Anhanguera

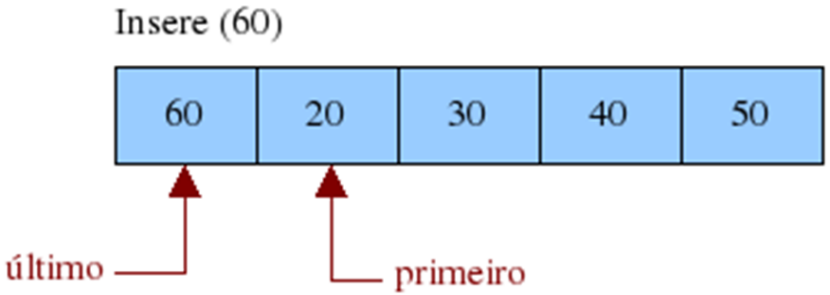
Aqui o seu esforço ganha força.

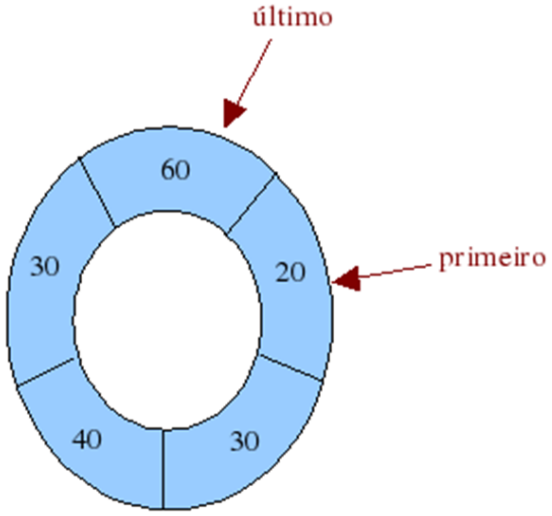
Listas Duplamente Encadeadas

Prof. Esp. Rodrigo Hentz

Definição

- Uma lista encadeada circular tem mais vantagens sobre uma lista encadeada linear, porém ainda apresenta algumas deficiências.
- Não podemos atravessar uma lista deste tipo no sentido contrário ou ainda para inserir ou adicionar um novo nó temos de ter um ponteiro para seu sucessor.



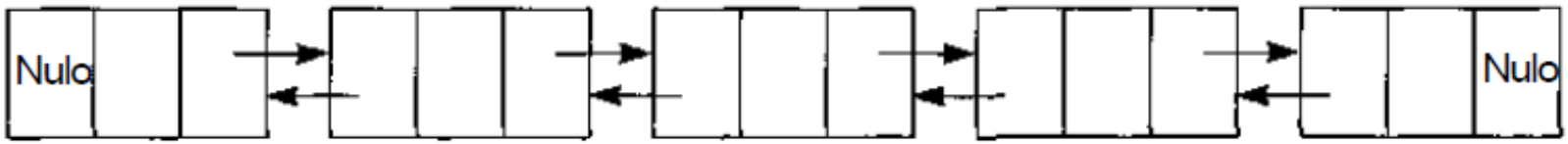


Definição

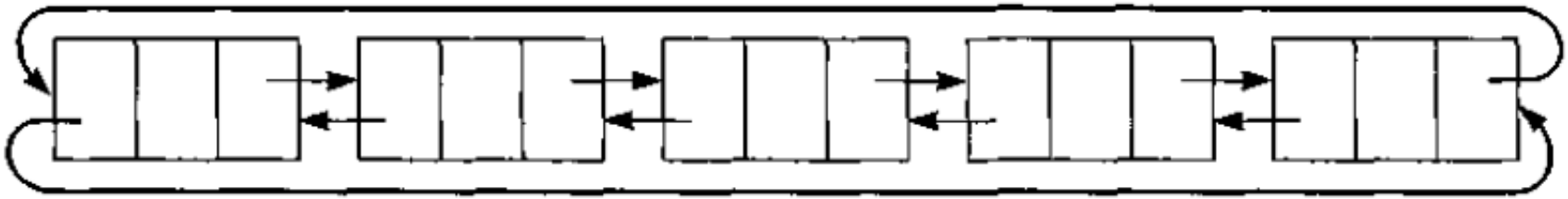
- Caso estes recursos sejam necessários, a estrutura de dados adequada é uma lista duplamente ligada.
- Cada nó em uma lista deste tipo contém dois ponteiros, um para seu predecessor e um para seu sucessor.
- As listas duplamente ligadas podem ser lineares ou circulares e podem ou não conter um nó de cabeçalho.

