referências para objetos do tipo vector e um índice. O primeiro parâmetro é a referência para o vector de origem. O segundo e terceiro parâmetros são as referências para objetos vector que receberão os elementos à esquerda e à direita do índice, respectivamente. Retorna verdadeiro se a operação for bem sucedida, ou falso em caso contrário.
- merge: mescla dois objetos ordenados do tipo vector. Recebe como parâmetros três referências para objetos do tipo vector. O primeiro e segundo parâmetros são as referências para os vector de origem. O terceiro parâmetro é a referência para o objeto vector de destino.
- partition: particiona um vector em dois. Recebe como parâmetros três referências para objetos do tipo vector e uma chave. O primeiro parâmetro é a referência para o vector de origem. O segundo e terceiro parâmetros são as referências para objetos vector que receberão os elementos menores e maiores que a chave, respectivamente. Retorna verdadeiro se a operação for bem sucedida, ou falso em caso contrário.

(*lab03*) Vetores bidimensionais – implemente os procedimentos genéricos abaixo, considerando que cada algoritmo recebe como parâmetros vetores bidimensionais (matrizes) implementados com a classe vector:

- transpose: transpõe uma matriz. Recebe como argumento as referências para duas matrizes implementadas com a classe vector. A segunda matriz deve se tornar a transposta da primeira.
- quadrada: verifica se uma matriz é quadrada. Recebe como argumento a referência para uma matriz implementada com a classe vector.
- simetrica: verifica se uma matriz é simétrica. Recebe como argumento a referência para uma matriz implementada com a classe vector.
- triangular: gera uma matriz triangular a partir de uma matriz simétrica. Recebe como argumento as referências para duas matrizes implementadas com a classe vector, sendo a primeira a matriz simétrica e a segunda a matriz diagonal.