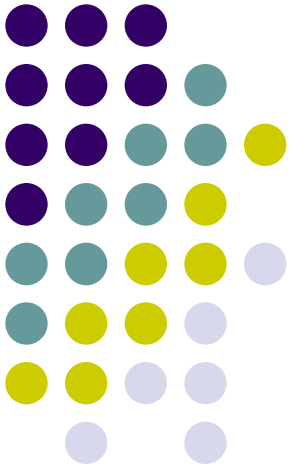
Tecnologia em Análise e Desenvolvimento de Sistemas

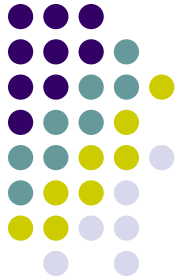
Estruturas de Dados

Prof. Igor Calebe Zadi



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO

3. Lista duplamente encadeada

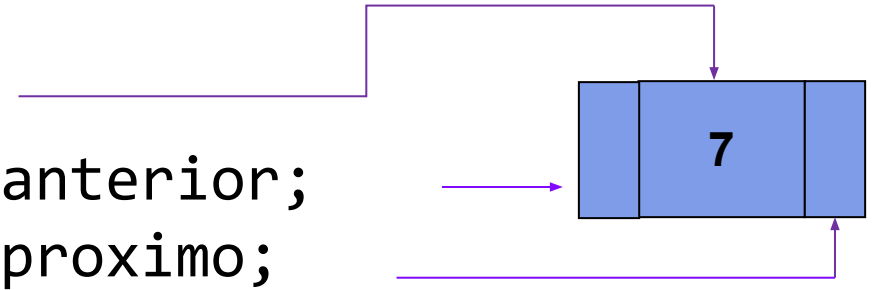


- Cada elemento armazena dados e dois ponteiros
  - Próximo e anterior
- Operações:
  - Inserção (início ou fim)
  - Consultar toda a lista (do início ao fim ou do fim ao início)
  - Remover (início ou fim)
  - Anterior e próximo (retorna um endereço ou NULL)

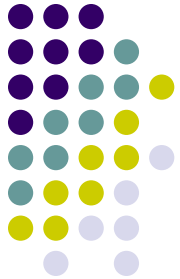
3. Listas encadeadas

- Estrutura da lista:
  - Cada nó normalmente é um registro (*struct*) auto referenciado
  - *Struct* auto referenciada, contém um ponteiro para um *struct* do mesmo tipo

```
struct Noh{

    int dados;
    struct Noh * anterior;
    struct Noh * proximo;

}
```



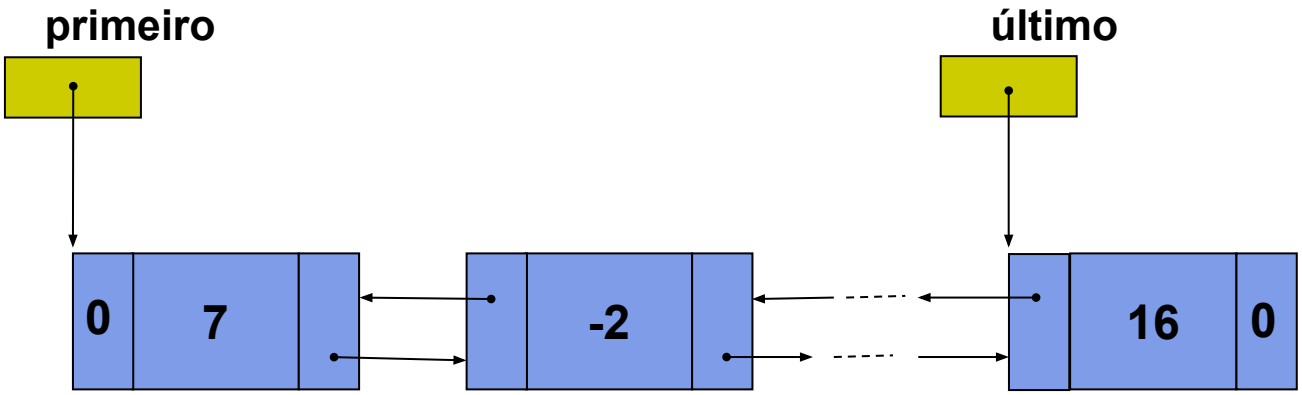
3. Listas encadeadas



- Lista duplamente encadeada
  - Regras:
    - Deve existir um ponteiro que aponte para o primeiro elemento da lista
    - Cada elemento, ou **nó**, da lista aponta para o próximo e para o anterior sucessivamente
    - O último elemento deve apontar para NULL, indicando o final da lista. No primeiro elemento o ponteiro anterior deve apontar para NULL.

