Estruturas de Dados Básicas I

Selan R. dos Santos

DIMAp – Departamento de Informática e Matemática Aplicada
Sala 231, ramal 231, selan.santos@ufrn.br
UFRN

2023.1

Lista Encadeada – Conteúdo

Inserção em Lista Simplesmente Encadeada (L.S.E.)

1. Introdução
2. Algoritmos de Inserção em L.S.E.
3. Referências

L.S.E. – Revisão

Previamente abordamos conceitos básicos, como:

- Como encadeamento em uma lista funciona e sua relação com alocação dinâmica.

L.S.E. – Revisão

Previamente abordamos conceitos básicos, como:

- Como encadeamento em uma lista funciona e sua relação com alocação dinâmica.
- Idiomas recorrentes como o percorrimento de lista.

L.S.E. – Revisão

Previamente abordamos conceitos básicos, como:

- Como encadeamento em uma lista funciona e sua relação com alocação dinâmica.
- Idiomas recorrentes como o percorrimento de lista.
- A criação manual de uma lista.

L.S.E. – Revisão

Previamente abordamos conceitos básicos, como:

- Como encadeamento em uma lista funciona e sua relação com alocação dinâmica.
- Idiomas recorrentes como o percorrimento de lista.
- A criação manual de uma lista.
- Inserção de novos elementos no início e no final de uma lista.

L.S.E. – Revisão

Previamente abordamos conceitos básicos, como:

- Como encadeamento em uma lista funciona e sua relação com alocação dinâmica.
- Idiomas recorrentes como o percorrimento de lista.
- A criação manual de uma lista.
- Inserção de novos elementos no início e no final de uma lista.

L.S.E. – Revisão

Previamente abordamos conceitos básicos, como:

- Como encadeamento em uma lista funciona e sua relação com alocação dinâmica.
- Idiomas recorrentes como o percorrimento de lista.
- A criação manual de uma lista.
- Inserção de novos elementos no início e no final de uma lista.

Vamos abordar agora a operação de **inserção** no meio da lista.

Pré-requisitos para Inserção

Para realizarmos uma inserção de um novo **Nó** em uma lista precisamos identificar primeiramente o local de inserção.

Pré-requisitos para Inserção

Para realizarmos uma inserção de um novo **Nó** em uma lista precisamos identificar primeiramente o local de inserção.

Isto pode ser feito via:

- Indicação da posição (índice) de inserção. Por exemplo, *inserir na posição 3* da lista (indexação começando em zero).

Pré-requisitos para Inserção

Para realizarmos uma inserção de um novo **Nó** em uma lista precisamos identificar primeiramente o local de inserção.

Isto pode ser feito via:

- Indicação da posição (índice) de inserção. Por exemplo, *inserir na posição 3* da lista (indexação começando em zero).
- Aplicação de busca que retorna um ponteiro para o elemento procurado.

Pré-requisitos para Inserção

Uma vez localizado o ponto de inserção, inserimos o novo nó antes (mais comum) ou depois da posição desejada.

Pré-requisitos para Inserção

Uma vez localizado o ponto de inserção, inserimos o novo nó antes (mais comum) ou depois da posição desejada.

De qualquer forma, precisamos sempre obter o ponteiro para o nó **anterior** ao local de inserção desejado.

