Estrutura de Dados I

Luciana Lee

Tópicos da Aula

1. Lista Simplesmente Encadeada

   - Funções de Inserção e Remoção em Listas Encadeadas que Retornam *void*
   - Nó Sentinela
   - Nó Cabeça
   - Nó Base

Lista Simplesmente Encadeada

- Vimos até a última aula:
  - Estrutura de um nó de uma Lista Simplesmente Encadeada
  - Algoritmos de busca, inserção e remoção em listas não ordenadas
  - Algoritmos de busca, inserção e remoção em listas ordenadas

Lista Simplesmente Encadeada

- Nas funções de operações de inserção e exclusão em listas simplesmente encadeadas implementadas até o momento, sempre retornamos o início da lista.
- Podemos trabalhar com a referência para o ponteiro da lista.
- Desta forma, o retorno das funções podem ser `void`.

Lista Simplesmente Encadeada

```
1  struct no *insereListaOrd (struct no *L, int valor) {
2    struct no *novo = criaNo(valor);
3    struct no *pred = NULL;
4    struct no *aux = buscaListaOrd(L, valor, &pred);
5    novo->prox = aux;
6    if (pred == NULL) L = novo;
7    else pred->prox = novo;
8    return L;
9  }
```

Chamada da função:

```
Lista = insereListaOrd(Lista, chave);
```

Lista Simplesmente Encadeada

```
1  struct no *insereListaOrd (struct no *L, int valor) {
2    struct no *novo = criaNo(valor);
3    struct no *pred = NULL;
4    struct no *aux = buscaListaOrd(L, valor, &pred);
5    novo->prox = aux;
6    if (pred == NULL) L = novo;
7    else pred->prox = novo;
8    return L;
9  }
```

Chamada da função:

```
Lista = insereListaOrd(Lista, chave);
```

```
1  void insereListaOrd (struct no **L, int valor) {
2    struct no *novo = criaNo(valor);
3    struct no *pred = NULL;
4    struct no *aux = buscaListaOrd(*L, valor, &pred);
5    novo->prox = aux;
6    if (pred == NULL) *L = novo;
7    else pred->prox = novo;
8  }
```

Chamada da função:

```
insereListaOrd(&Lista, chave);
```

Lista Simplesmente Encadeada

Memória		
posição	variável	conteúdo
102	`struct no *V`	NULL
103		
104		
105		
106		
107		
108		
109		
110	`struct no **L`	
111	`int ch`	
112		
113		

```
void insereOrd (struct no **L, int ch) {
  struct no *novo = criaNo(ch);
  struct no *pred = NULL;
  struct no *aux = buscaListaOrd(*L, ch, &pred);
  novo->prox = aux;
  if (pred == NULL) *L = novo;
  pred->prox = novo;
}

int main () {
  struct no *V = NULL;

  insereOrd(&V, 11);
  insereOrd(&V, 13);

  return 0;
}
```

Lista Simplesmente Encadeada

Memória		
posição	variável	conteúdo
102	`struct no *V`	NULL
103		
104		
105		
106		
107		
108		
109		
110	`struct no **L`	102
111	`int ch`	11
112		
113		

```
void insereOrd (struct no **L, int ch) {
  struct no *novo = criaNo(ch);
  struct no *pred = NULL;
  struct no *aux = buscaListaOrd(*L, ch, &pred);
  novo->prox = aux;
  if (pred == NULL) *L = novo;
  pred->prox = novo;
}

int main () {
  struct no *V = NULL;

  insereOrd(&V, 11);
  insereOrd(&V, 13);

  return 0;
}
```

Lista Simplesmente Encadeada

Memória		
posição	variável	conteúdo
102	struct no *V	105
103		
104		
105	chave	11
106	prox	NULL
107		
108		
109		
110	struct no **L	102
111	int ch	13
112		
113		

```
void insereOrd (struct no **L, int ch) {
  struct no *novo = criaNo(ch);
  struct no *pred = NULL;
  struct no *aux = buscaListaOrd(*L, ch, &pred);
  novo->prox = aux;
  if (pred == NULL) *L = novo;
  pred->prox = novo;
}

int main () {
  struct no *V = NULL;

  insereOrd(&V, 11);
  insereOrd(&V, 13);

  return 0;
}
```

