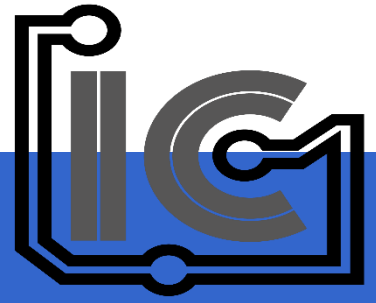
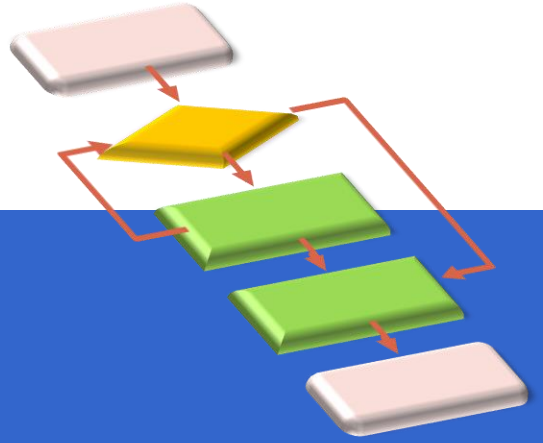


INSTITUTO DE COMPUTAÇÃO



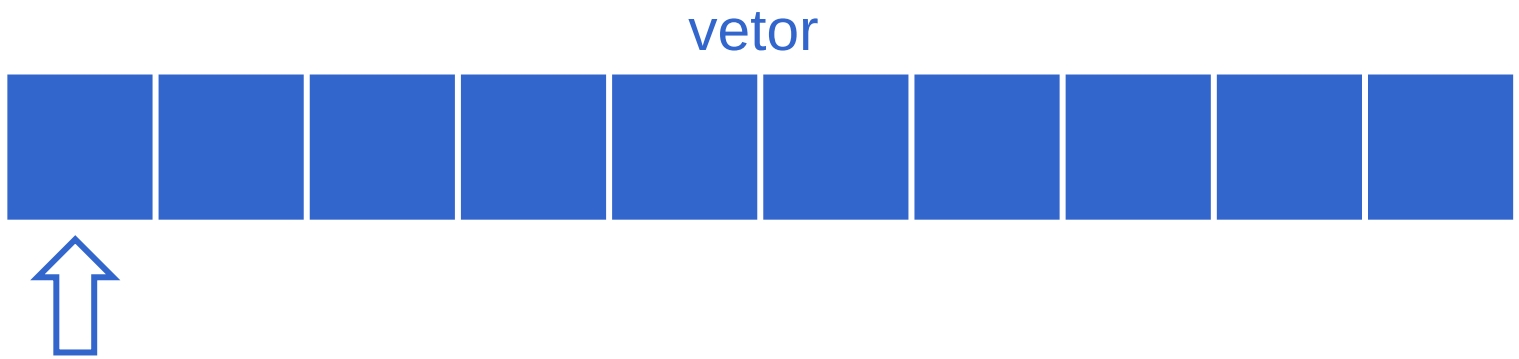
Algoritmos II

ALOCAÇÃO DINÂMICA DE MEMÓRIA
Listas Encadeadas Dinâmicas

Prof.ª Vanessa de Oliveira Campos

Introdução

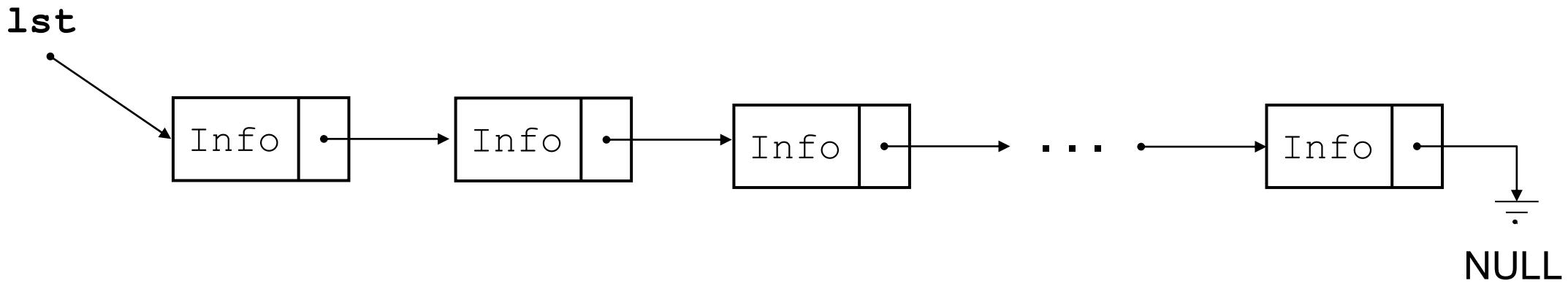
- Vetor
  - ocupa um espaço contíguo de memória;
  - permite acesso randômico aos elementos;
  - número fixo de elementos.