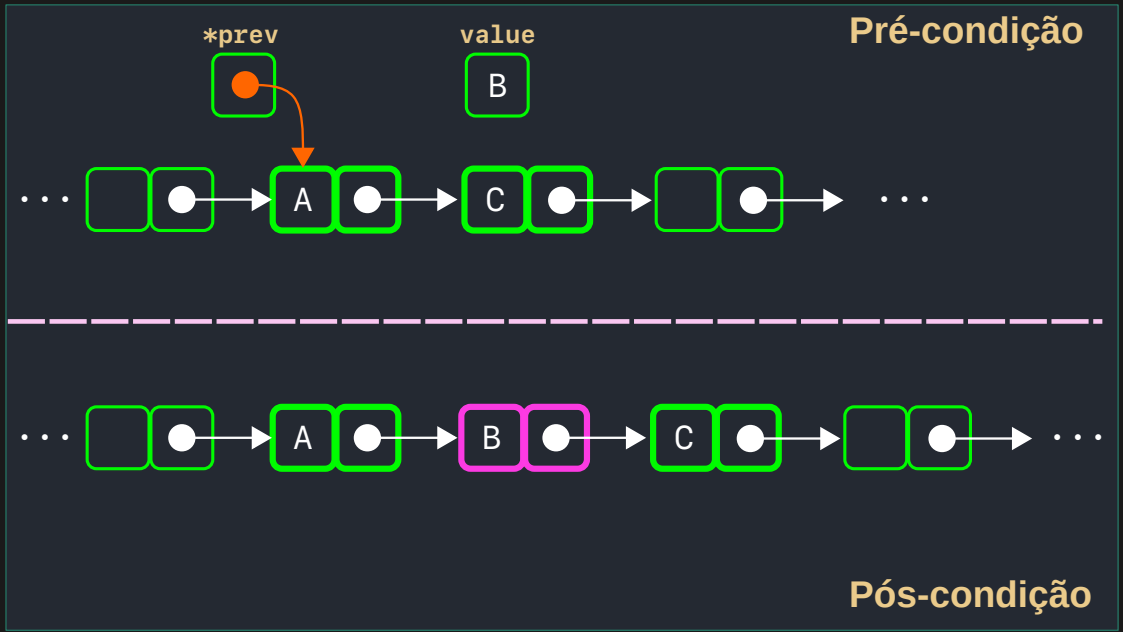
Pré e Pós Condição da Inserção

O diagrama a seguir representa uma lista **antes** e **depois** de uma inserção.

Pré e Pós Condição da Inserção

O diagrama a seguir representa uma lista **antes** e **depois** de uma inserção.

O ponteiro `*prev` aponta para o nó anterior ao local de inserção, e `value` contém o valor a ser inserido.


*prev
value
B
Pré-condição
A
C
A
B
C
Pós-condição

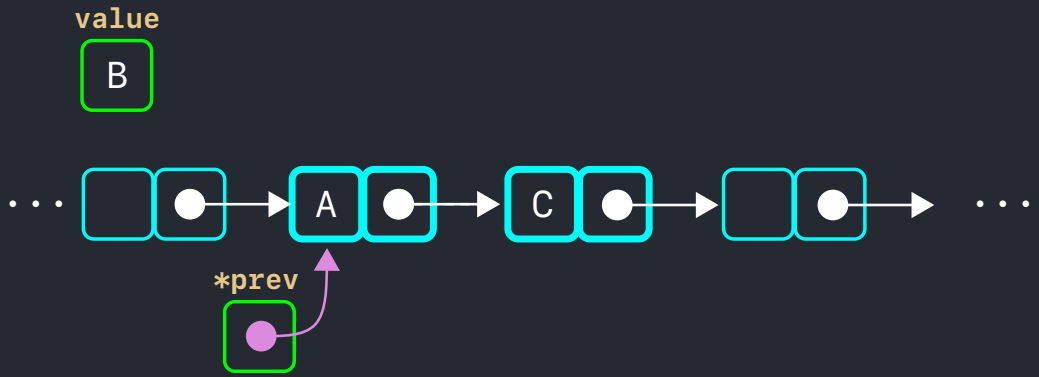
Pré e Pós Condição da Inserção

Veremos a seguir o algoritmo de inserção em ação.

