

AULA 06

ESTRUTURA DE DADOS

Lista ligada (implementação dinâmica)

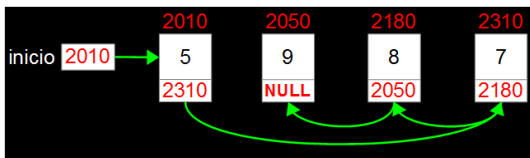
Norton T. Roman & Luciano A. Digiampietri

Lista ligada

Na última aula aprendemos como modelar e gerenciar **listas ligadas**.

- Realizamos o que chamamos de *implementação estática* (utilizamos um arranjo para armazenar nossos registros);
- Hoje aprenderemos a *implementação dinâmica*, isto é, alocaremos e desalocaremos a memória para os elementos **sob demanda**.
- Vantagens: não precisamos **gastar memória** que não estamos usando e não precisamos gerenciar uma lista de **elementos disponíveis**.

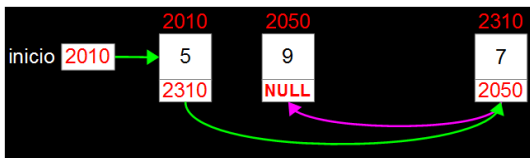
Lista ligada (ideia geral)



Temos um ponteiro para o primeiro elemento

Cada elemento indica seu sucessor

Lista ligada (ideia geral)

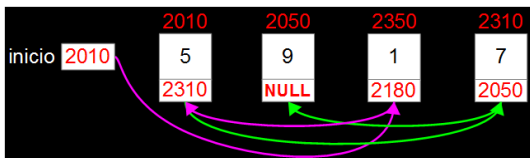


Temos um ponteiro para o primeiro elemento

Cada elemento indica seu sucessor

Como excluimos o elemento 8?

Lista ligada (ideia geral)



Temos um ponteiro para o primeiro elemento

Cada elemento indica seu sucessor

Como excluimos o elemento 8?

Como inserimos o elemento 1?

Modelagem

```
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0

typedef int bool;
typedef int TIPOCHAVE;

typedef struct {
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct aux {
    REGISTRO reg;
    struct aux* prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
    PONT inicio;
} LISTA;
```

Funções de gerenciamento

Implementaremos funções para:

- Inicializar a estrutura

- Retornar a quantidade de elementos válidos

- Exibir os elementos da estrutura

- Buscar por um elemento na estrutura

- Inserir elementos na estrutura

- Excluir elementos da estrutura

- Reinicializar a estrutura

Inicialização

Para inicializarmos nossa lista ligada, precisamos:

- Colocar o valor *NULL* na variável *inicio*.

```
void inicializarLista(LISTA* l){  
    l->inicio = NULL;  
}
```


Retornar número de elementos

Já que optamos por não criar um campo com o número de elementos na lista, precisaremos **percorrer todos os elementos** para contar quantos são.

```
int tamanho(LISTA* l) {  
    PONT end = l->inicio;  
    int tam = 0;  
    while (end != NULL) {  
        tam++;  
        end = end->prox;  
    }  
    return tam;  
}
```

Exibição/Impressão

Para exibir os elementos da estrutura precisaremos iterar pelos **elementos** e, por exemplo, **imprimir suas chaves**.

```
void exibirLista(LISTA* l){
    PONT end = l->inicio;
    printf("Lista: \" ");
    while (end != NULL) {
        printf("%i ", end->reg.chave);
        end = end->prox;
    }
    printf("\n");
}
```

Buscar por elemento

A função de busca deverá:

- Receber uma chave do usuário

- Retornar o endereço em que este elemento se encontra (caso seja encontrado)

- Retornar *NULL* caso não haja um registro com essa chave na lista

Busca sequencial

```
PONT buscaSequencial(LISTA* l, TIPOCHAVE ch) {
    PONT pos = l->inicio;
    while (pos != NULL) {
        if (pos->reg.chave == ch) return pos;
        pos = pos->prox;
    }
    return NULL;
}

// lista ordenada pelos valores das chaves dos registros
PONT buscaSeqOrd(LISTA* l, TIPOCHAVE ch) {
    PONT pos = l->inicio;
    while (pos != NULL && pos->reg.chave < ch) pos = pos->prox;
    if (pos != NULL && pos->reg.chave == ch) return pos;
    return NULL;
}
```

Inserção de um elemento

O usuário passa como parâmetro um registro a ser inserido na lista

Realizaremos a inserção **ordenada pelo valor da chave** do registro passado e não permitiremos a inserção de **elementos repetidos**; Na inserção precisamos identificar **entre quais elementos** o novo elemento será inserido;

Alocaremos memória para o novo elemento.

Precisamos saber quem será o **predecessor do elemento**.

Inserção ordenada

Desenvolveremos uma **função auxiliar** para procurar por uma dada chave e nos informar:

- O **endereço desse elemento** se ele existir;
 - O **endereço** de quem seria o **predecessor** desse elemento (independentemente do elemento existir ou não na lista).
- Como a função irá nos passar **dois endereços** diferentes?