Lista Simplesmente Encadeada

Memória		
posição	variável	conteúdo
102	struct no *V	105
103		
104		
105	chave	11
106	prox	107
107	chave	13
108	prox	NULL
109		
110	struct no **L	
111	int ch	
112		
113		

```
void insereOrd (struct no **L, int ch) {
  struct no *novo = criaNo(ch);
  struct no *pred = NULL;
  struct no *aux = buscaListaOrd(*L, ch, &pred);
  novo->prox = aux;
  if (pred == NULL) *L = novo;
  pred->prox = novo;
}

int main () {
  struct no *V = NULL;

  insereOrd(&V, 11);
  insereOrd(&V, 13);

  return 0;
}
```

Nó Sentinela

- É um nó especial do mesmo tipo que os demais nós da lista.

Nó Sentinela

- É um nó especial do mesmo tipo que os demais nós da lista.
- Não é utilizado para guardar dados de um elemento da lista.

Nó Sentinela

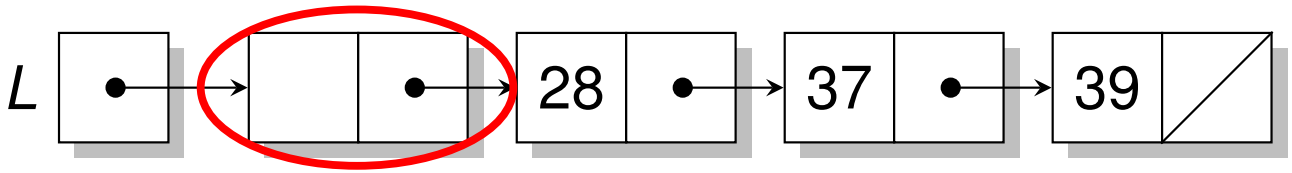
- É um nó especial do mesmo tipo que os demais nós da lista.
- Não é utilizado para guardar dados de um elemento da lista.
- Tal nó é utilizado apenas para facilitar operações de inserção/remoção.

Nó Sentinela

- É um nó especial do mesmo tipo que os demais nós da lista.
- Não é utilizado para guardar dados de um elemento da lista.
- Tal nó é utilizado apenas para facilitar operações de inserção/remoção.
- Pode ser utilizado no início de cada lista.

Nó Sentinela

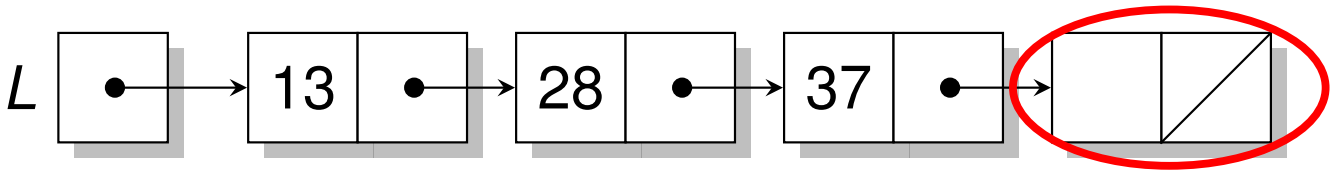
- É um nó especial do mesmo tipo que os demais nós da lista.
- Não é utilizado para guardar dados de um elemento da lista.
- Tal nó é utilizado apenas para facilitar operações de inserção/remoção.
- Pode ser utilizado no início de cada lista.
- Pode ser utilizado no final de cada lista.



nó sentinela no início

Nó Sentinela

- É um nó especial do mesmo tipo que os demais nós da lista.
- Não é utilizado para guardar dados de um elemento da lista.
- Tal nó é utilizado apenas para facilitar operações de inserção/remoção.
- Pode ser utilizado no início de cada lista.
- Pode ser utilizado no final de cada lista.



nó sentinela no final

Nó Cabeça

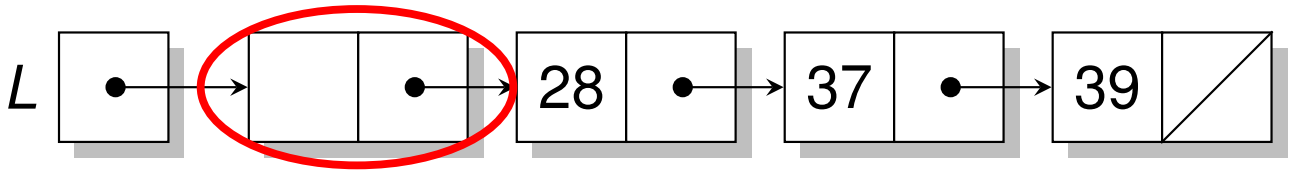
- Quando o nó sentinela é utilizado no início da lista, ele é chamado de **nó de cabeçalho**, ou **nó cabeça**, ou simplesmente de **cabeça**.