Listas Duplamente Encadeadas

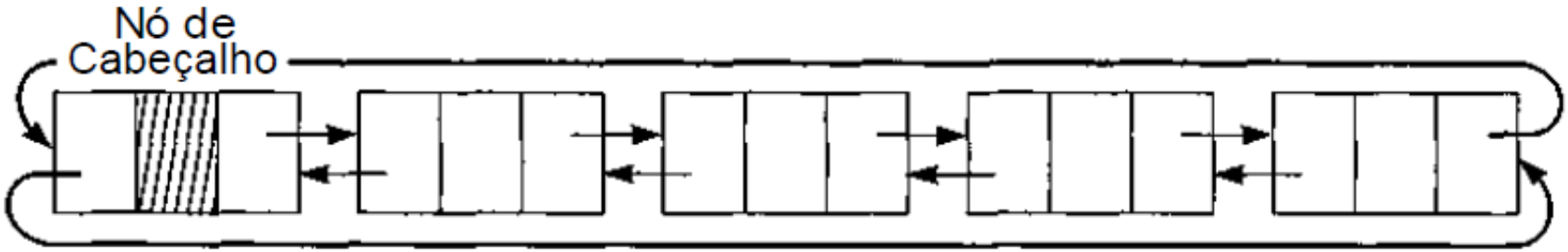
Definição



(a) Uma lista linear duplamente ligada.



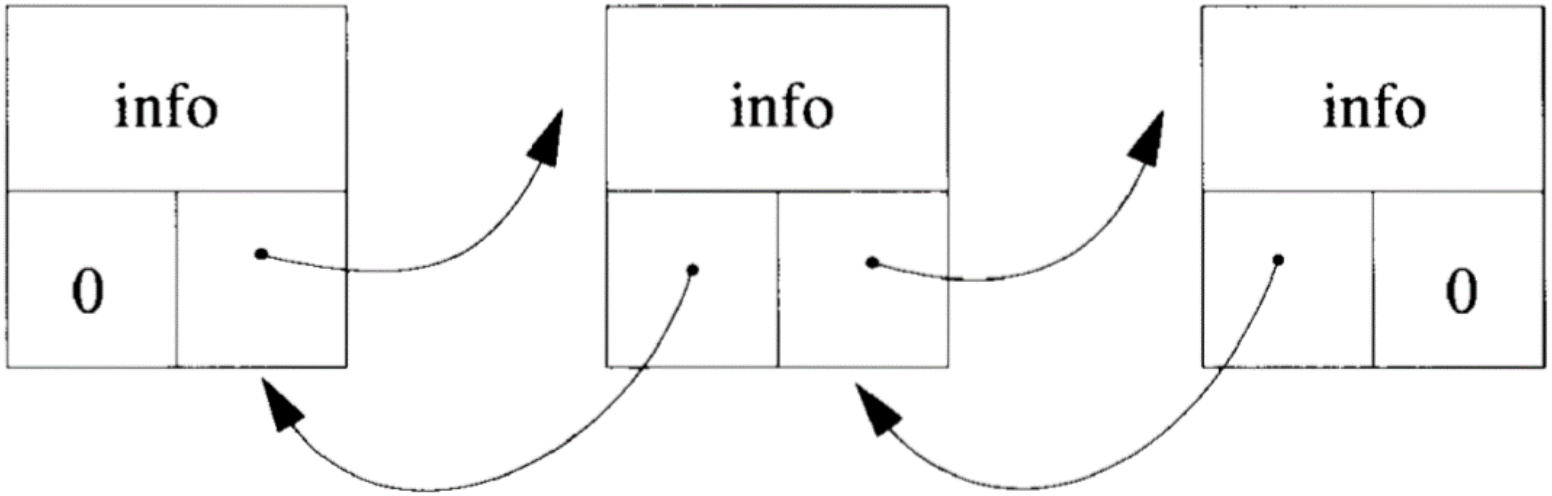
(b) Uma lista circular duplamente ligada sem cabeçalho.



