**Exercícios 06 :: Lista Encadeada**

Instruções Gerais

- Utilize os arquivos trabalhados na última aula e acrescente as funções solicitadas pelos exercícios abaixo: **linkedlist-cpp.zip**.
- Implemente os exercícios em C++, como funções dentro da classe **LinkedList**.
- Ao final, envie o arquivo main.cpp pelo Moodle.

# Operações Básica e Verificações

1. Escreva uma função para informar se a lista está vazia.
   ```
   bool is_empty();
   ```

2. Escreva uma função para calcular e devolver a quantidade de elementos na lista.
   ```
   int size();
   ```

3. Escreva uma função para retornar o endereço do primeiro Node que possui o valor "key" informado ou **nullptr**, caso não exista.
   ```
   Node* find(int elem);
   ```

4. Escreva uma função para simular o acesso por posição (índice) na Lista Encadeada. A função deve retornar o endereço do Node na posição especificada. Considere que, assim como em um array, as posições na lista iniciam em 0 (zero).
   ```
   Node* get(int pos);
   ```

5. Escreva uma função para imprimir o valor (campo "key") do último Node da lista. Lembre-se que o último Node da lista não possui próximo, isto é, `node->next == nullptr`.
   ```
   void print_last(LinkedList* list);
   ```

6. Escreva uma função para verificar se duas listas são iguais. Listas iguais contém os mesmos elementos, na mesma ordem.
   ```
   bool equals(LinkedList* list1, LinkedList* list2);
   ```

7. Escreva uma função para verificar se a lista está ordenada crescentemente.
   ```
   bool is_sorted(LinkedList* list);
   ```

# Inserções e Remoções

8. Escreva uma função para remover o primeiro elemento da lista.
   ```
   bool pop_front();
   ```

9. Escreva uma função para inserir o inteiro passado como um novo Node após o final da lista.
   ```
   bool push_back(int elem);
   ```

10. Escreva uma função para adicionar em “lote”: recebe um array de inteiros (e seu tamanho) e adiciona seus elementos ao final da lista. Note que é necessário criar um Node para cada novo elemento.
    **`bool push_back(int n, int vec*)`**

11. Escreva uma função para remover o último elemento da lista.
    **`bool pop_back();`**

12. Escreva uma função para inserir o valor “key” na lista, em ordem crescente de id.
    **`bool insert_sorted(int key)`**

    **Exemplo:**

```
LinkedList* list1 = new LinkedList;
list1->insert_sorted(10); // head -> [10]
list1->insert_sorted(5);  // head -> [5] -> [10]
list1->insert_sorted(7);  // head -> [5] -> [7] -> [10]
list1->insert_sorted(2);  // head -> [2] -> [5] -> [7] -> [10]
list1->insert_sorted(15); // head -> [2] -> [5] -> [7] -> [10] -> [15]
```

13. Escreva uma função para buscar e remover o primeiro Node que contenha o valor “key” informado.
    **`void remove(int key)`**

## Criações de novas listas

14. Escreva uma função para realizar a cópia profunda (*deep copy*) de uma lista. Uma cópia profunda consiste em criar e devolver uma nova lista, contendo novos Nodes com cópias dos valores da lista original (passada por parâmetro). Note que é necessário criar um novo Node para cada um dos elementos da lista original.
    **`LinkedList* deep_copy();`**

15. Escreva uma função para concatenar duas listas em uma nova e a retornar. Não deve alterar as listas originais: crie um novo Node para cada elemento da lista resultante.
    **`LinkedList* concat(LinkedList* list2);`**

16. Escreva uma função para mesclar ordenadamente os elementos de duas listas ordenadas em uma nova, a ser retornada. Não é permitido alterar as listas originais: deve criar um novo Node para cada elemento da lista resultante.

```
Exemplo:
list1  -> [1] -> [3] -> [4]
list2  -> [2] -> [3] -> [6]
merged -> [1] -> [2] -> [3] -> [3] -> [4] -> [6]
```

    **`LinkedList* merge(LinkedList* list2);`**