Estruturas de Dados

Listas

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

- Uma lista é uma forma simples de relacionar os elementos de um conjunto.

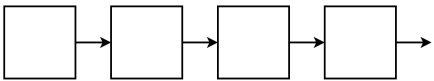


Figura 1: Lista simples.

- Exemplos de conjuntos:
  - números inteiros,
  - notas de alunos,
  - funcionários de uma empresa,
  - itens de estoque, etc.

- Tipos básicos de listas, de acordo com a forma de encadeamento dos itens.

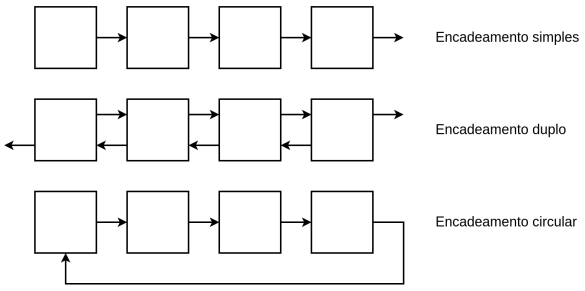


Figura 2: Tipos de listas.

- **Aplicações de listas encadeadas**
  - Implementação de pilhas, filas e grafos.
  - Execução de operações aritméticas em números inteiros longos.
  - Manipulação de polinômios.
  - Representação de matrizes esparsas.
  - Visualizador de imagens
  - Página anterior e seguinte no navegador da web.
  - Music Player (lista circular).
  - Listas circulares duplamente encadeadas são usadas para implementação de estruturas de dados avançadas como Fibonacci Heap.

- Exemplos de operações possíveis:
  - Criar uma lista vazia.
  - Inserir um novo item imediatamente após o i-ésimo item.
  - Retirar o i-ésimo item.
  - Localizar o i-ésimo item.
  - Localizar um item a partir de um valor particular.
  - Combinar duas listas.
  - Partir uma lista linear em duas.
  - Fazer uma cópia da lista.
  - Ordenar os itens da lista.

- Definição de uma lista de inteiros.

Código 1: List.h

```
1  typedef struct List List;
2
3  List *List_alloc();
4  void  List_free(List *l);
5  void  List_insert(List* l, int value);
6  int   List_remove(List* l, int value);
7  int   List_getLength(List *l);
8  void  List_print(List *l);
```

- Há duas maneiras básicas de implementação:
  - Usando vetores.
  - Usando estruturas encadeadas (através de ponteiros).

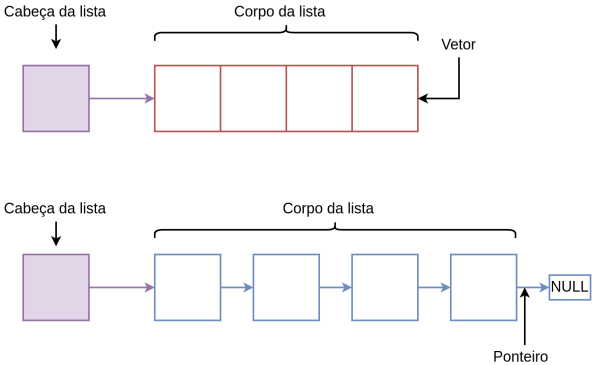
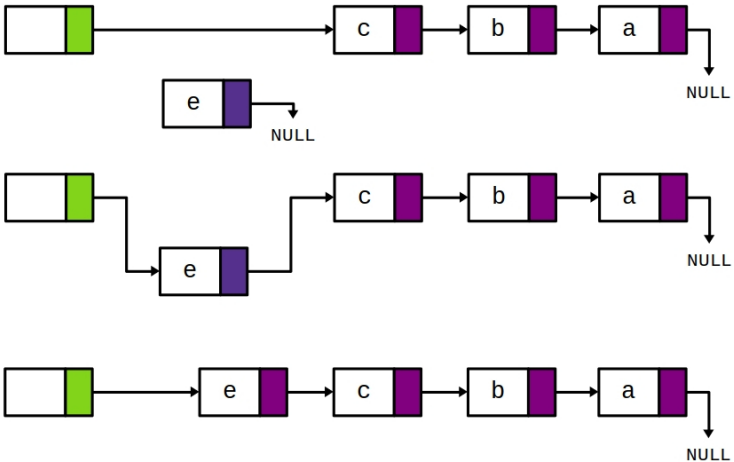


Figura 3: Tipos de implementação.