(c) Uma lista circular duplamente ligada, com um cabeçalho.

Definição

- Podemos considerar os nós em uma lista duplamente ligada como consistindo em três campos: um atributo ***INF*** que contém as informações armazenadas no nó e os campos ***anterior*** e ***próximo***, que contém ponteiros para os nós em ambos os lados (esquerdo e direito).



Definição

- Podemos declarar um conjunto de nós deste tipo usando a implementação em vetor ou dinâmica:

Implementacão em Vetor

```
struct nodetype  {
   int info;
   int anterior, proximo;
};
struct nodetype node[NUMNODES];
```

Implementacao Dinamica

```
struct node {
   int info;
struct node *anterior, *proximo;
};
typedef struct node *NODEPTR;
```

Rotinas

- Apresentaremos agora rotinas que operam sobre listas duplamente encadeadas.
- Uma propriedade conveniente dessas listas é que, se p for um ponteiro para um nó qualquer, permitindo que anterior(p) seja uma abreviação para no[p].anterior ou p->anterior, e proximo(p) uma abreviação para no[p].proximo ou p->proximo, teremos:

$$anterior(proximo(p)) = p = proximo(anterior(p))$$

Rotinas

- Apresentaremos agora rotinas que operam sobre listas duplamente encadeadas.
- Uma propriedade conveniente dessas listas é que, se p for um ponteiro para um nó qualquer, permitindo que anterior(p) seja uma abreviação para no[p].anterior ou p->anterior, e proximo(p) uma abreviação para no[p].proximo ou p->proximo, teremos:

$$anterior(proximo(p)) = p = proximo(anterior(p))$$

Rotinas

- Funções utilizadas em uma lista duplamente encadeada:
  - Criação da lista encadeada e inicialização
  - Criação do nó dinamicamente
  - Inserir nó no inicio
  - Inserir nó no final
  - Inserir nó ordenado
  - Remover nó
  - Pesquisar nó
  - Imprimir nós

Anhanguera

Criação da Estrutura

```
typedef struct no {
   int info;
   struct no* anterior;
   struct no* proximo;
} sLista, sNo;
```

Criação da lista

- O ponteiro inicial da lista é definido e atribuído a nulo.

```
sLista* inicializaLista()
{
    printf("\nLista criada.");
    return NULL;
}
```

Criação da lista

- O ponteiro inicial da lista é definido e atribuído a nulo.

```
sLista* inicializaLista()
{
    printf("\nLista criada.");
    return NULL;
}
```

```c
int main(int argc, char** argv) {
    sLista* lista;
    int opcao;
    do
    {
        printf("\n");
        printf("1 - Iniciar lista\n");
        printf("0 - SAIR\n");
        printf("\nEntre com a opcao: ");
        scanf("%d", &opcao);
        switch (opcao)
        {
            case 1:
                lista = inicializaLista();
                break;
        }
        fflush(stdin);
    } while (opcao != 0);
    return 0;
}
```

