



Estrutura de Dados

Ricardo José Cabeça de Souza

www.ricardojcsouza.com.br

ricardo.souza@ifpa.edu.br

Parte 9

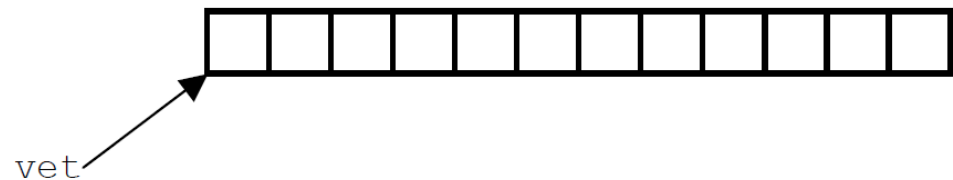
LISTAS



- **VETOR**

- Ao declararmos um vetor, reservamos um espaço contíguo de memória para armazenar seus elementos

```
#define MAX 1000  
int vet[MAX];
```



- Vetor não é uma estrutura de dados muito flexível, pois precisamos dimensioná-lo com um número máximo de elementos

LISTAS



- **LISTAS**

- Estruturas de dados que crescem à medida que precisarmos armazenar novos elementos (e diminuem à medida que precisarmos retirar elementos armazenados anteriormente)
- Tais estruturas são chamadas *dinâmicas* e armazenam cada um dos seus elementos usando alocação dinâmica

LISTAS



- **LISTAS ENCADEADAS**

– É uma sequência de células; cada célula contém um objeto de algum tipo e o endereço da célula seguinte



LISTAS



- **LISTAS ENCADEADAS**

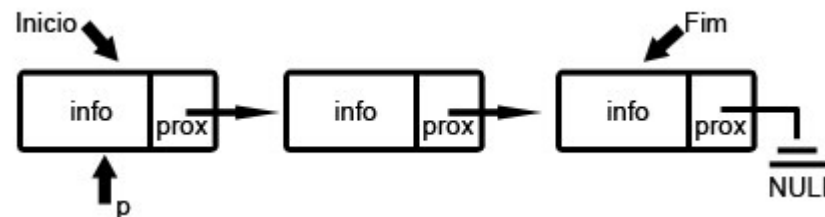
- Para cada novo elemento inserido na estrutura, alocamos um espaço de memória para armazená-lo
- O espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenado
- Não podemos garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo
- Devemos explicitamente guardar o encadeamento dos elementos, o que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista

LISTAS



- **LISTAS ENCADEADAS**

- É conveniente tratar as células como um novo tipo-de-dados e atribuir um nome a esse novo tipo (**typedef**)
- A estrutura consiste numa sequência encadeada de elementos, em geral chamados de **nós da lista**
- Os nós são ligados entre si para indicar a relação de ordem existente entre eles
- A lista é representada por um **ponteiro** para o primeiro elemento



- O último elemento da lista aponta para **NULL**, sinalizando que não existe um próximo elemento

LISTAS



- **LISTAS ENCADEADAS**

```
//Estrutura de cada célula
```

```
struct cel {  
    int          conteudo;  
    struct cel *prox;  
};
```

```
//Células serão um novo tipo-de-dados
```

```
//Atribuir um nome a esse novo tipo
```

```
typedef struct cel celula;
```

```
//Uma célula c e um ponteiro p para a célula são declarados
```

```
celula c;
```

```
celula *p;
```



conteudo prox

- Se **c** é uma célula então **c.conteudo** é o conteúdo da célula e **c.prox** é o endereço da próxima célula
- Se **p** é o endereço de uma célula, então **p->conteudo** é o conteúdo da célula e **p->prox** é o endereço da próxima célula
- Se **p** é o endereço da *última* célula da lista então **p->prox** vale **NULL**

LISTAS



- **ENDEREÇO DE UMA LISTA ENCADEADA**
 - O *endereço* de uma lista encadeada é o endereço de sua **primeira célula**
 - Se **p** é o endereço de uma **lista**, convém, às vezes, dizer simplesmente "**p é uma lista**"

LISTAS



- **LISTAS COM CABEÇA E SEM CABEÇA**

- Lista *com* cabeça

- O conteúdo da primeira célula é irrelevante: ela serve apenas para marcar o início da lista
- A primeira célula é a *cabeça* da lista
- A primeira célula está sempre no mesmo lugar na memória, mesmo que a lista fique vazia