- **Implementação usando um vetor**
  - **Vantagem**:
    - Economia de memória (não requer memória para armazenar ponteiros).
    - A lista pode ser percorrida em qualquer direção.
    - A inserção de um novo item pode ser realizada após o último item com custo constante.
  - **Desvantagens**:
    - Custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso.
    - O tamanho máximo da lista tem de ser definido em tempo de compilação.

- **Implementação usando estruturas encadeadas**
  - **Vantagem**:
    - Permite utilizar posições não contíguas de memória.
    - É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.
    - A inserção de um novo item pode ser realizada no início da lista com custo constante.
    - O tamanho máximo da lista é definido em tempo de execução.
  - **Desvantagens**:
    - Cada item é encadeado com o seguinte usando um ponteiro (requer mais memória).
    - O acesso aos itens no meio da lista tem que ser realizados sequencialmente a partir do primeiro/último item.

- Inserção na implementação encadeada



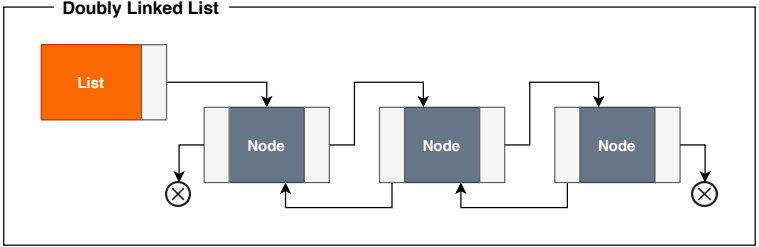
Exercício 1

Implemente as funções do arquivo `List.h` usando um vetor para armazenar os dados de uma lista de números inteiros.

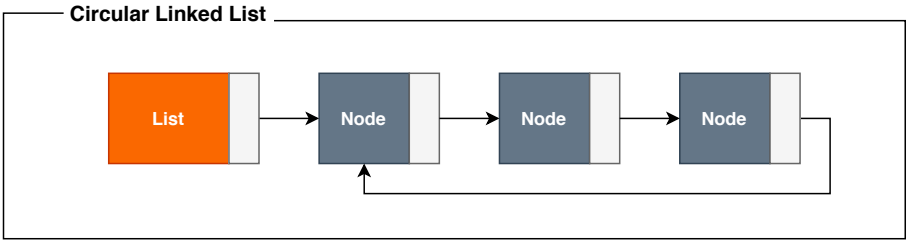
Exercício 2

Implemente as funções do arquivo `List.h` usando estruturas encadeadas para armazenar os dados de uma lista de números inteiros.

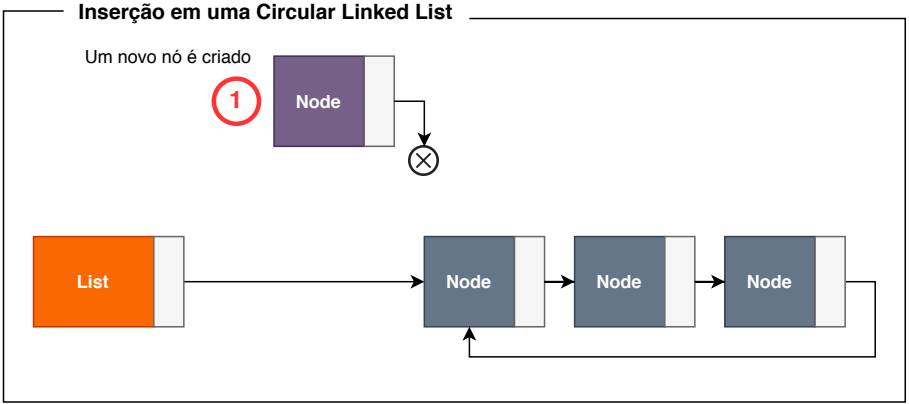
- Em uma lista duplamente encadeada, cada nó possui um ponteiro para o nó anterior e outro para o nó sucessor.
- Permite percurso em duas direções.