Nó Cabeça

- Quando o nó sentinela é utilizado no início da lista, ele é chamado de **nó de cabeçalho**, ou **nó cabeça**, ou simplesmente de **cabeça**.

- Frequentemente, a parte referente às informações deste nó podem ser utilizados para armazenar informações globais sobre a lista. Por exemplo, o número de nós da lista.

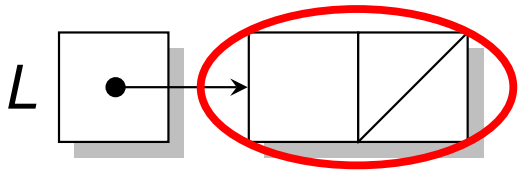
Nó Cabeça



nó cabeça

- Uma lista simplesmente encadeada com nó cabeça
- Uma lista vazia
- Nó cabeça guardando o número total de nós da lista (sem contar com o próprio nó cabeça)

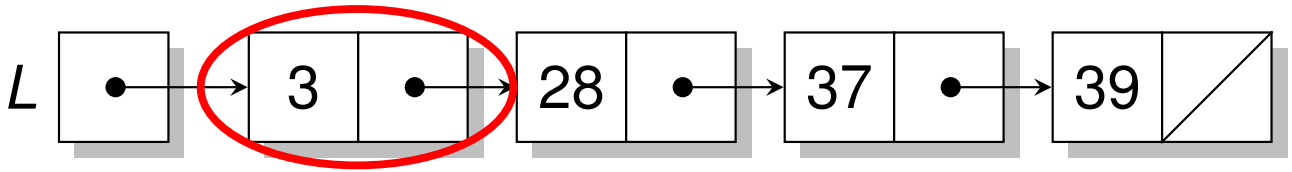
Nó Cabeça



nó cabeça

- Uma lista simplesmente encadeada com nó cabeça
- Uma lista vazia
- Nó cabeça guardando o número total de nós da lista (sem contar com o próprio nó cabeça)

Nó Cabeça



nó cabeça

- Uma lista simplesmente encadeada com nó cabeça
- Uma lista vazia
- Nó cabeça guardando o número total de nós da lista (sem contar com o próprio nó cabeça)

Nó Cabeça

- Como a implementação da função de inserção no início mudaria?

```
struct no *insereInicio(struct no *L, int valor){
  struct no *novo = criaNo(valor);
  novo->prox = L;
  L = novo;
  return L;
}
```

Chamada da função:

```
Lista = insereInicio(Lista, chave);
```

```
void insereInicio (struct no *L, int valor){
  struct no *novo = criaNo(valor);
  novo->prox = L->prox;
  L->prox = novo;
}
```

Chamada da função:

```
insereInicio(Lista,ch);
```

Nó Cabeça

- Como a implementação da função de inserção no início mudaria?

```
struct no *insereInicio(struct no *L, int valor){
  struct no *novo = criaNo(valor);
  novo->prox = L;
  L = novo;
  return L;
}
```

Chamada da função:

```
Lista = insereInicio(Lista, chave);
```

```
void insereInicio (struct no *L, int valor){
  struct no *novo = criaNo(valor);
  novo->prox = L->prox;
  L->prox = novo;
}
```

Chamada da função:

```
insereInicio(Lista,ch);
```

Nó Cabeça

- Como a implementação da função de inserção no início mudaria?

```
struct no *insereInicio(struct no *L, int valor){
  struct no *novo = criaNo(valor);
  novo->prox = L;
  L = novo;
  return L;
}
```

Chamada da função:

```
Lista = insereInicio(Lista, chave);
```

```
void insereInicio (struct no *L, int valor){
  struct no *novo = criaNo(valor);
  novo->prox = L->prox;
  L->prox = novo;
}
```

Chamada da função:

```
insereInicio(Lista,ch);
```

Nó Cabeça

- Como a implementação da função de inserção no final mudaria?

```
struct no *insereFinal(struct no *L, int valor){
  struct no *novo = criaNo(valor);
  struct no *aux = L;
  if (L == NULL) L = novo;
  else {
    while (aux->prox != NULL) aux = aux->prox;
    aux->prox = novo;
  }
  return L;
}
```