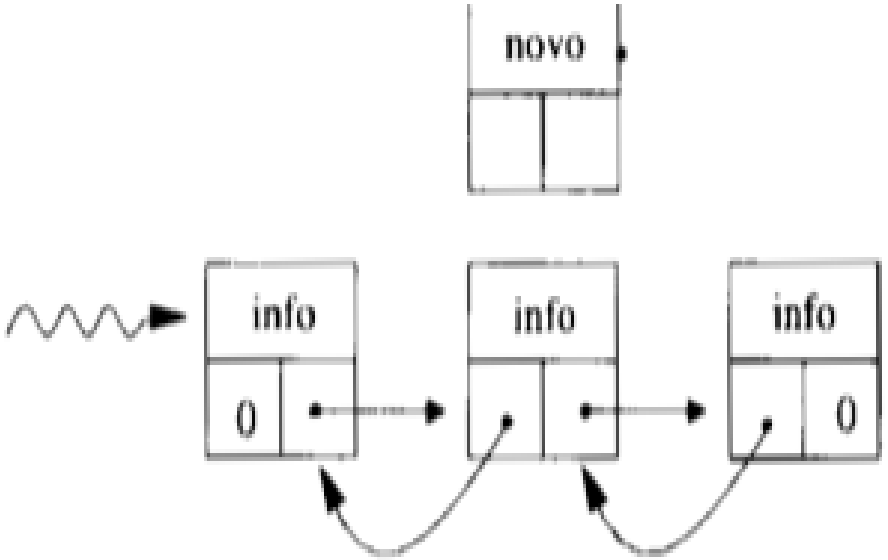
Criação de um nó dinamicamente

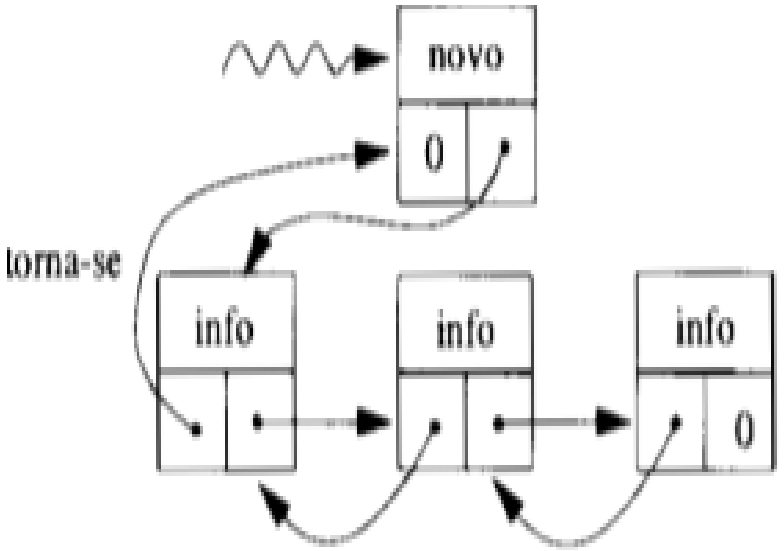
- Um novo nó é criado utilizando as funções de alocação dinâmica e seu ponteiro retornado com base no endereço de memória onde foi criado.

```
sNo* criarNo(int valor, sNo* anterior, sNo* proximo)
{
  sNo* p = (sNo*)malloc(sizeof(sNo));
  p->info = valor;
  p->anterior = anterior;
  p->proximo = proximo;
  return p;
}
```

Inserir nó no início

- Um novo nó é inserido no início da lista passando a ser o cabeçalho da lista. Caso já existir um nó ele deve ser amarrado ao novo.





