

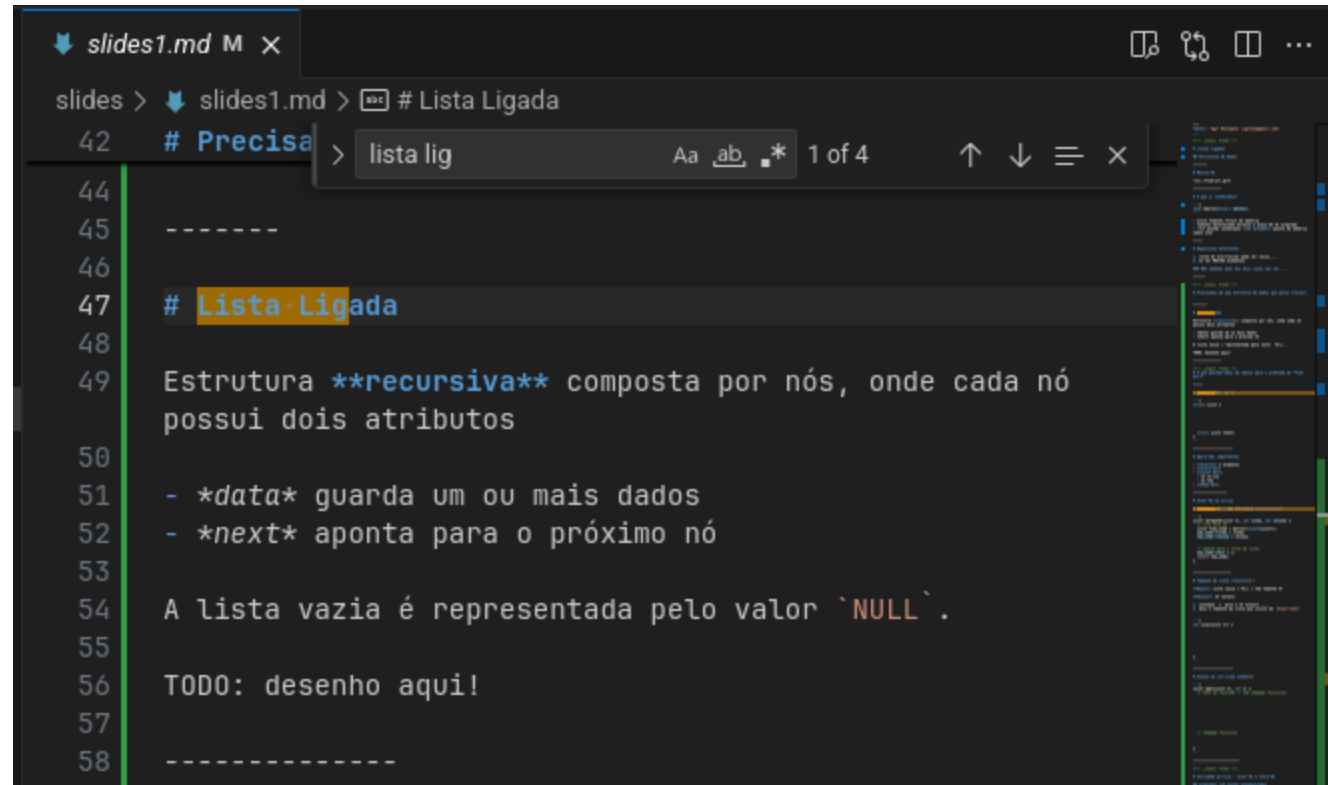
# Listas Ligadas

## Estruturas de Dados

# Materials

1. fontes: <https://github.com/igordsm/listas-ligadas/tree/main>
2. exercícios: [https://us.prairielearn.com/pl/course\\_instance/197989/assessment/2606188](https://us.prairielearn.com/pl/course_instance/197989/assessment/2606188)

# Motivação



The image shows a presentation editor interface with a dark theme. The main slide content is as follows:

```
slides > slides1.md > # Lista Ligada
42 # Precisa
43 > lista lig
44 -----
45
46
47 # Lista Ligada
48
49 Estrutura **recursiva** composta por nós, onde cada nó
50 possui dois atributos
51
52 - *data* guarda um ou mais dados
53 - *next* aponta para o próximo nó
54
55 A lista vazia é representada pelo valor `NULL`.
56
57 TODO: desenho aqui!
58 -----
```

On the right side, there is a sidebar showing a list of slides. The first slide is highlighted, and its content is visible in a smaller view on the right. The search bar at the top right shows the text "lista lig" and "1 of 4".

# O que já conhecemos?

Alocação de arrays usando `malloc`

```
void *malloc(size_t nbytes);
```

- aloca tamanho *fixo* de memória
- tamanho determinado durante a execução do programa
- útil quando conhecemos **de antemão** quanto de memória vamos usar

# Requisitos diferentes

1. lista de ocorrências pode ser vazia....
2. ou ter MUITOS elementos

**Não sabemos qual dos dois casos vai ser....**

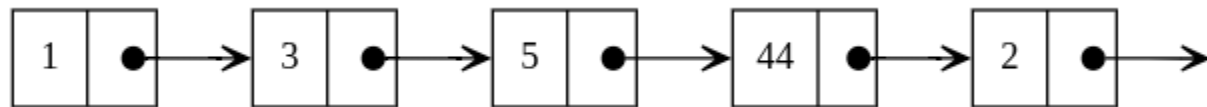
**Precisamos de uma estrutura de dados  
que possa crescer!**

# Lista Ligada

Estrutura **recursiva** composta por nós, onde cada nó possui dois atributos

- *data* guarda um ou mais dados
- *next* aponta para o próximo nó

A lista vazia é representada pelo valor **NULL**.



O que guardaríamos em *data* para o problema do "Find All"?



# Lista Ligada (em C)

```
struct LList {  
  
    struct LList *next;  
}
```

# Operações importantes

- **Acesso** a elemento
- **Iteração**
- **Inserção:**
  - no início
  - no fim
- **Remoção**

# Inserção no início

A Lista Ligada é uma estrutura **recursiva**!

```
LList *prepend(LList *l, int linha, int coluna) {  
    // cria novo nó  
    LList *new_node = malloc(sizeof(LList));  
    new_node->linha = linha;  
    new_node->coluna = coluna;  
  
    // aponta para o resto da lista  
    new_node->next = l;  
    return new_node;  
}
```

# Tamanho da lista (recursivo!)

**Base:** Lista vazia ( `NULL` ) tem tamanho 0!

## Passo: nó *atual*

1. contamos 1 para o nó *atual* ....
2. mais o tamanho da lista que inicia em `atual->next`

```
int size(LList *l) {
```

## Acesso ao $i$ -ésimo elemento

```

LList *get(LList *l, int i) {
    // base da recursão -- sem chamada recursiva!

    // chamada recursiva

}

```

# **Atividade prática - inserção e iteração**

**problemas com testes automatizados**

# Fechamento

|                 | <b>Array</b> | <b>Lista Ligada</b> |
|-----------------|--------------|---------------------|
| Prepend         | $O(N)$       | $O(1)$              |
| Iteração        | $O(N)$       | $O(N)$              |
| Tamanho         | $O(1)$       | $O(N)$              |
| <i>i</i> -ésimo | $O(1)$       | $O(N)$              |