Chamada da função:

```
Lista = insereFinal(Lista, chave);
```

```
void insereFinal (struct no *L, int valor){
  struct no *novo = criaNo(valor);
  struct no *aux = L;
  while (aux->prox != NULL) aux = aux->prox;
  aux->prox = novo;
}
```

Chamada da função:

```
insereFinal(Lista,ch);
```

Nó Cabeça

- Como a implementação da função de inserção no final mudaria?

```
struct no *insereFinal(struct no *L, int valor){
  struct no *novo = criaNo(valor);
  struct no *aux = L;
  if (L == NULL) L = novo;
  else {
    while (aux->prox != NULL) aux = aux->prox;
    aux->prox = novo;
  }
  return L;
}
```

Chamada da função:

```
Lista = insereFinal(Lista, chave);
```

```
void insereFinal (struct no *L, int valor){
  struct no *novo = criaNo(valor);
  struct no *aux = L;
  while (aux->prox != NULL) aux = aux->prox;
  aux->prox = novo;
}
```

Chamada da função:

```
insereFinal(Lista,ch);
```

Nó Cabeça

- Como a implementação da função de inserção no final mudaria?

```
struct no *insereFinal(struct no *L, int valor){
  struct no *novo = criaNo(valor);
  struct no *aux = L;
  if (L == NULL) L = novo;
  else {
    while (aux->prox != NULL) aux = aux->prox;
    aux->prox = novo;
  }
  return L;
}
```

Chamada da função:

```
Lista = insereFinal(Lista, chave);
```

```
void insereFinal (struct no *L, int valor){
  struct no *novo = criaNo(valor);
  struct no *aux = L;
  while (aux->prox != NULL) aux = aux->prox;
  aux->prox = novo;
}
```

Chamada da função:

```
insereFinal(Lista,ch);
```

Nó Cabeça

- Como a implementação da função de exclusão no início mudaria?

```
struct no *excluiInicio (struct no *L) {
  struct no *aux = L;
  if (L == NULL) return NULL;
  else {
    L = L->prox;
    free(aux);
    return L;
  }
}
```

Chamada da função:

```
Lista = excluiInicio(Lista);
```

```
void excluiInicio(struct no *L){
  struct no *aux = L->prox;
  if (aux != NULL) {
    L->prox = aux->prox;
    free(aux);
  }
}
```

Chamada da função:

```
excluiInicio(Lista);
```

Nó Cabeça

- Como a implementação da função de exclusão no início mudaria?

```
struct no *excluiInicio (struct no *L) {
  struct no *aux = L;
  if (L == NULL) return NULL;
  else {
    L = L->prox;
    free(aux);
    return L;
  }
}
```

Chamada da função:

```
Lista = excluiInicio(Lista);
```

```
void excluiInicio(struct no *L){
  struct no *aux = L->prox;
  if (aux != NULL) {
    L->prox = aux->prox;
    free(aux);
  }
}
```

Chamada da função:

```
excluiInicio(Lista);
```

Nó Cabeça

- Como a implementação da função de exclusão no início mudaria?

```
struct no *excluiInicio (struct no *L) {
  struct no *aux = L;
  if (L == NULL) return NULL;
  else {
    L = L->prox;
    free(aux);
    return L;
  }
}
```

Chamada da função:

```
Lista = excluiInicio(Lista);
```

```
void excluiInicio(struct no *L){
  struct no *aux = L->prox;
  if (aux != NULL) {
    L->prox = aux->prox;
    free(aux);
  }
}
```

Chamada da função:

```
excluiInicio(Lista);
```

Nó Cabeça

- Como a implementação da função de exclusão no final mudaria?

```
struct no *excluiFinal(struct no *L){
  struct no *aux = L;
  struct no *pred = NULL;
  if (L == NULL) return NULL;
  else {
    while (aux->prox != NULL){
      pred = aux;
      aux = aux->prox;
    }
    if (pred == NULL) L = NULL;
    else pred->prox = NULL;
    free(aux);
    return L;
  }
}
```

Chamada da função:

```
Lista = excluiFinal(Lista);
```

```
void excluiFinal (struct no *L) {
  struct no *pred = L;
  struct no *aux = L->prox;
  if (aux != NULL){
    while (aux->prox != NULL) {
      pred = aux;
      aux = aux->prox;
    }
    pred->prox = NULL;
    free(aux);
  }
}
```