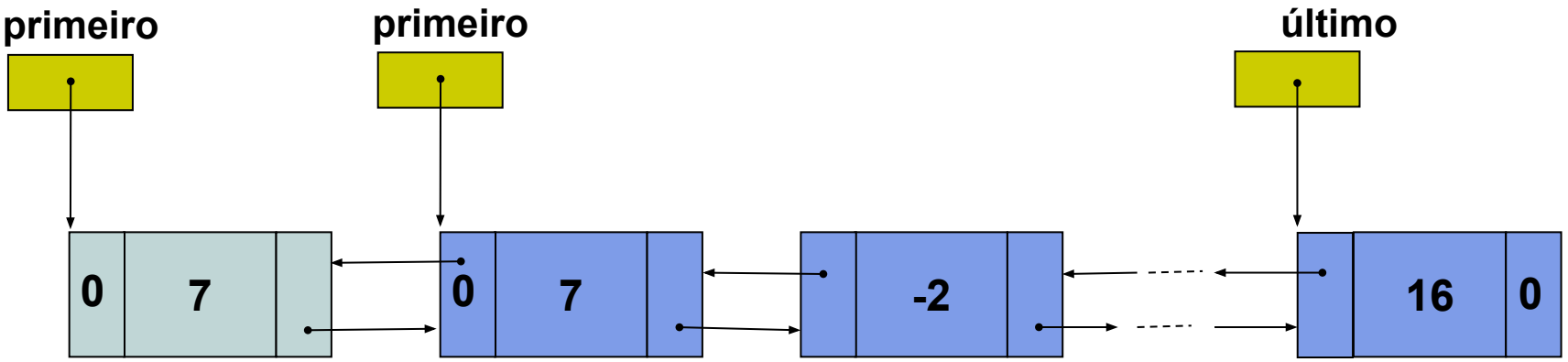
3. Lista duplamente encadeada

- Exemplo de lista duplamente encadeada

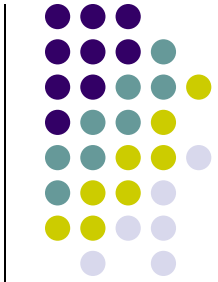


3. Listas encadeadas

- Inclusão - início



3. Listas encadeadas

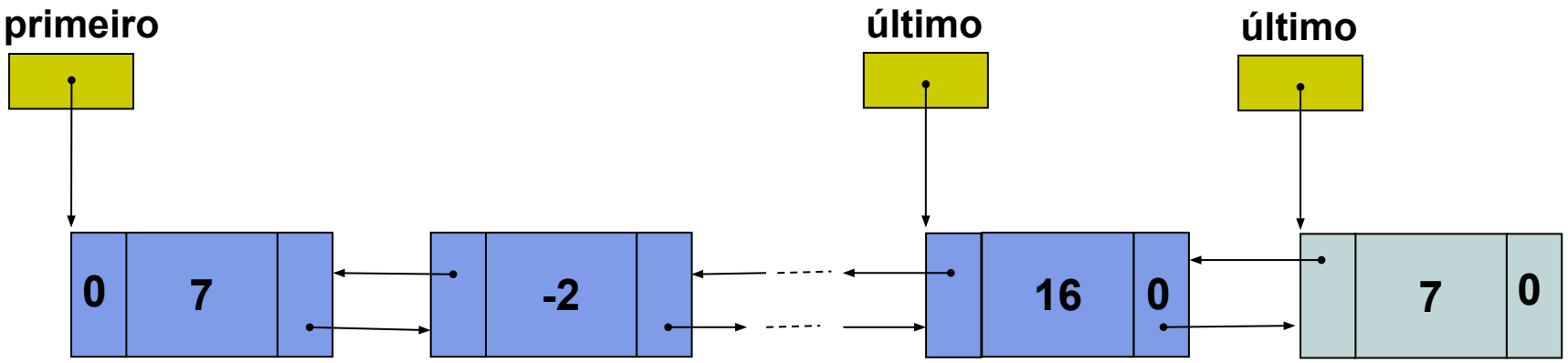


- Inserir no início

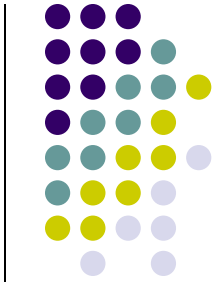
```
void incluir_inicio(Noh **primeiro, Noh **ultimo, int valor) {
   Noh *novo = (Noh *)malloc(sizeof(Noh));
   if (novo == NULL) {
       printf("Erro de alocação\n");return;
   }
   novo->dado = valor;
   novo->anterior = NULL;
   novo->proximo = *primeiro;
   if (*primeiro == NULL) {
       *ultimo = novo;
   } else {
       (*primeiro)->anterior = novo;
   }
   *primeiro = novo;
}
```