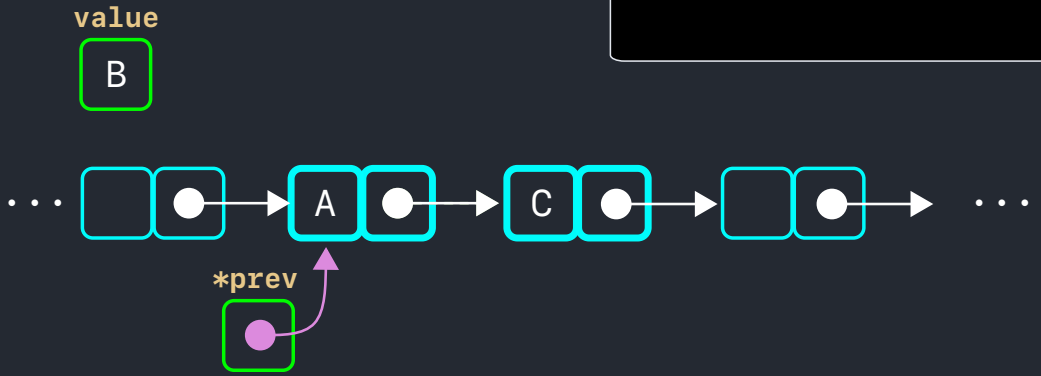
Pré e Pós Condição da Inserção

Veremos a seguir o algoritmo de inserção em ação.

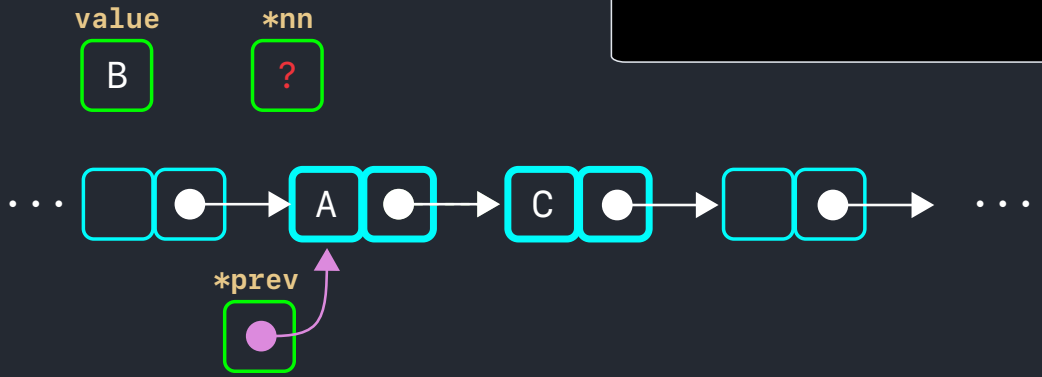
O ponteiro `*prev` aponta para o nó anterior ao local de inserção, e `value` contém o valor a ser inserido.


value
B
A
C
*prev

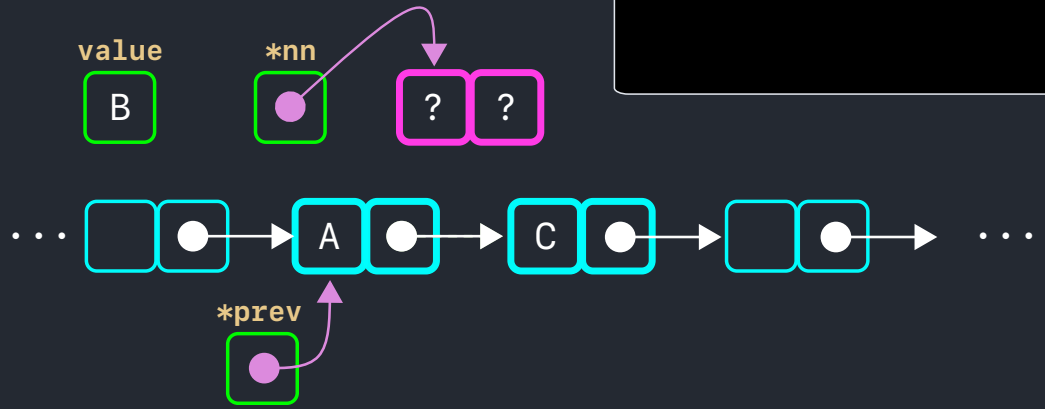
1. Alocar novo nó.