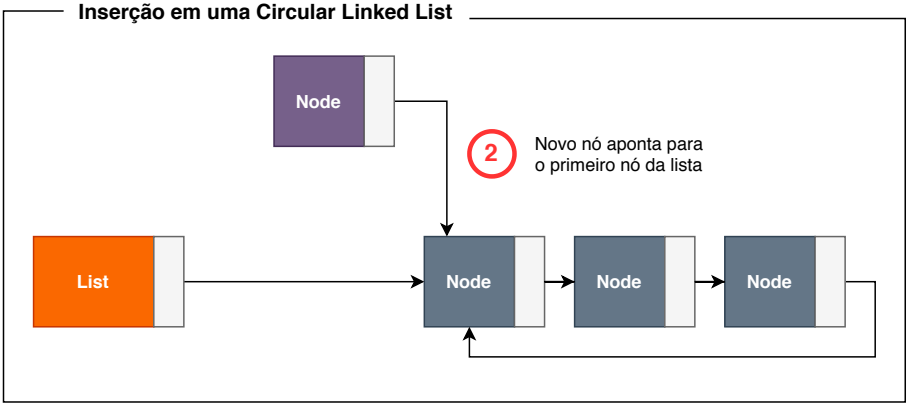


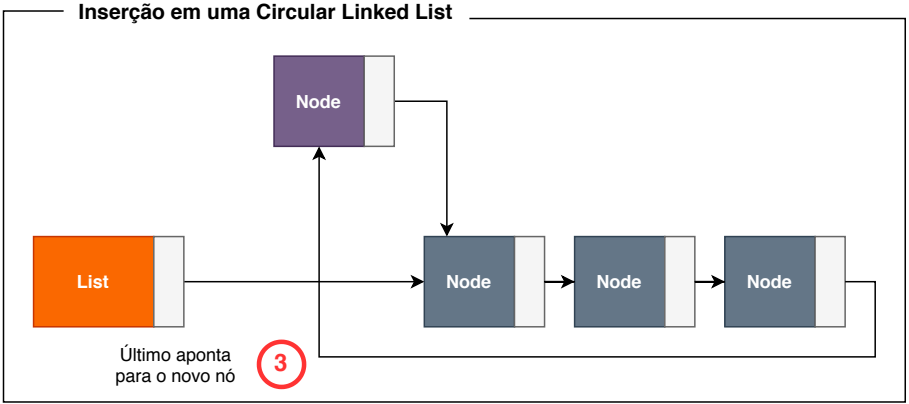
- Em uma lista circular o último nó aponta para o primeiro nó, permitindo reiniciar o percurso pela lista.