3. Listas encadeadas

- Inclusão - final



3. Listas encadeadas

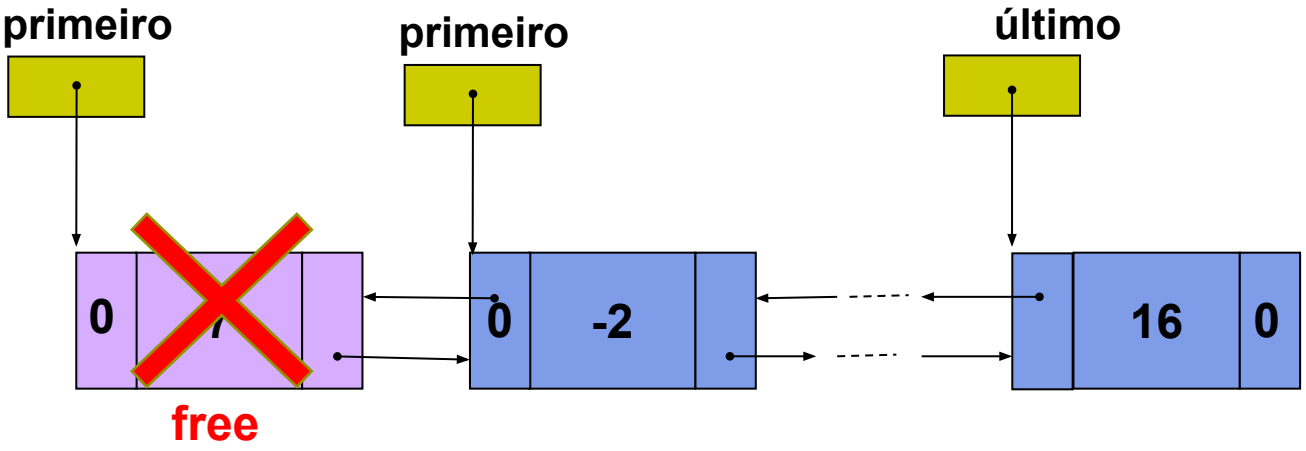


- Inserir no final

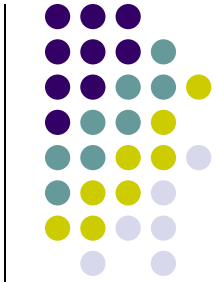
```
void incluir_fim(Noh **primeiro, Noh **ultimo, int valor) {
   Noh *novo = (Noh *)malloc(sizeof(Noh));
   if (novo == NULL) {
        printf("Erro de alocação\n");return;
   }
   novo->dado = valor;
   novo->proximo = NULL;
   novo->anterior = *ultimo;

   if (*ultimo == NULL) {
        *primeiro = novo;
   } else {
        (*ultimo)->proximo = novo;
   }
   *ultimo = novo;
}
```

