



Aula 02

Marcos Silvano Almeida marcossilvano@utfpr.edu.br Departamento de Computação UTFPR Campo Mourão

## TAD \_ Tipos Abstratos de Dados



- Um tipo abstrato é aquele que não define a forma como será implementado, mas sim, o modelo que representa e as operações possíveis
  - O TAD não define a estratégia de implementação
  - Na implementação, vamos usar <u>estruturas de dados</u>, como arrays e structs, para construir o TAD.
- Vamos conhecer os seguintes TADs
  - Lista Sequencial
  - Lista Encadeada
    - Lista Duplamente Encadeada
    - Lista Circular
  - o Fila
  - Pilha





- Sequência de dados armazenados contiguamente.
  - o Permite acesso **sequencial** e **aleatório** aos seus elementos
    - Elemento são acessados por índice (int)
  - Operações típicas:
    - Criar e Destruir
    - Adicionar, Inserir e Remover
    - Acessar e Pesquisar

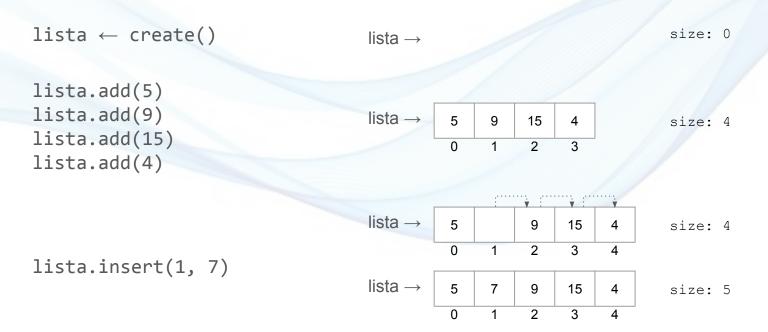
lista →	5	9	15	5	7	1
	0	1	2	3	4	5

size: 4





- Características
  - Acesso sequencial e aleatório.
  - Bom para para inserções/remoções ao final da lista
  - Ruim para inserções/remoções no início ou meio da lista.







Python não possui a estrutura de dados Array, mas o TAD Lista

```
lista = []
                        lista -> []
                                                   size: 0
lista.append(5)
                        lista -> [5]
                                                   size: 1
lista.append(9)
                        lista -> [5, 9]
                                                   size: 2
lista.append(15)
                        lista -> [5, 9, 15] size: 3
lista.append(4)
                        lista -> [5, 9, 15, 4] size: 4
lista.insert(1, 7)
                                                size: 5
                        lista -> [5, 7, 9, 15, 9]
                                                                        size: 0
lista ← create()
                                      lista →
lista.add(5)
lista.add(9)
                                      lista →
                                               5
                                                        15
                                                             4
                                                                        size: 4
lista.add(15)
                                               0
lista.add(4)
                                      lista →
lista.insert(1, 7)
                                               5
                                                            15
                                                                 4
                                                                        size: 5
```





Implementação da Lista Sequencial





- Implementada clássica utiliza vetor
  - Vetor redimensionável, alocado em Heap
  - TAD será definida como um struct, contendo:
    - Capacidade do vetor → tamanho real do vetor
    - Tamanho da lista → quantidade de elementos na lista (posições ocupadas)
    - Ponteiro para o vetor alocado em heap



## O tipo Lista Sequencial



Manipulação da Lista Sequencial ocorre por meio das operações (funções)

```
VectorList* list1 = list_create(10);
list_push_back(list1, 5); // adiciona ao final
list_push_back(list1, 9);
list_push_back(list1, 15);
list_push_back(list1, 5);
list_push_back(list1, 7);
list_pop_back(list1); // remove do final
list_print(list1);
```



#### Operações de criação e acesso



```
// Cria e prepara lista. Retorna NULL se erro.
VectorList* list create(int capacity);
// Libera memoria alocada para o vetor.
VectorList* list destroy(VectorList* this);
// Realoca vetor interno e move elementos. Retorna NULL se erro.
bool list resize(VectorList* this);
void list print(VectorList* this);
                                   // Imprime o conteúdo da lista.
int list_get(VectorList* this, int pos); // Obtém elemento na posição
int list find(VectorList* this, int elem); // Devolve posição do elemento ou -1
                                     // Informa se lista vazia.
bool list_is_empty(VectorList* this);
bool list_is_full(VectorList* this); // Informa se lista cheia.
```



#### Operações de modificação



```
// Insere elemento ao final da lista (false/erro)
bool list push back(VectorList* this, int elem);
// Remove elemento do final da lista (false/erro)
bool list_remove(VectorList* this);
// Insere elemento na posição especificada, abrindo espaço para o novo elemento
bool list_insert(VectorList* this, int elem, int pos);
// Insere elemento em ordem crescente
bool list insert_sorted(VectorList* list, int elem);
// Remove elemento da posição especificada e ajusta lista (fecha "buraco")
bool list remove(VectorList* this, int pos);
```