LISTAS

- **LISTAS COM CABEÇA E SEM CABEÇA**

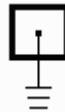
- Lista *com* cabeça

- Digamos que **ini** é o endereço da primeira célula.
Então **ini->prox == NULL** se e somente se a lista está **vazia**
- Para criar uma lista vazia, basta dizer:

```
celula c, *ini;  
c.prox = NULL;  
ini = &c;
```

```
celula *ini;  
ini = (celula*)malloc(sizeof(celula));  
ini->prox = NULL;
```

Representação:



LISTAS



- **LISTAS COM CABEÇA E SEM CABEÇA**

- Lista *sem* cabeça

- O conteúdo da primeira célula é tão relevante quanto o das demais
- Nesse caso, a lista está vazia se o endereço de sua primeira célula é NULL
- Para criar uma lista vazia basta fazer:

```
celula *ini;  
ini = NULL;
```



LISTAS

- **LISTAS – PRINCIPAIS FUNÇÕES**

- Função de inicialização

- Deve criar uma lista vazia, sem nenhum elemento
- Uma **lista vazia** é representada pelo ponteiro **NULL**

LISTAS



```
/*Lista para armazenar valores inteiros*/
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
/*Estrutura da Lista*/
struct cel {
    int conteudo;
    struct cel *prox;
};
/*Definição do novo tipo "celula"*/
typedef struct cel celula;
/* função de inicialização: retorna uma lista vazia */
celula *inicializa (void)
{
    return NULL;
}
/*Função Principal*/
main()
{
    celula *ini;
    ini = inicializa();
    getch();
}
```



LISTAS

• LISTAS – PRINCIPAIS FUNÇÕES

– Função de inserção

- Uma vez criada a lista vazia, podemos inserir novos elementos nela
- Para cada elemento inserido na lista, devemos alocar dinamicamente a memória necessária para armazenar o elemento e encadeá-lo na lista existente
- A função de inserção mais simples insere o novo elemento no início da lista
- A função de inserção recebe como parâmetros de entrada a lista onde será inserido o novo elemento e a informação do novo elemento, e tem como valor de retorno a nova lista

LISTAS



```
/* inserção no início: retorna a lista atualizada */
celula *insere (celula *l, int i)
{
    celula *novo = (celula*) malloc(sizeof(celula));
    novo->conteudo = i;
    novo->prox = l;
    return novo;
}
/*Função Principal*/
main()
{
    celula *ini;
    int valor;
    ini = inicializa();
    printf("Digite um numero:");
    scanf("%d", &valor);
    ini = insere(ini, valor);
    getch();
}
```

LISTAS

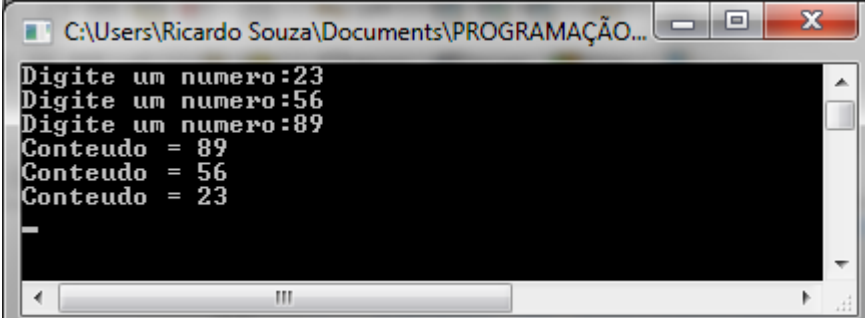


- **LISTAS – PRINCIPAIS FUNÇÕES**
 - Função exibir elementos da lista
 - Mostra os elementos da lista até o seu final
 - O final da lista é definido com o ponteiro para o próximo nó igual a **NULL**

LISTAS



```
/* inserção no início: retorna a lista atualizada */
celula *insere (celula *l, int i)
{
    celula *novo = (celula*) malloc(sizeof(celula));
    novo->conteudo = i;
    novo->prox = l;
    return novo;
}
/* função imprime: imprime valores dos elementos */
void imprime (celula *l)
{
    celula *p; /* variável auxiliar para percorrer a lista */
    for (p = l; p != NULL; p = p->prox)
        printf("Conteudo = %d\n", p->conteudo);
}
/*Função Principal*/
main()
{
    celula *ini;
    int valor,x=0;
    ini = inicializa();
    /*Inserir 3 elementos*/
    do {
        printf("Digite um numero:");
        scanf("%d",&valor);
        ini = insere(ini,valor);
        x++;
    }while (x<3);
    imprime(ini);
    getch();
}
```



```
C:\Users\Ricardo Souza\Documents\PROGRAMAÇÃO...
Digite um numero:23
Digite um numero:56
Digite um numero:89
Conteudo = 89
Conteudo = 56
Conteudo = 23
-
```