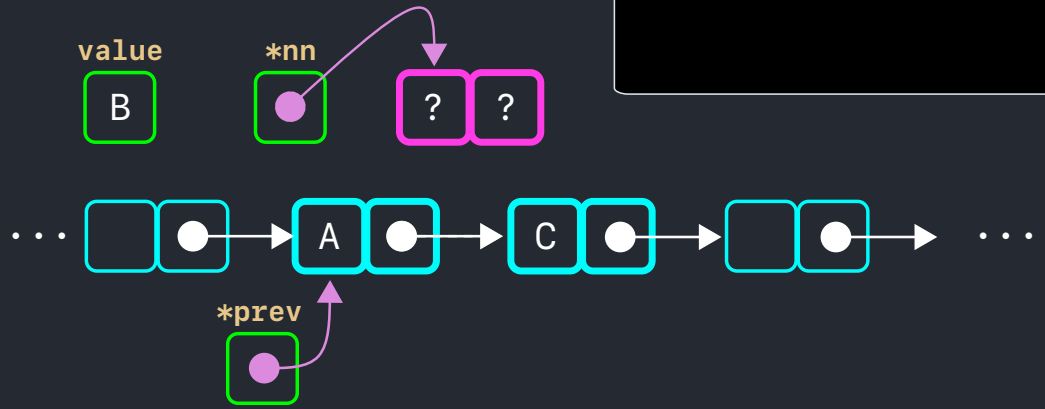
1. Alocar novo nó.



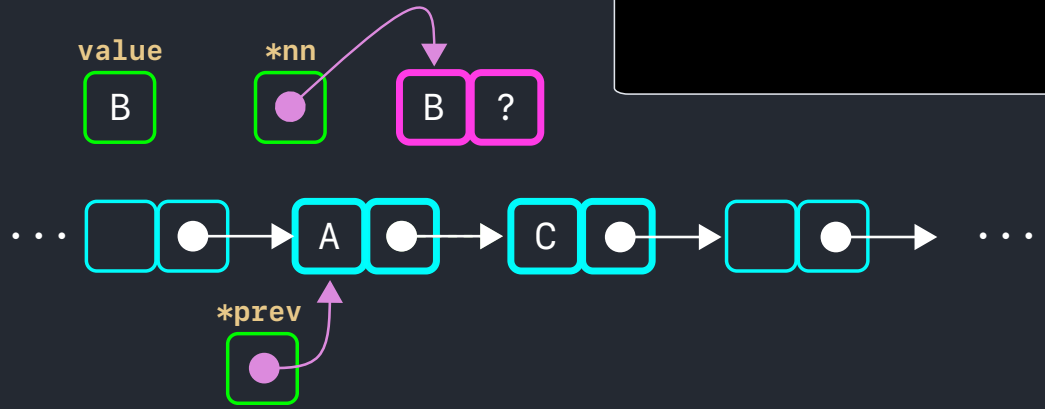
```
1. Node *nn = new Node;
```



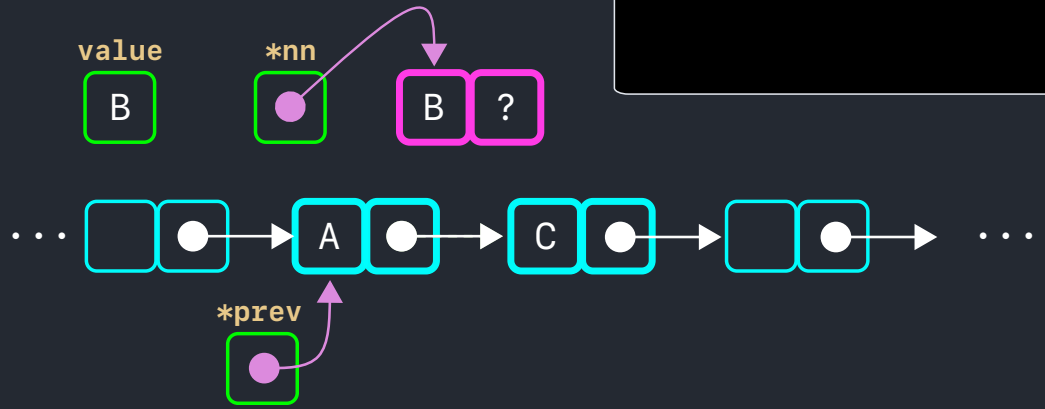
1. Node *nn = new Node;
2. Preencher novo nó.