Chamada da função:

```
excluiFinal(Lista);
```

Nó Cabeça

- Como a implementação da função de exclusão no final mudaria?

```
struct no *excluiFinal(struct no *L){
  struct no *aux = L;
  struct no *pred = NULL;
  if (L == NULL) return NULL;
  else {
    while (aux->prox != NULL){
      pred = aux;
      aux = aux->prox;
    }
    if (pred == NULL) L = NULL;
    else pred->prox = NULL;
    free(aux);
    return L;
  }
}
```

Chamada da função:

```
Lista = excluiFinal(Lista);
```

```
void excluiFinal (struct no *L) {
  struct no *pred = L;
  struct no *aux = L->prox;
  if (aux != NULL){
    while (aux->prox != NULL) {
      pred = aux;
      aux = aux->prox;
    }
    pred->prox = NULL;
    free(aux);
  }
}
```

Chamada da função:

```
excluiFinal(Lista);
```

Nó Cabeça

- Como a implementação da função de exclusão no final mudaria?

```c
struct no *excluiFinal(struct no *L){
  struct no *aux = L;
  struct no *pred = NULL;
  if (L == NULL) return NULL;
  else {
    while (aux->prox != NULL){
      pred = aux;
      aux = aux->prox;
    }
    if (pred == NULL) L = NULL;
    else pred->prox = NULL;
    free(aux);
    return L;
  }
}
```

Chamada da função:

```c
Lista = excluiFinal(Lista);
```

```c
void excluiFinal (struct no *L) {
  struct no *pred = L;
  struct no *aux = L->prox;
  if (aux != NULL){
    while (aux->prox != NULL) {
      pred = aux;
      aux = aux->prox;
    }
    pred->prox = NULL;
    free(aux);
  }
}
```

Chamada da função:

```c
excluiFinal(Lista);
```

Nó Cabeça

- Como a implementação da função de busca em lista ordenada mudaria?

Nó Cabeça

- Como a implementação da função de busca em lista ordenada mudaria?

```
struct no *buscaListaOrd (struct no *L, int valor, struct no **pred){
  struct no *aux = L;
  (*pred) = NULL;
  if (L == NULL) return NULL;
  else{
    while (aux != NULL){
      if (valor <= aux->chave) break;
      (*pred) = aux;
      aux = aux->prox;
    }
    return aux;
  }
}
```

Chamada da função:

```
aux = buscaListaOrd(Lista,chave,&pred);
```

Nó Cabeça

- Como a implementação da função de busca em lista ordenada mudaria?

```
struct no *buscaListaOrd (struct no *L, int valor, struct no **pred){
  struct no *aux = L->prox;
  (*pred) = L;
  while (aux != NULL){
    if (valor <= aux->chave) break;
    (*pred) = aux;
    aux = aux->prox;
  }
  return aux;
}
```

Chamada da função:

```
aux = buscaListaOrd(Lista, ch, &pred);
```

Nó Cabeça

- Como a implementação da função de inserção em lista ordenada mudaria?

Nó Cabeça

- Como a implementação da função de inserção em lista ordenada mudaria?

```
struct no *insereListaOrd (struct no *L, int valor) {
  struct no *novo = criaNo(valor);
  struct no *pred = NULL;
  struct no *aux = buscaListaOrd(L, valor, &pred);
  novo->prox = aux;
  if (pred == NULL) L = novo;
  else pred->prox = novo;
  return L;
}
```