LISTAS



- **LISTAS – PRINCIPAIS FUNÇÕES**

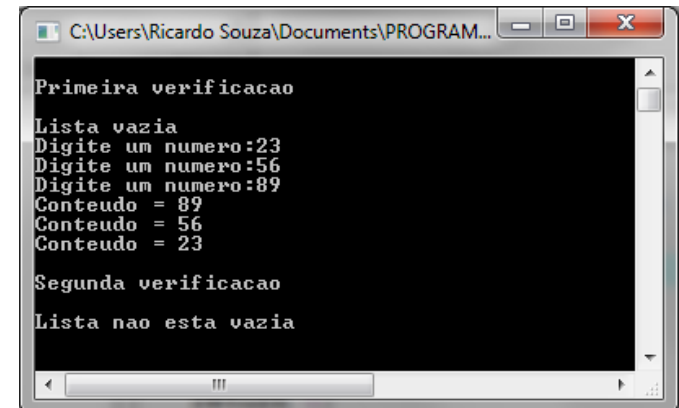
- Função que verifica se lista está vazia

- Função recebe a lista e retorna **1** se estiver vazia ou **0** se não estiver vazia
 - Uma lista está vazia se seu valor é NULL

LISTAS



```
/* função vazia: retorna 1 se vazia ou 0 se não vazia */
int vazia (celula *l)
{
    if (l == NULL)
        return 1;
    else
        return 0;
}
/*Função Principal*/
main()
{
    celula *ini;
    int valor,x=0,verifica;
    ini = inicializa();
    verifica = vazia(ini);
    printf("\nPrimeira verificacao\n");
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    /*Inserir 3 elementos*/
    do {
        printf("Digite um numero:");
        scanf("%d",&valor);
        ini = insere(ini,valor);
        x++;
    }while(x<3);
    imprime(ini);
    printf("\nSegunda verificacao\n");
    verifica = vazia(ini);
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    getch();
}
```



```
C:\Users\Ricardo Souza\Documents\PROGRAM...
Primeira verificacao
Lista vazia
Digite um numero:23
Digite um numero:56
Digite um numero:89
Conteudo = 89
Conteudo = 56
Conteudo = 23

Segunda verificacao
Lista nao esta vazia
```

LISTAS



- **LISTAS – PRINCIPAIS FUNÇÕES**
 - Função de busca
 - Consiste em verificar se um determinado elemento está presente na lista



LISTAS



```
/* função busca: busca um elemento na lista */
int busca (celula *l, int v)
{
    celula *p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->conteudo == v)
            return 1;
    return 0; /* não achou o elemento */
}

/*Função Principal*/
main()
{
    celula *ini;
    int valor,x=0,verifica;
    ini = inicializa();
    verifica = vazia(ini);
    printf("\nPrimeira verificacao\n");
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    /*Inserir 3 elementos*/
    do {
        printf("Digite um numero:");
        scanf("%d",&valor);
        ini = insere(ini,valor);
        x++;
    }while(x<3);
    imprime(ini);
    printf("\nSegunda verificacao\n");
    verifica = vazia(ini);
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    printf("Informe o valor a pesquisar:");
    scanf("%d",&valor);
    ini = busca(ini,valor);
    if(ini == NULL)
        printf("\nValor nao encontrado!");
    else
        printf("\nValor encontrado!");
    getch();
}
```

```
C:\Users\Ricardo Souza\Do...
Primeira verificacao
Lista vazia
Digite um numero:23
Digite um numero:56
Digite um numero:89
Conteudo = 89
Conteudo = 56
Conteudo = 23

Segunda verificacao

Lista nao esta vazia
Informe o valor a pesquisar:89

Valor encontrado!
```

```
C:\Users\Ricardo Souza\Documents...
Primeira verificacao
Lista vazia
Digite um numero:23
Digite um numero:56
Digite um numero:89
Conteudo = 89
Conteudo = 56
Conteudo = 23

Segunda verificacao

Lista nao esta vazia
Informe o valor a pesquisar:5

Valor nao encontrado!
```

