

# Listas e *Arrays*

Gustavo Zambonin

Paradigmas de Programação (UFSC-INE5416)

**Nota:** o código-fonte em C foi compilado com `gcc -lm ine5416_r8.c`. Note que a compilação com `g++` falhará por conta de um recurso chamado *variable-length array*. Uma referência sobre isto pode ser encontrada [aqui](#).

## Parte 1

- A declaração de *arrays* em C pode ser feita de algumas maneiras, utilizando a notação de colchetes (`int numeros[]`, com número de elementos desejado opcional dentro dos colchetes), ou ponteiros (`int *numeros`). Estruturas multidimensionais podem ser criadas aumentando o número de colchetes ou de ponteiros (uma matriz pode ser declarada como `int matriz[][]` ou `int **matriz`). Por conta da tipagem da linguagem, não existem *arrays* que abrigam objetos de tipo diferente. A inicialização destas variáveis pode ser realizada com alocação dinâmica de memória (`malloc` e `calloc`) ou através de notação de chaves englobando elementos. Não existem definições de listas em C, comparado a outras linguagens, além de estruturas de dados como listas ligadas ou circulares.
- Por outro lado, a declaração de listas em Python é mais dinâmica. Por exemplo: `lista = []` atribuirá uma lista vazia à variável correspondente, e também é possível popular a lista com elementos dentro dos colchetes; `[i for i in range(10)]` gerará uma lista onde  $0 \leq i \leq 9$  (este método chama-se *list comprehension*, baseado originalmente na funcionalidade de Haskell). Um *array* funciona de modo levemente diferente, pois esta estrutura é homogênea e [limitada](#) quanto ao tipo dos objetos que armazena.
- *Arrays* como argumentos de funções têm diferenças apenas na notação, em C. O compilador sempre entenderá o argumento como um ponteiro, ou seja, o valor do endereço do primeiro elemento do *array* na memória. Assim como Python, os objetos originais serão modificados se passados para uma função. Para prevenir este comportamento, cópias em memória deverão ser realizadas.

## Parte 2

- `ine5416_r8.c`  
Algumas estratégias foram utilizadas para facilitar a produção do código, como a definição de uma pequena função para retornar o tamanho de um *array*, e o uso de *VLAs* onde possível, evitando criação de *arrays* dinamicamente.
- `ine5416_r8.py`  
A complexidade de espaço destas funções mostra-se claramente maior do que em C, porém isso se paga na facilidade de manipulação dos dados. Funções básicas da linguagem ajudam a simplificar os métodos. Também não é necessário se preocupar com alocação dinâmica de memória.