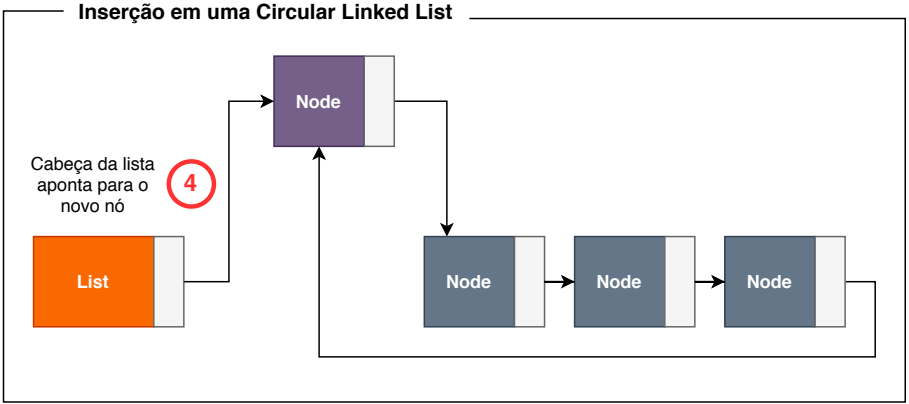


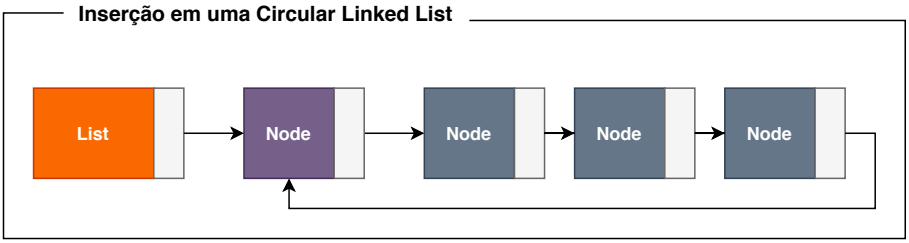
- Qualquer nó pode ser um ponto de partida.
- Podemos percorrer a lista inteira começando em qualquer ponto.
- Só precisamos parar quando o primeiro nó visitado for visitado novamente.
- Útil para implementação de fila. Não precisamos manter ponteiros para a frente e para trás se usarmos uma lista ligada circular.


Inserção em uma Circular Linked List
Um novo nó é criado
1
Node
List
Node
Node
Node

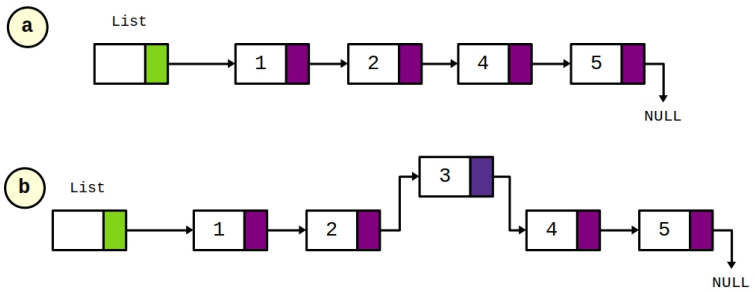

Inserção em uma Circular Linked List
Node
2
Novo nó aponta para
o primeiro nó da lista
List
Node
Node
Node


Inserção em uma Circular Linked List
Node
List
Node
Node
Node
Último aponta
para o novo nó
3


Inserção em uma Circular Linked List
Node
Cabeça da lista aponta para o novo nó
4
List
Node
Node
Node


Inserção em uma Circular Linked List
List
Node
Node
Node
Node

- **Lista ordenada** é uma lista encadeada em que os itens aparecem em ordem.
- Para manter essa ordem, cada item inserido na lista encadeada deve ser corretamente posicionado entre aqueles já existentes na lista.



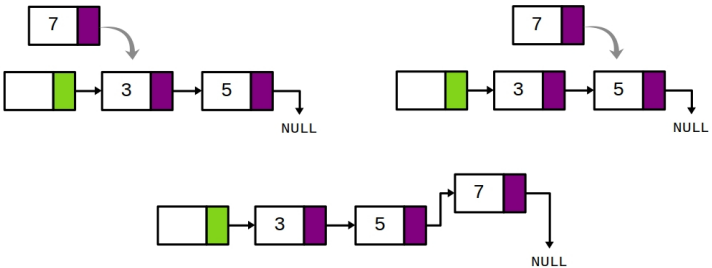
- **Função de inserção ordenada**

```
1  void List_insertSorted(List *l, int value);
```

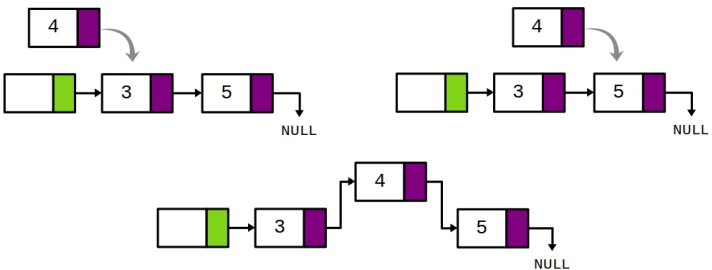
- Assim como na função de busca é necessário realizar **comparações** para realizar uma inserção em ordem.