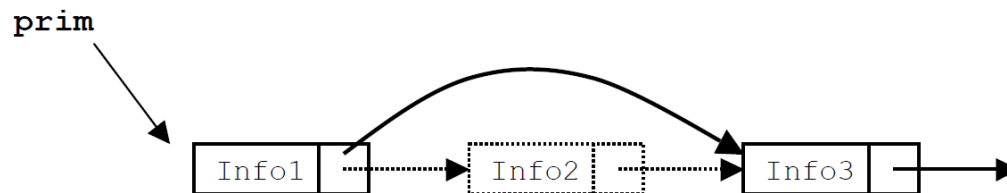
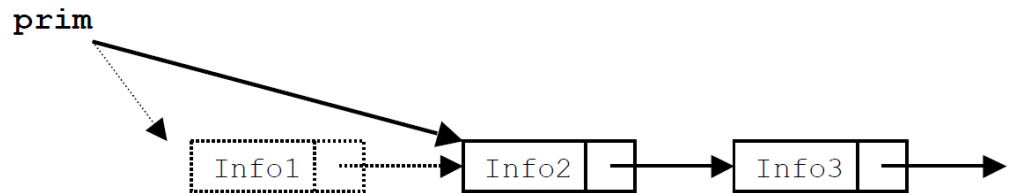


LISTAS

• LISTAS – PRINCIPAIS FUNÇÕES

– Função de busca e remoção

- Dado um inteiro y , remover da lista a primeira célula que contém y
- Função que retira um elemento da lista



LISTAS



```
/*Função Principal*/
main()
{
    celula *ini;
    int valor,x=0,verifica,procura;
    ini = inicializa();
    verifica = vazia(ini);
    printf("\nPrimeira verificacao\n");
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    /*Inserir 3 elementos*/
    do {
        printf("Digite um numero:");
        scanf("%d",&valor);
        ini = insere(ini,valor);
        x++;
    }while(x<3);
    imprime(ini);
    printf("\nSegunda verificacao\n");
    verifica = vazia(ini);
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    printf("Informe o valor a PESQUISAR:");
    scanf("%d",&valor);
    procura = busca(ini,valor);
    if(procura == 0)
        printf("\nValor nao encontrado!");
    else
        printf("\nValor encontrado!");
    printf("\nInforme o valor a RETIRAR:");
    scanf("%d",&valor);
    ini = retira(ini,valor);
    /*Exibicao lista atualizada*/
    imprime(ini);
    getch();
}
```

```
/* função retira: retira elemento da lista */
celula *retira (celula *l, int v) {
    celula *ant = NULL; /* ponteiro para elemento anterior */
    celula *p = l; /* ponteiro para percorrer a lista*/
    /* procura elemento na lista, guardando anterior */
    while (p != NULL && p->conteudo != v)
    {
        ant = p;
        p = p->prox;
    }
    /* verifica se achou elemento */
    if (p == NULL)
        return l; /* não achou: retorna lista original */
    /* retira elemento */
    if (ant == NULL) {
        /* retira elemento do inicio */
        l = p->prox;
    }
    else {
        /* retira elemento do meio da lista */
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```



LISTAS

- **LISTAS – PRINCIPAIS FUNÇÕES**

- Função para liberar a lista

- Destroi a lista, liberando todos os elementos alocados
- A função percorre elemento a elemento, liberando-os

LISTAS



```
/*Função Principal*/
main()
{
    celula *ini;
    int valor,x=0,verifica,procura;
    ini = inicializa();
    verifica = vazia(ini);
    printf("\nPrimeira verificacao\n");
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    /*Inserir 3 elementos*/
    do {
        printf("Digite um numero:");
        scanf("%d",&valor);
        ini = insere(ini,valor);
        x++;
    }while(x<3);
    imprime(ini);
    printf("\nSegunda verificacao\n");
    verifica = vazia(ini);
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    printf("Informe o valor a PESQUISAR:");
    scanf("%d",&valor);
    procura = busca(ini,valor);
    if(procura == 0)
        printf("\nValor nao encontrado!");
    else
        printf("\nValor encontrado!");
    printf("\nInforme o valor a RETIRAR:");
    scanf("%d",&valor);
    ini = retira(ini,valor);
    /*Exibicao lista atualizada*/
    imprime(ini);
    /*Liberacao da lista*/
    ini=libera(ini);
    /*Verifica se realmente está vazia*/
    verifica = vazia(ini);
    printf("\nUltima verificacao\n");
    if(verifica == 1)
        printf("\nLista vazia\n");
    else
        printf("\nLista nao esta vazia\n");
    getch();
}
```