Anhanguera

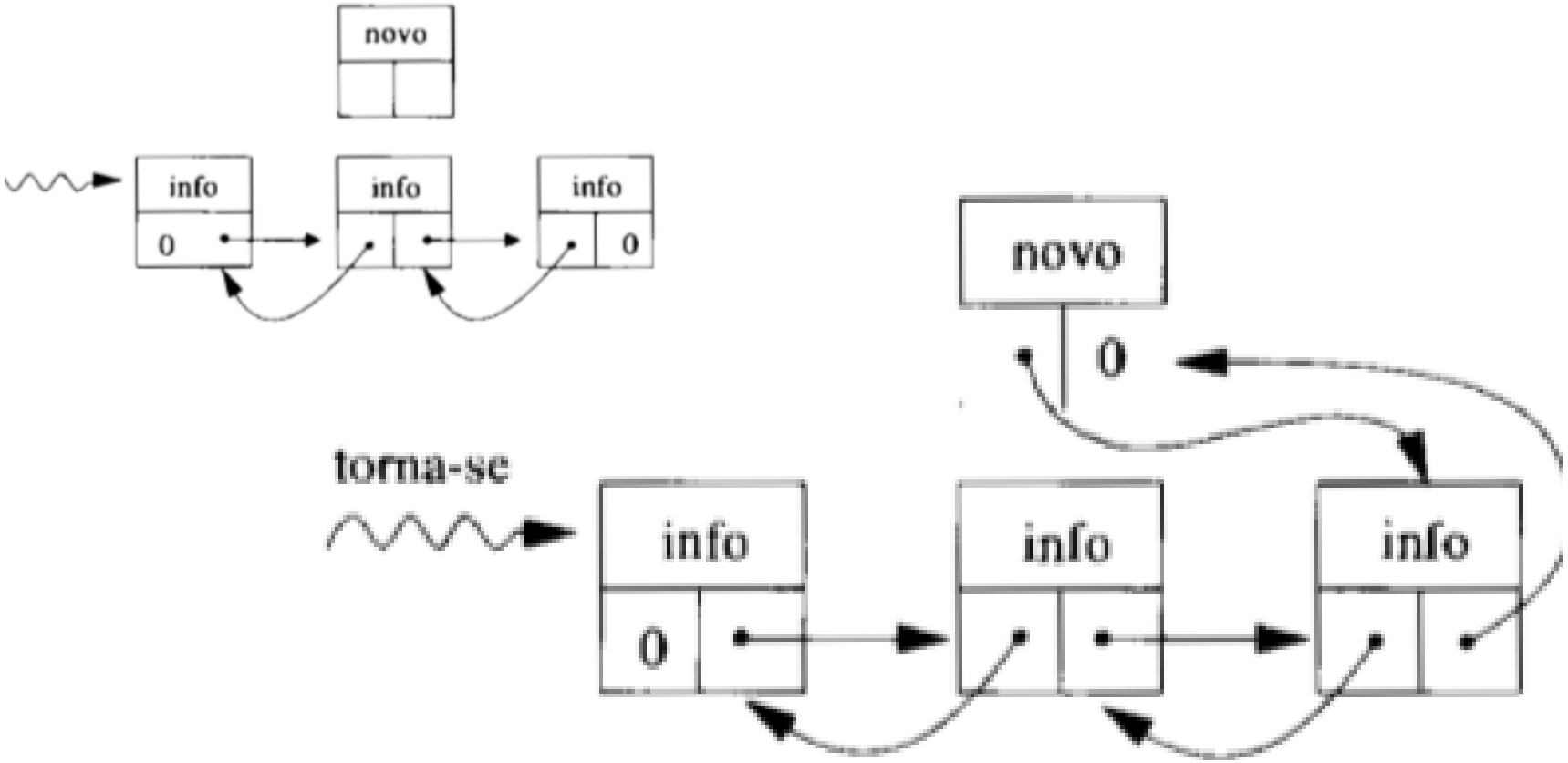
Inserir nó no início e chamada do menu

```
sLista* inserirInicio(sNo* no, int valor)
{
  sNo* novo = criarNo(valor, NULL, no);
  if (no != NULL) no->anterior = novo;
  return novo;
}
```

```
case 2:
    printf ("\nEntre com o numero para o novo no: ");
    scanf ("%d", &num);
    lista = inserirInicio(lista, num);
    break;
```

Inserir nó no final

- Um novo nó é inserido no final da lista.