“Passando dois resultados”

```
#include <stdio.h>
```

```
int funcaoQuadradoCubo(int x, int* y) {  
    *y = x*x*x;  
    return x*x;  
}
```

```
int main() {  
    int a = 30;  
    int cubo;  
    int quadrado = funcaoQuadradoCubo(a, &cubo);  
    printf("a: %i; a*a: %i; a*a*a: %i\n", a, quadrado, cubo);  
}
```

2040	a	30
2032	cubo	
2024	quadrado	

“Passando dois resultados”

```
#include <stdio.h>
```

```
int funcaoQuadradoCubo(int x, int* y) {  
    *y = x*x*x;  
    return x*x;  
}
```

```
int main() {  
    int a = 30;  
    int cubo;  
    int quadrado = funcaoQuadradoCubo(a, &cubo);  
    printf("a: %i; a*a: %i; a*a*a: %i\n", a, quadrado, cubo);  
}
```

2040	a	30	4200	x	30
2032	cubo		4192	y	2032
2024	quadrado				

“Passando dois resultados”

```
#include <stdio.h>
```

```
int funcaoQuadradoCubo(int x, int* y) {  
    *y = x*x*x;  
    return x*x;  
}
```

```
int main() {  
    int a = 30;  
    int cubo;  
    int quadrado = funcaoQuadradoCubo(a, &cubo);  
    printf("a: %i; a*a: %i; a*a*a: %i\n", a, quadrado, cubo);  
}
```

2040	a	30	4200	x	30
2032	cubo	27000	4192	y	2032
2024	quadrado				

“Passando dois resultados”

```
#include <stdio.h>
```

```
int funcaoQuadradoCubo(int x, int* y) {  
    *y = x*x*x;  
    return x*x;  
}
```

```
int main() {  
    int a = 30;  
    int cubo;  
    int quadrado = funcaoQuadradoCubo(a, &cubo);  
    printf("a: %i; a*a: %i; a*a*a: %i\n", a, quadrado, cubo);  
}
```

2040	a	30	4200	x	30
2032	cubo	27000	4192	y	2032
2024	quadrado				

“Passando dois resultados”

```
#include <stdio.h>
```

```
int funcaoQuadradoCubo(int x, int* y) {  
    *y = x*x*x;  
    return x*x;  
}
```

```
int main() {  
    int a = 30;  
    int cubo;  
    int quadrado = funcaoQuadradoCubo(a, &cubo);  
    printf("a: %i; a*a: %i; a*a*a: %i\n", a, quadrado, cubo);  
}
```

2040	a	30
2032	cubo	27000
2024	quadrado	900

“Passando dois resultados”

```
#include <stdio.h>
```

```
int funcaoQuadradoCubo(int x, int* y) {  
    *y = x*x*x;  
    return x*x;  
}
```

```
int main() {  
    int a = 30;  
    int cubo;  
    int quadrado = funcaoQuadradoCubo(a, &cubo);  
    printf("a: %i; a*a: %i; a*a*a: %i\n", a, quadrado, cubo);  
}
```

2040	a	30
2032	cubo	27000
2024	quadrado	900

Saída:

```
$ a: 30; a*a: 900; a*a*a: 27000
```

Busca - auxiliar

```
PONT buscaSequencialExc(LISTA* l, TIPOCHAVE ch, PONT* ant){
    *ant = NULL;
    PONT atual = l->inicio;
    while ((atual != NULL) && (atual->reg.chave < ch)) {
        *ant = atual;
        atual = atual->prox;
    }
    if ((atual != NULL) && (atual->reg.chave == ch)) return atual;
    return NULL;
}
```

Inserção ordenada

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
    TIPOCHAVE ch = reg.chave;
    PONT ant, i;
    i = buscaSequencialExc(l, ch, &ant);
    if (i != NULL) return false;
    i = (PONT) malloc(sizeof(ELEMENTO));
    i->reg = reg;
    if (ant == NULL) {
        i->prox = l->inicio;
        l->inicio = i;
    } else {
        i->prox = ant->prox;
        ant->prox = i;
    }
    return true;
}
```

Exclusão de um elemento

- O usuário passa a chave do elemento que ele quer excluir
- Se houver um elemento com esta chave na lista, **exclui este elemento** da lista, **acerta os ponteiros** envolvidos e retorna *true*.
- Caso contrário, retorna *false*
- Para esta função precisamos saber quem é o **predecessor** do elemento a ser excluído.

Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {  
    PONT ant, i;  
    i = buscaSequencialExc(l, ch, &ant);  
    if (i == NULL) return false;  
    if (ant == NULL) l->inicio = i->prox;  
    else ant->prox = i->prox;  
    free(i);  
    return true;  
}
```


Reinicialização da lista

Para reinicializar a estrutura, precisamos **excluir todos os seus elementos** e atualizar o campo *inicio* para *NULL*.

Reinicialização da lista

```
void reinicializarLista(LISTA* l) {  
    PONT end = l->inicio;  
    while (end != NULL) {  
        PONT apagar = end;  
        end = end->prox;  
        free(apagar);  
    }  
    l->inicio = NULL;  
}
```

AULA 06

ESTRUTURA DE DADOS

Lista ligada (implementação dinâmica)

Norton T. Roman & Luciano A. Digiampietri