```
/* função libera: retira todos os elemento da lista */
celula *libera (celula *l)
{
    celula *p = l,*t;
    while (p != NULL)
    {
        t = p->prox; /* guarda referência para o próximo elemento */
        free(p); /* libera a memória apontada por p */
        p = t; /* faz p apontar para o próximo */
    }
    return p;
}
```

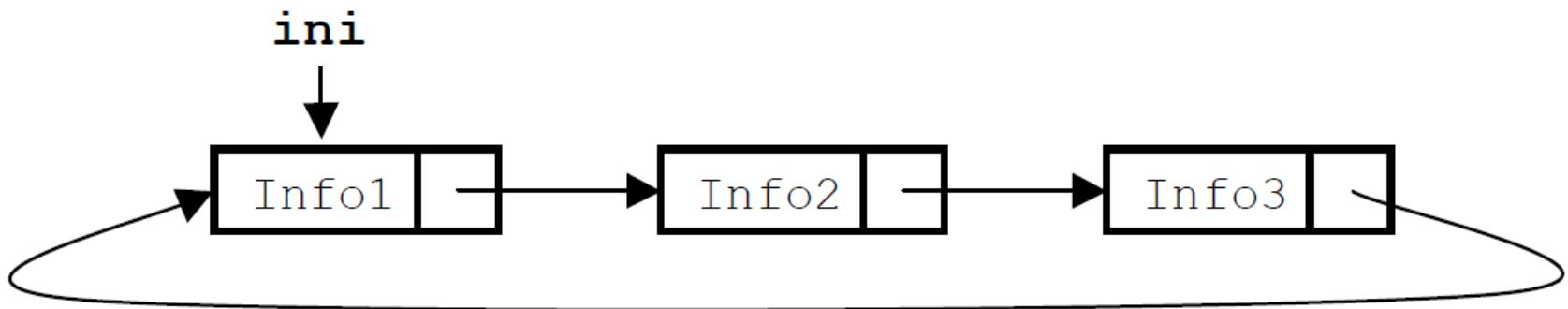
LISTAS



- **LISTAS**

- **Listas circulares**

- O último elemento tem como próximo o primeiro elemento da lista, formando um ciclo
 - Para percorrer os elementos de uma lista circular, visitamos todos os elementos a partir do ponteiro do elemento inicial até alcançarmos novamente esse mesmo elemento

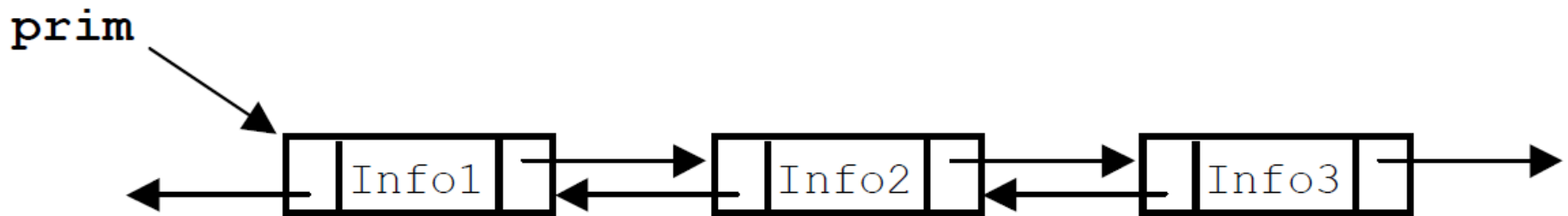


LISTAS



- **LISTAS DUPLAMENTE ENCADEADAS**

- Cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior
- Desta forma, dado um elemento, podemos acessar ambos os elementos adjacentes: o próximo e o anterior





LISTAS

- **LISTAS DUPLAMENTE ENCADEADAS**

```
/*Estrutura da Lista Duplamente Encadeada*/  
struct cels {  
    int conteudo;  
    struct cels *prox;  
    struct cels *ant;  
};  
/*Definição do novo tipo "celula"*/  
typedef struct cels duplcelula;
```

Estrutura de Dados



- **REFERÊNCIAS**
- Feofiloff, Paulo. **Projeto de Algoritmos em C**. Disponível em <http://www.ime.usp.br/~pf/algoritmos/aulas/lista.html> acesso em 12/07/2011.
- Tenenbaum, Aaron M. Langsam, Yedidyah, Augenstein, Moshe J. **Estruturas de dados usando C**. São Paulo : MAKRON Books, 1995.
- Veloso, Paulo. et. al. **Estrutura de dados**. Rio de Janeiro: Campus, 1986.
- Moraes, Celso Roberto. **Estrutura de dados e algoritmos**. 2. ed. São Paulo: Futura, 2003.
- Celes, W. Rangel, J. L. **Curso de Estrutura de Dados**. PUC-Rio, 2002.
- W. Celes, R. Cerqueira, J.L. Rangel. **Introdução a Estruturas de Dados - com técnicas de programação em C**. Rio de Janeiro: Campus, 2004.