- **Inserção em Lista Ordenada**
  - *Caso 1*: lista vazia

- **Inserção em Lista Ordenada**
  - *Caso 2*: lista de tamanho **n**



- **Inserção em Lista Ordenada**
  - *Caso 2*: lista de tamanho **n**



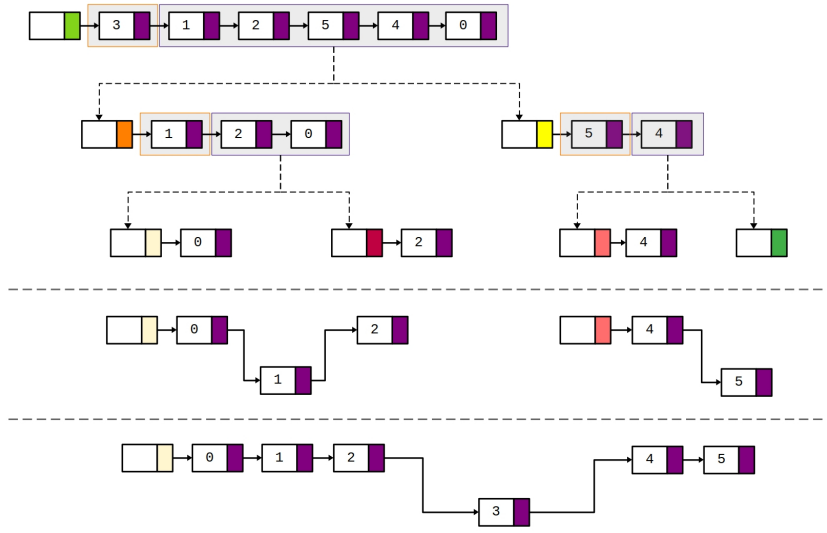
Exercício 3

Implemente a função de inserção ordenada para a implementação de lista com um vetor.

Exercício 4

Implemente a função de inserção ordenada para a implementação de lista com estruturas encadeadas.

- Quick Sort



- Implementação de uma lista genérica.

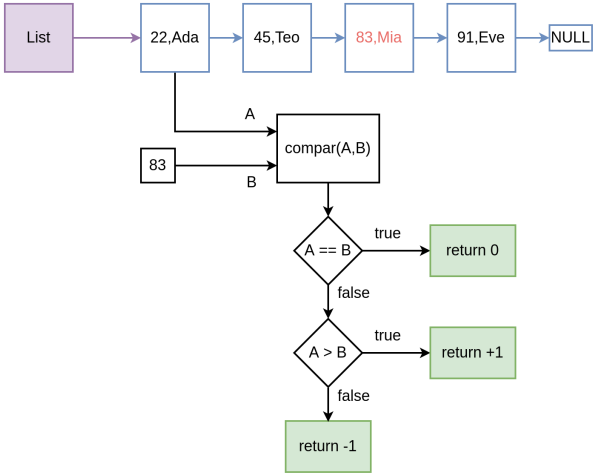
Código 2: List.h

```
1  List *List_alloc();
2  void  List_free(List *l);
3  void  List_insert(List* l, void *value);
4  void *List_find(List* l, void *value, int (*compar)(void*, void*) );
5  void *List_remove(List* l, void *value, int (*compar)(void*, void*) );
6  int   List_getLength(List *l);
7
8  void  List_print(List *l, void (*print)(void *));
```

```c
#include "List.h"

typedef struct Node {
    void *value;
    struct Node *next;
} Node;

struct List {
    int length;
    Node *first;
};

List *List_alloc() {
    List *l = malloc(sizeof(List));
    if (l != NULL) {
        l->first = NULL;
        l->length = 0;
    }
    return l;
}
```

```
1  void List_insert(List* l, void *value){
2      Node *new = NULL;
3      if (l != NULL && value != NULL) {
4          new = malloc(sizeof(Node));
5          new->value = value;
6          new->next = NULL;
7
8          if (l->first == NULL)
9              l->first = new;
10         else {
11             new->next = l->first;
12             l->first = new;
13         }
14     }
15 }
```