3. Listas encadeadas

- Remover - início



3. Listas encadeadas

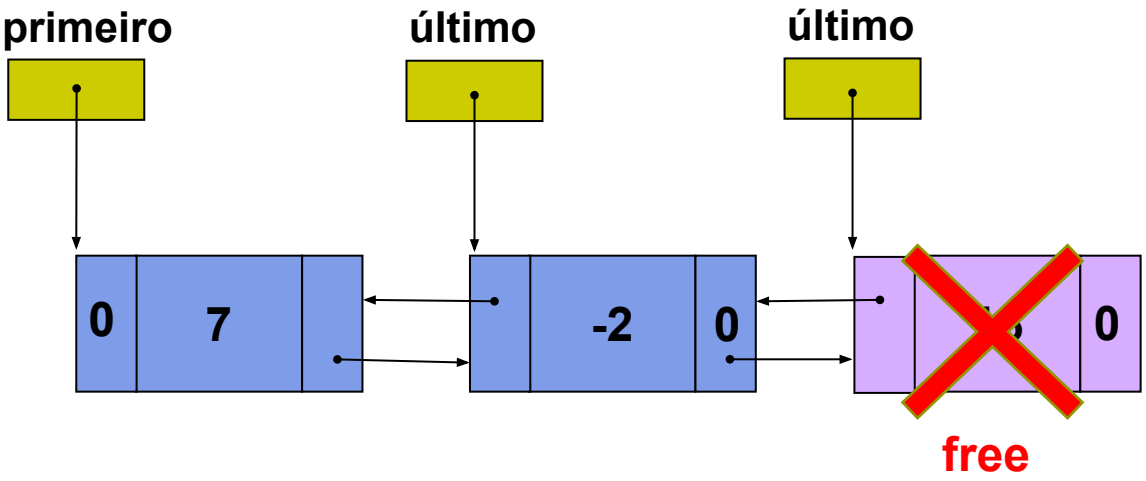


- Remover no início

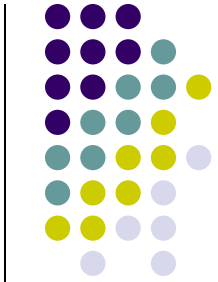
```
void remover_inicio(Noh **primeiro, Noh **ultimo) {
    if (*primeiro == NULL) {
        printf("Lista vazia\n");return;
    }
    Noh *temp = *primeiro;
    *primeiro = (*primeiro)->proximo;
    if (*primeiro != NULL) {
        (*primeiro)->anterior = NULL;
    } else {
        *ultimo = NULL;
    }
    free(temp);
}
```

3. Listas encadeadas

- Remover - final



3. Listas encadeadas

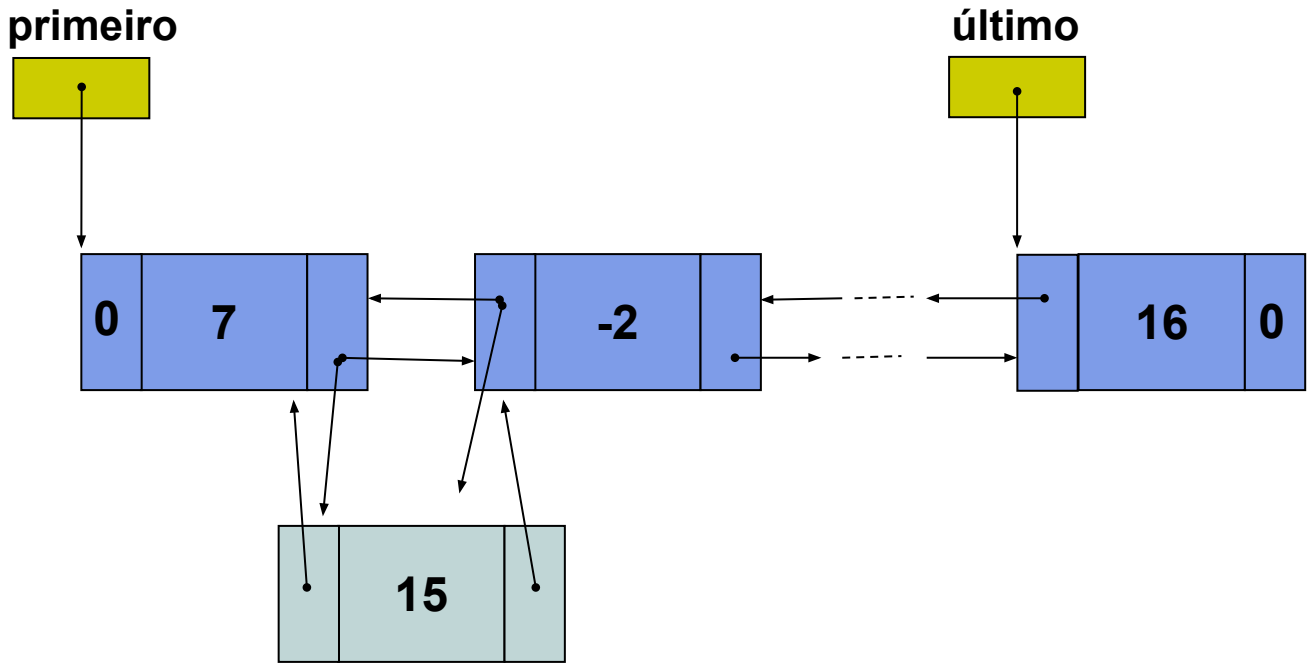


- Remover no final

```
void remover_fim(Noh **primeiro, Noh **ultimo) {
    if (*fim == NULL) {
        printf("Lista vazia\n"); return;
    }
    Noh *temp = *ultimo;
    *ultimo = (*ultimo)->anterior;
    if (*ultimo != NULL) {
        (*ultimo)->proximo = NULL;
    } else {
        *primeiro = NULL;
    }
    free(temp);
}
```