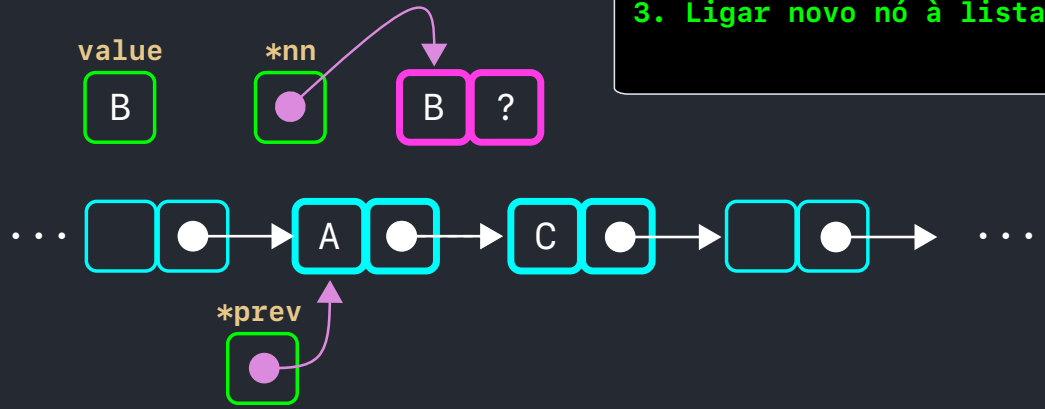
1. Node *nn = new Node;
2. Preencher novo nó.



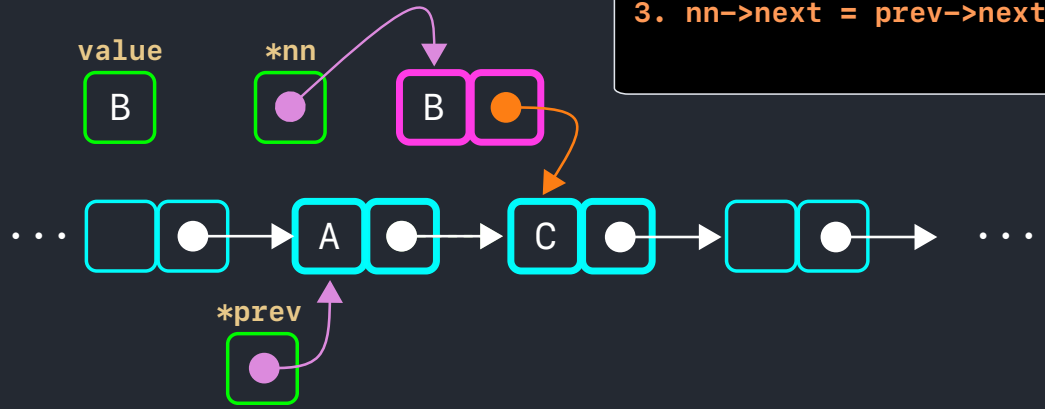
```
1. Node *nn = new Node;
2. nn->data = value;
```



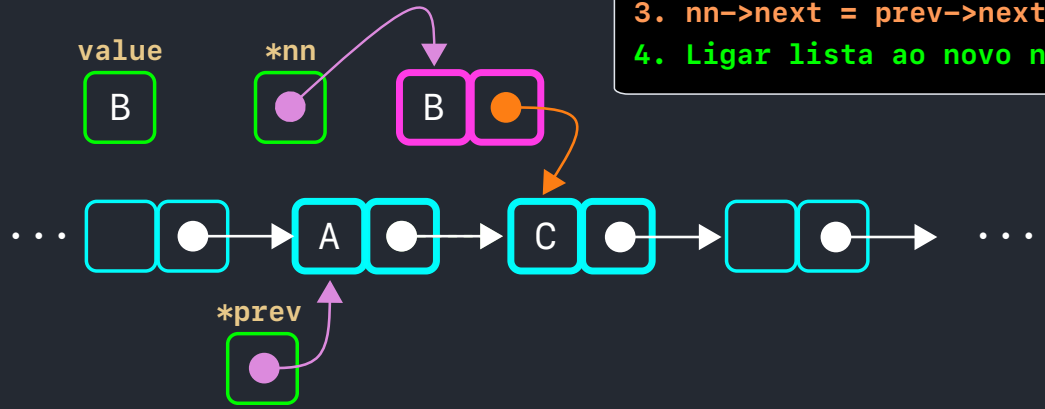
1. Node *nn = new Node;
2. nn->data = value;
3. Ligar novo nó à lista.