- **Busca**


List
22,Ada
45,Teo
83,Mia
91,Eve
NULL
A
83
B
compar(A,B)
A == B
true
return 0
false
A > B
true
return +1
false
return -1

```c
void *List_find(List* l, void *value, int (*compar)(void*, void*) ) {
    Node *n = NULL;

    if (l && value && compar) {
        n = l->first;

        while (n != NULL) {
            if (compar(n->value, value) == 0) {
                return n->value;
            }
            n = n->next;
        }
    }
    return n;
}
```

- Para realizar a comparação adequada dos itens da lista a função

```
int (*compar)(void*, void*)
```

deve seguir os seguintes critérios:

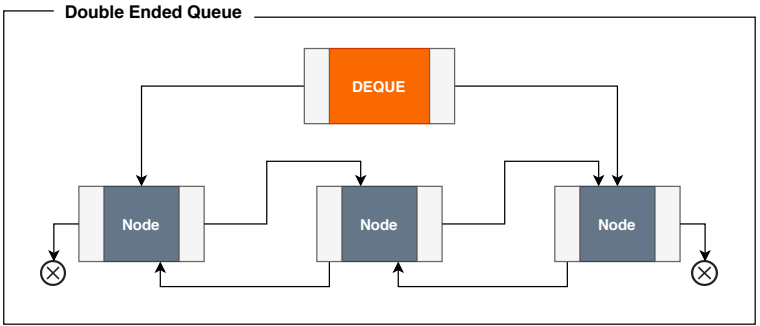
- A função retorna um número < 0 se o primeiro elemento for menor que o segundo,
- a função retorna um número > 0 se o primeiro elemento for maior que o segundo,
- a função retorna 0 se os elementos forem iguais.