Anhanguera

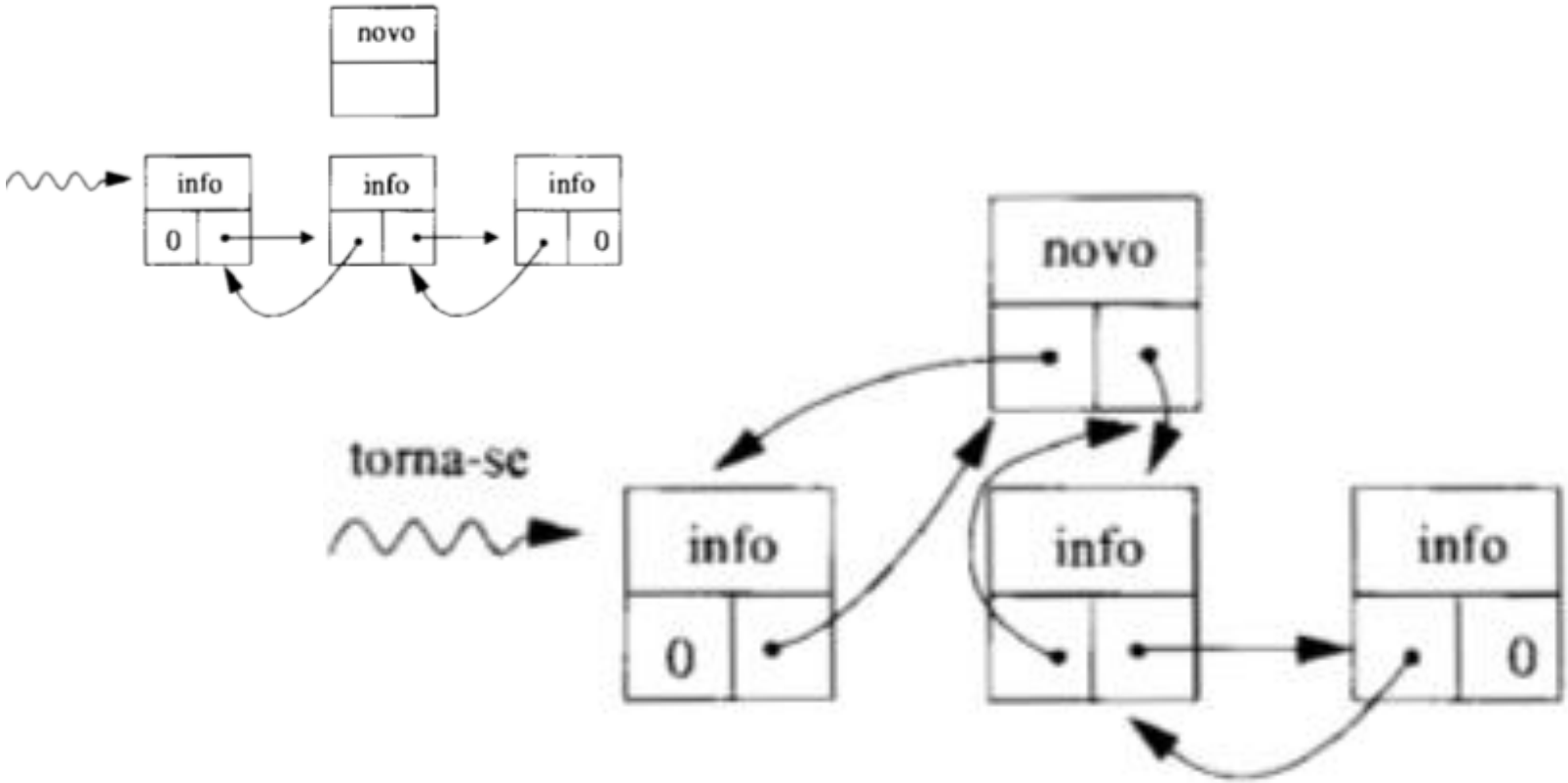
Inserir nó no final e chamada do menu

```
sLista* inserirFinal(sLista* lista, int valor)
{
  sLista* p = lista;
  if (lista == NULL) return inserirInicio(lista, valor);
  else
  {
    while(p->proximo != NULL) p = p->proximo;
    p->proximo = criarNo(valor, p, NULL);
    return lista;
  }
}
```

```
case 3:
    printf ("\nEntre com o numero para o novo no: ");
    scanf ("%d", &num);
    lista = inserirFinal(lista, num);
    break;
```

Inserir nó ordenado

- Um novo nó é inserido na posição baseado no índice.



Listas Duplamente Encadeadas

Anhanguera

```
sLista* inserirOrdenado(sLista* lista, int valor)
{
    sNo* p = lista;
    if (p == NULL) lista = inserirInicio(lista, valor);
    else
    {
      while (p != NULL && p->info < valor) p = p->proximo;
      if (p == NULL) lista = inserirFinal(lista, valor);
      else
      {
        if (p->anterior == NULL) lista = inserirInicio(lista, valor);
        else {
            sNo* novo = criarNo(valor, p->anterior, p->anterior->proximo);
            p->anterior->proximo = novo;
            p->anterior = novo;
        }
      }
    }
  return lista;
}
```

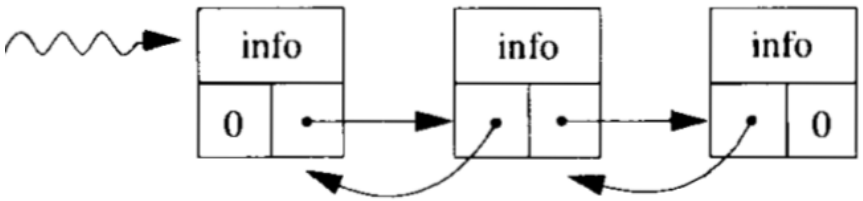
Listas Duplamente Encadeadas

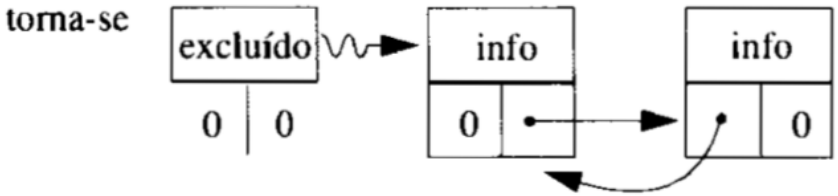
```
case 4:
    printf ("\nEntre com o numero para o novo no: ");
    scanf ("%d", &num);
    lista = inserirOrdenado(lista, num);
    break;
```