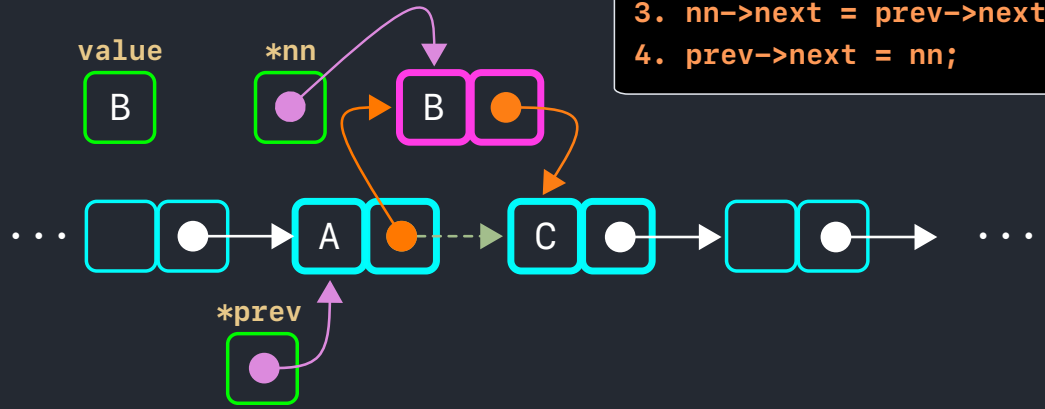
```
1. Node *nn = new Node;
2. nn->data = value;
3. nn->next = prev->next;
```



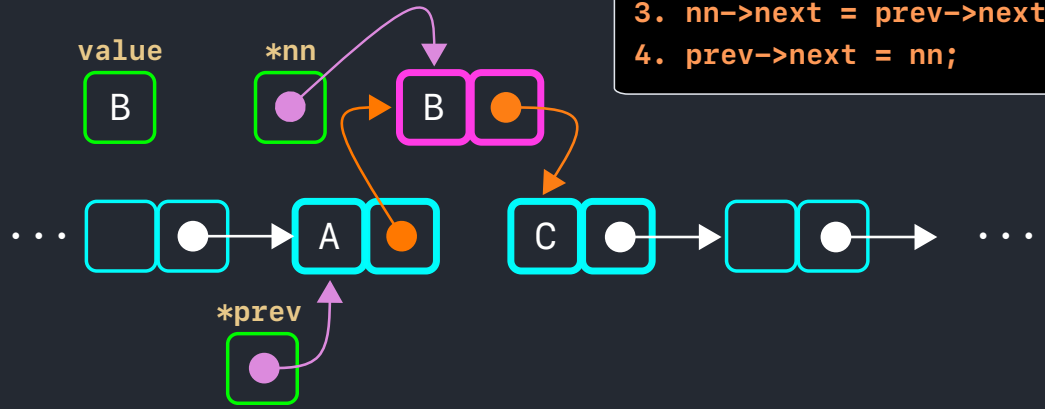
```
1. Node *nn = new Node;
2. nn->data = value;
3. nn->next = prev->next;
4. Ligar lista ao novo nó.
```