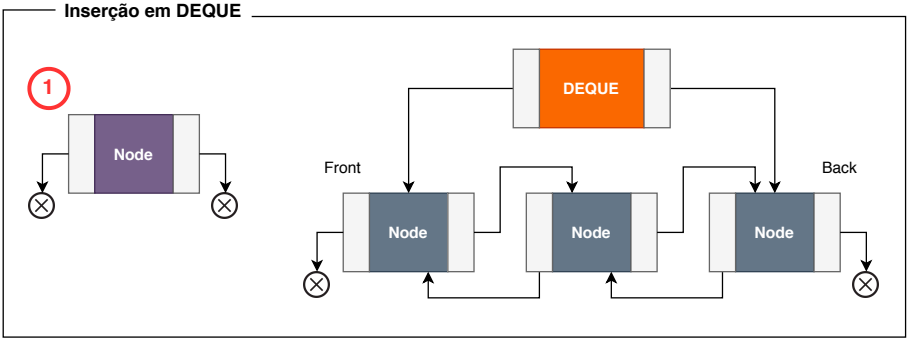
Exercício 5

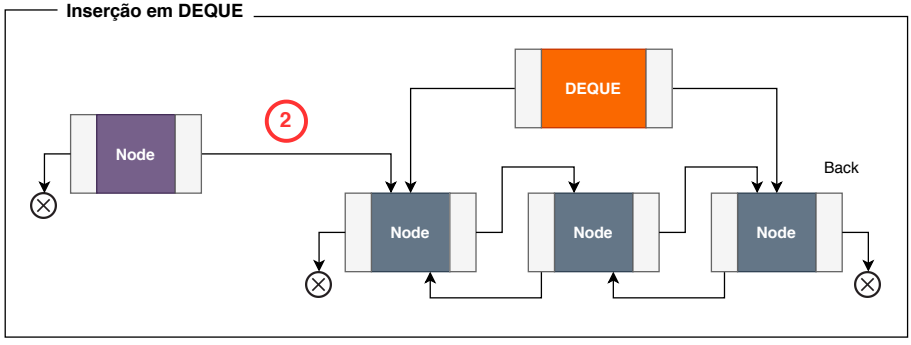
Implemente um programa para o cadastro de Usuários. Cada usuário possui um login na forma de um e-mail e uma senha de oito dígitos numérica. O programa deve apresentar as opções de criar, listar e excluir usuários. Apresente um menu com as opções e use uma lista genérica para armazenar os usuários.

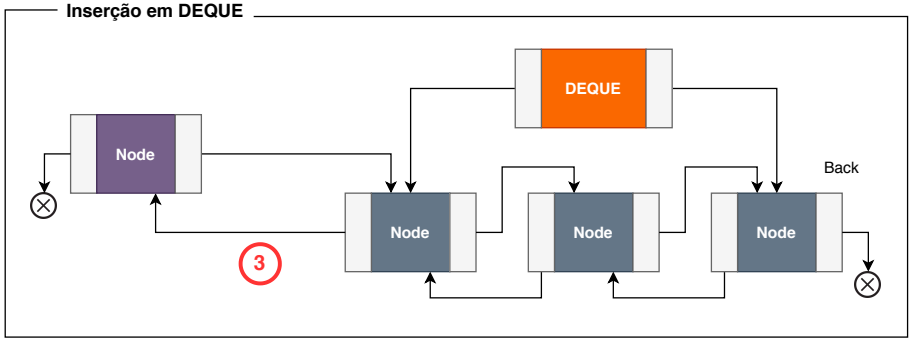
- Criar: aloca dinamicamente memória para o tipo 'usuário' que você definiu, lê pelo console um login e senha e guarda o novo usuário em uma lista.
- Listar: apresente os usuários armazenados na lista até o momento.
- Excluir: Remove um usuário da lista. Use o login do usuário como uma forma de localizá-lo na lista.

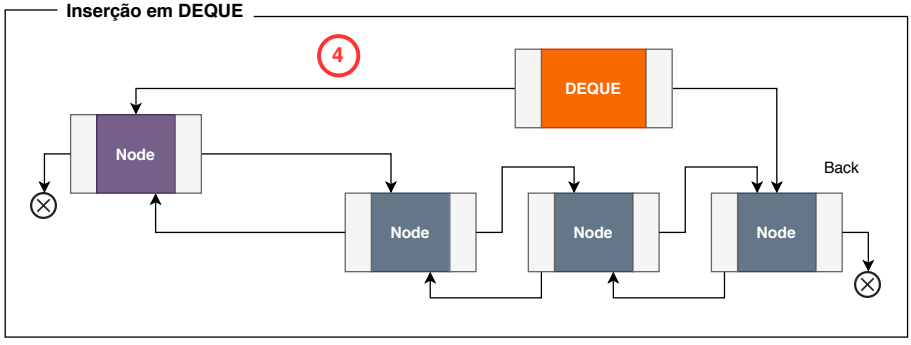
- Com uma alteração na definição da lista duplamente encadeada, pode-se criar uma estrutura de dados mais genérica.
- DEQUE ou *Double Ended Queue* é uma estrutura que permite inserção e remoção no início ou no fim de uma lista de itens, além de permitir o percurso na lista nos dois sentidos, para frente e para trás.




Inserção em DEQUE
1
Node
DEQUE
Front
Back
Node
Node
Node


Inserção em DEQUE
DEQUE
2
Node
Back
Node
Node
Node


Inserção em DEQUE
DEQUE
Node
Back
Node
Node
Node
3


Inserção em DEQUE
4
DEQUE
Node
Back
Node
Node
Node

Exercício extra

Crie uma implementação de *Double Ended Queue* com base no arquivo de cabeçalho `DEQ.h` (*link*).