Excluir um nó

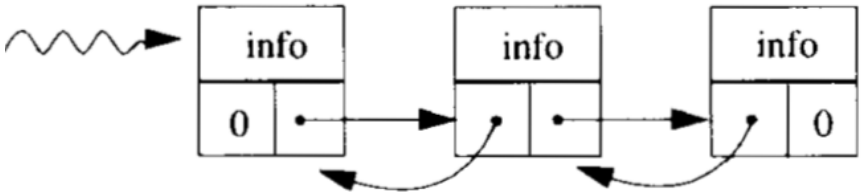
- O nó é retirado da lista e sua alocação é liberada. Os ponteiros anterior e próximo são alimentados nos outros nós.

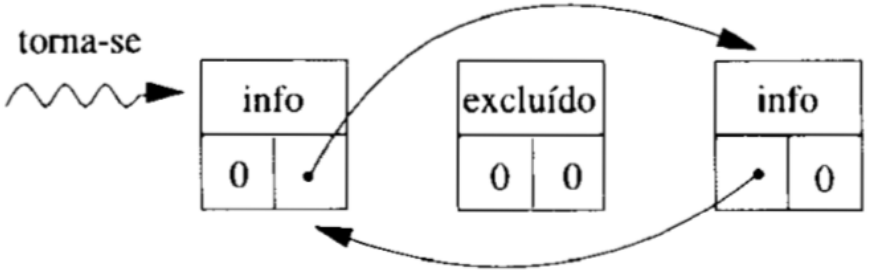
Apagando o primeiro item



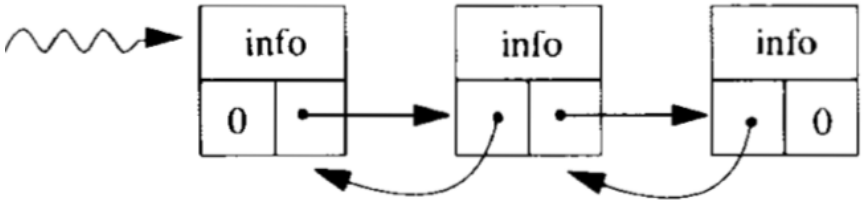


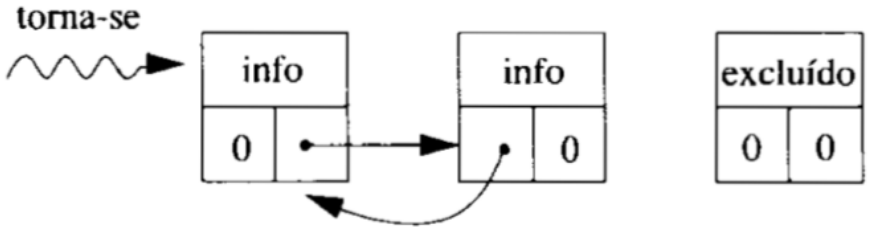
Apagando o item do meio





Apagando o último item





Listas Duplamente Encadeadas

```
sLista* excluirNo(sLista* lista, int valor)
{
    sNo* p = lista;
    while (p != NULL && p->info != valor) p = p->proximo;
    if (p == NULL) printf("\nValor nao encontrado para excluir.");
    else
    {
        if(p->anterior == NULL) {
            p->proximo->anterior = NULL;
            lista = p->proximo;
        }
```

Listas Duplamente Encadeadas

```
        else
        {
            if(p->proximo == NULL) p->anterior->proximo = NULL;
            else
            {
                p->anterior->proximo = p->proximo;
                p->proximo->anterior = p->anterior;
            }
        }
        printf("\nValor excluido.");
        free(p);
    }
    return lista;
}
```

```
case 5:
    printf("\nEntre com o numero para excluir: ");
    scanf("%d", num);
    lista = excluirNo(lista, num);
    break;
```

Anhanguera

Aqui o seu esforço ganha força.