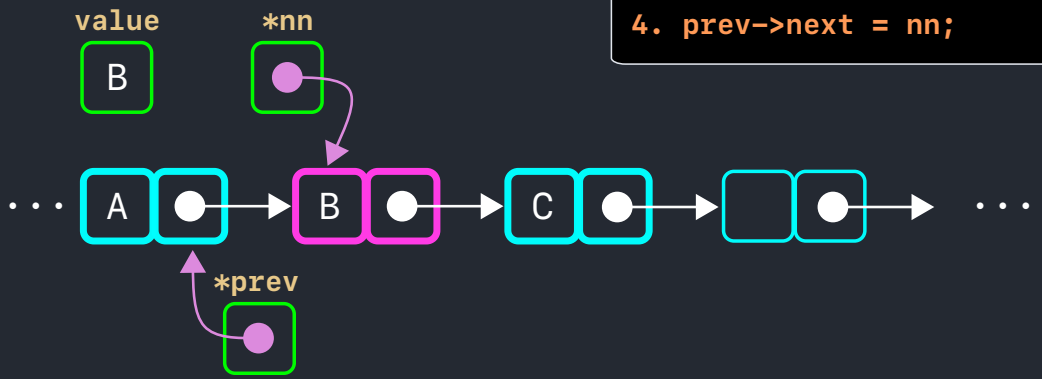
```
1. Node *nn = new Node;
2. nn->data = value;
3. nn->next = prev->next;
4. prev->next = nn;
```



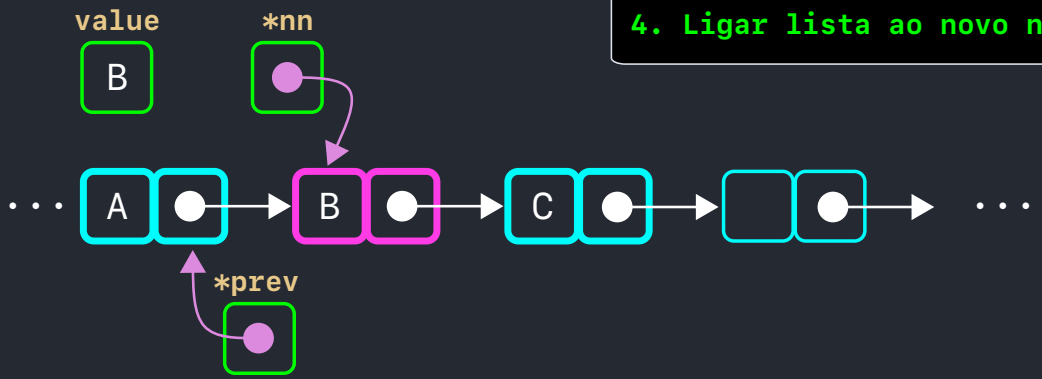
```
1. Node *nn = new Node;
2. nn->data = value;
3. nn->next = prev->next;
4. prev->next = nn;
```



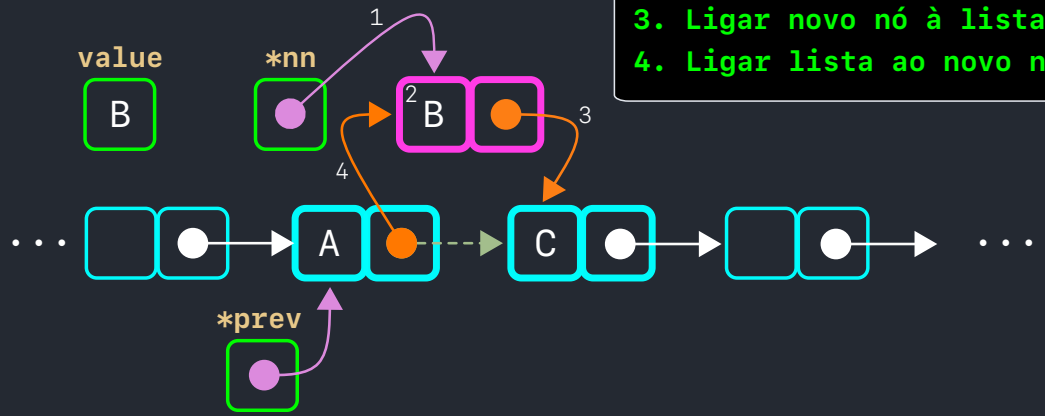
```
1. Node *nn = new Node;
2. nn->data = value;
3. nn->next = prev->next;
4. prev->next = nn;
```



1. Alocar novo nó.
2. Preencher novo nó.
3. Ligar novo nó à lista.
4. Ligar lista ao novo nó.



1. Alocar novo nó.
2. Preencher novo nó.
3. Ligar novo nó à lista.
4. Ligar lista ao novo nó.



Inserção em L.S.E.