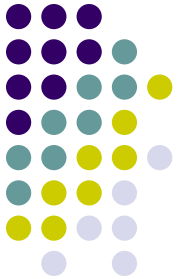
3. Listas encadeadas

- Inclusão - meio



3. Listas encadeadas

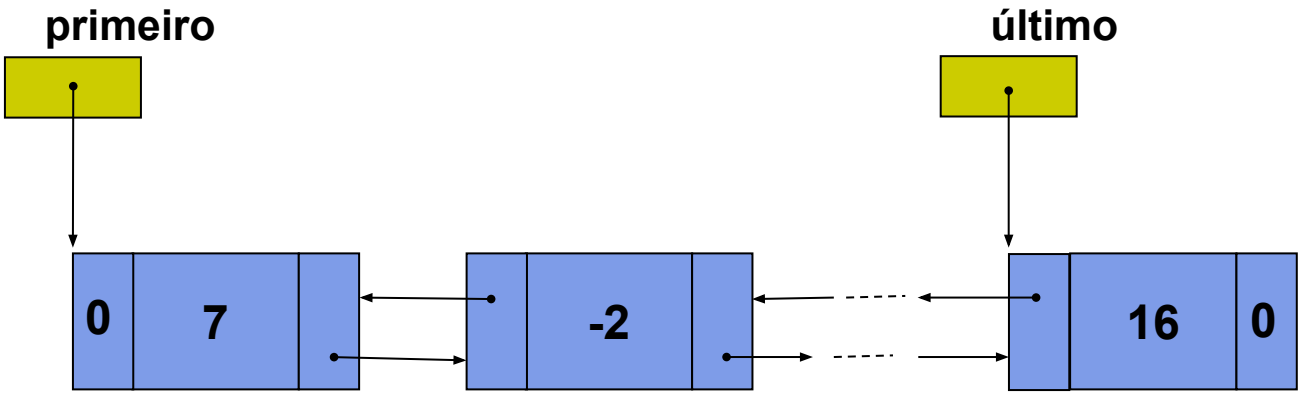
- Inclusão - meio



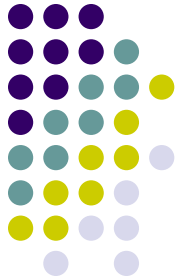
```
void incluir_meio(Noh **primeiro, Noh **ultimo,
int valor, int posicao) {
    Noh *atual = *primeiro;
    int i = 0;
    while (atual != NULL && i < posicao - 1) {
        atual = atual->proximo;
        i++;
    }
    if (atual == NULL) {
        printf("Fora da lista\n");
        return;
    }
    Noh *novo = (Noh *)malloc(sizeof(Noh));
    if (novo == NULL) {
        printf("Erro de alocação\n");
        return;
    }
    novo->dado = valor;
    novo->proximo = atual->proximo;
    novo->anterior = atual;

    if (atual->proximo != NULL) {
        atual->proximo->anterior = novo;
    } else {
        *ultimo = novo;
    }
    atual->proximo = novo;
}
```

3. Lista duplamente encadeada

- Percorrer a lista



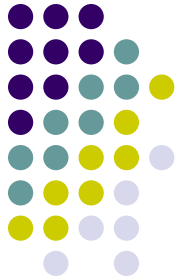
3. Listas encadeadas



- Percorrer

```
void percorrer(Noh *primeiro) {
    Noh *atual = primeiro;
    printf("Lista: ");
    while (atual != NULL) {
        printf("%d ", atual->dado);
        atual = atual->proximo;
    }
    printf("\n");
}
```

Observações sobre o material eletrônico



- O material ficará disponível na pasta compartilhada que é acessada sob convite
- O material foi elaborado a partir de diversas fontes (livros, internet, colegas, alunos etc.)
- Alguns trechos podem ter sido inteiramente transcritos a partir dessas fontes
- Outros trechos são de autoria própria
- Esta observação deve estar presente em qualquer utilização do material fora do ambiente de aulas do IFSP - Catanduva