Chamada da função:

```
Lista = insereListaOrd(Lista, chave);
```

Nó Cabeça

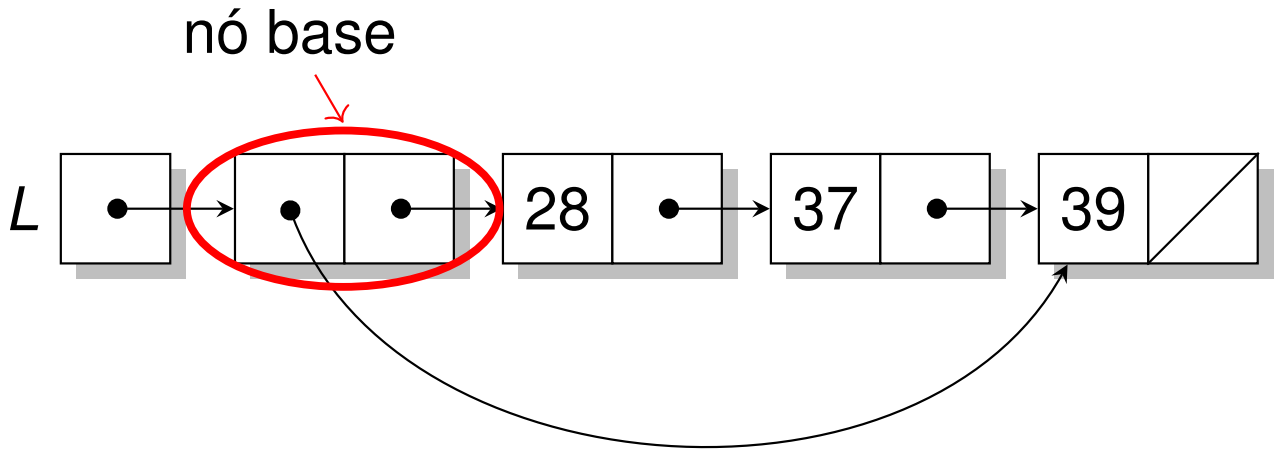
- Como a implementação da função de inserção em lista ordenada mudaria?

```
void insereOrd (struct no *L, int ch) {
  struct no *novo = criaNo(ch);
  struct no *pred = NULL;
  struct no *aux = buscaListaOrd(L, ch, &pred);
  novo->prox = aux;
  pred->prox = novo;
}
```

Chamada da função:

```
insereOrd(Lista,ch);
```

Nó Base

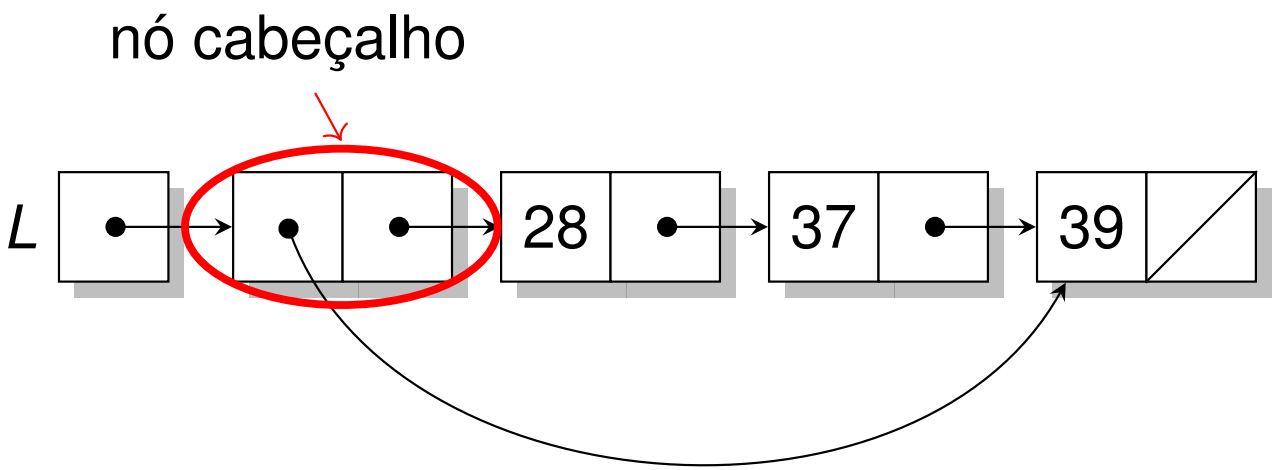


- Contém um ponteiro para o primeiro elemento da lista
- Contém um ponteiro para o último elemento da lista
- Como poderíamos implementar uma lista com um nó base?
- Para qual algoritmo visto essa estrutura é mais vantajosa?

Outra forma de implementar o nó cabeça

- De acordo com [Tenenbaum et al., 1995]

  "Se a parte *info* de um nó pode conter um ponteiro, surgem possibilidades adicionais para o uso de um nó de cabeçalho. Por exemplo, a parte *info* de um cabeçalho de lista poderia conter um ponteiro para o último na lista, (...)"



- Como podemos implementar a estrutura descrita?

References

Tenenbaum, A.M. and Langsam, Y. and Augenstein, M.J. (1995)

Estruturas de dados usando C

Editora Pearson Makron Books