Esta é a essência do processo de inserção.

Inserção em L.S.E.

Esta é a essência do processo de inserção.

Tudo que precisamos agora é escolher formas de obter o ponteiro `*prev`, que podem ser:

Inserção em L.S.E.

Esta é a essência do processo de inserção.

Tudo que precisamos agora é escolher formas de obter o ponteiro `*prev`, que podem ser:

- Indicando uma posição (índice) na lista.

Inserção em L.S.E.

Esta é a essência do processo de inserção.

Tudo que precisamos agora é escolher formas de obter o ponteiro `*prev`, que podem ser:

- Indicando uma posição (índice) na lista.
- Fazer uma busca por um elemento na lista *antes* do qual desejamos inserir o novo valor.

Inserção em L.S.E.

Em ambos os casos, a função deve retornar o ponteiro para o nó **anterior** ao item selecionado.

Inserção em L.S.E.

Em ambos os casos, a função deve retornar o ponteiro para o nó **anterior** ao item selecionado.

Precisamos também ter cuidado com situações de inserção onde não existe `*prev`.

Inserção em L.S.E.

Em ambos os casos, a função deve retornar o ponteiro para o nó **anterior** ao item selecionado.

Precisamos também ter cuidado com situações de inserção onde não existe `*prev`. Isto é, inserção no início da lista.

Exemplo #01: Inserção com `prev`

```
/// Insere `value` após nó apontado por `prev`
void insert(Node* &L, Node* prev, int value) {
    Node *nn = new Node(value);
    if(prev == nullptr){ // Inserção na frente da lista.
        nn->next = L;    // Funciona para lista vazia.
        L = nn;          // `nn` é a nova frente da lista.
    } else {             // Inserção com `prev` presente.
        nn->next = prev->next; // `nn` se liga a lista.
        prev->next = nn;       // Lista se liga a `nn`.
    }
}
```

Exemplo #02: Inserção “Indexada”

```
// Retorna ptr p/ nó anterior a `idx`.
// Se lista vazia retorna nullptr.
// Se `idx` estiver fora da lista, retorna ptr último nó.
Node* before_idx(Node* L, size_t idx){
    if(idx == 0) return nullptr;
    if(idx >= size(L)) return get_last_node(L);
    // Caminhar na lista.
    for(size_t step{0}; step<(idx-1) ; ++step)
        L = L->next;
    return L;
}
```

Exemplo #02: Inserção “Indexada”

```
void insert_at(Node* &L, size_t idx, int value) {
    // Ptr anterior ao índice desejado.
    auto *prev = before_idx(idx);
    Node *nn = new Node(value); // Alocar e preencher nó.
    if(prev == nullptr) {       // Inserção na frente.
        nn->next = L;           // Funciona p/ lista vazia.
        L = nn;                 // nn é nova frente lista.
    } else {                    // Inserção depois `prev`.
        nn->next = prev->next;
        prev->next = nn;
    }
}
```

Outras Operações sobre Lista

Nos próximos slides vamos abordar

- Remoção de elementos da lista;

Outras Operações sobre Lista

Nos próximos slides vamos abordar

- Remoção de elementos da lista;
- Variações de lista, como nó cabeça e lista duplamente encadeada.

Referências

Nick Parlante.
Pointers and Memory, Document #102.
Computer Science Education Library, Stanford University.
`http://cslibrary.stanford.edu/102`

Nick Parlante.
Linked List Basics, Document #103.
Computer Science Education Library, Stanford University.
`http://cslibrary.stanford.edu/103`