Motivação

- Estruturas de dados dinâmicas:
  - crescem (ou decrescem) à medida que elementos são inseridos (ou removidos).
  - Exemplo:
    - listas encadeadas: amplamente usadas para implementar outras estruturas de dados.

Listas Encadeadas

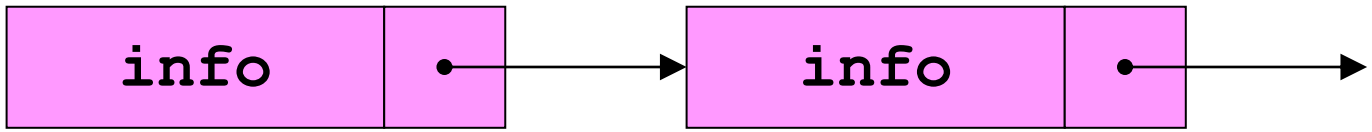


Listas Encadeadas

- Lista encadeada:
  - sequência encadeada de elementos, chamados de nós da lista.
  - nó da lista é representado por dois tipos de campos:
    - a informação armazenada e;
    - o ponteiro para o próximo elemento da lista;
  - a lista é representada por um ponteiro para o primeiro nó;
  - o ponteiro do último elemento é NULL.

Estrutura com ponteiro para ela mesma

```
struct elemento{
   int info;
   struct elemento *prox;
};
typedef struct elemento TNo;
```



Exemplo: Lista de inteiros

```
struct elemento{
   int info;
   struct elemento* prox;
};
typedef struct elemento TNo;
```

- Lista é uma estrutura auto-referenciada, pois o campo **prox** é um ponteiro para uma próxima estrutura do mesmo tipo.
- uma lista encadeada é representada pelo ponteiro para seu primeiro elemento, do tipo **TNo***.

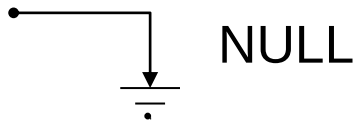
Exemplo: Lista de inteiros (outra forma)

```
typedef struct elemento TNo;

struct elemento{
   int info;
   Tno* prox;
};
```

Listas Encadeadas de inteiros: Criação

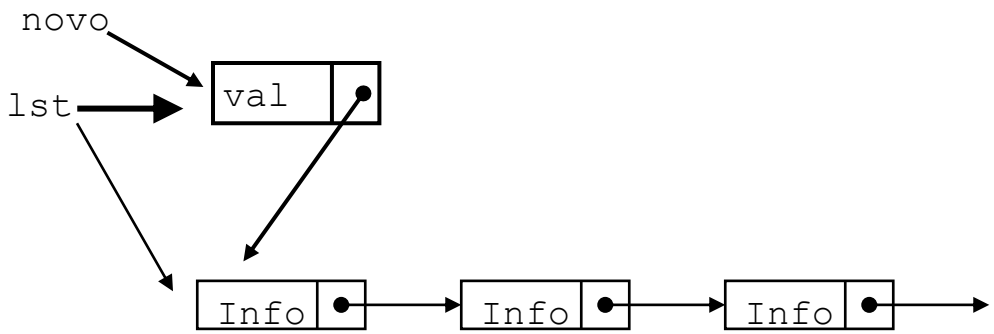
```
/* função de criação: retorna uma lista vazia */
TNo* lst_cria (void)
{
   return NULL;
}
```



Cria uma lista vazia, representada pelo ponteiro NULL

Listas Encadeadas de inteiros: Inserção

- aloca memória para armazenar o elemento;
- encadeia o elemento na lista existente.




```
/* inserção no início: retorna a lista atualizada */
TNo* lst_insere (TNo* lst, int val)
{
    TNo* novo = (TNo*) malloc(sizeof(TNo));
    novo->info = val;
    novo->prox = lst;
    return novo;
}
```

```
lst = lst_insere(lst, 35); /* insere na lista o elemento 35 */
```

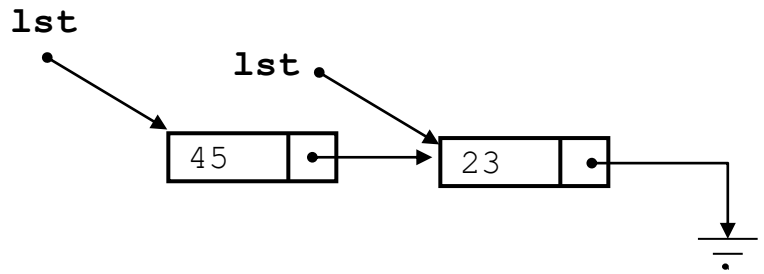
Listas Encadeadas: Exemplo

- Cria uma lista e insere elementos

```
void main (void)
{
    TNo* lst;                  /* declara uma lista não inicializada */
    lst = lst_cria();          /* cria e inicializa lista como vazia */

    lst = lst_insere(lst, 23); /* insere na lista o elemento 23 */
    lst = lst_insere(lst, 45); /* insere na lista o elemento 45 */
    ...
    return 0;
}
```

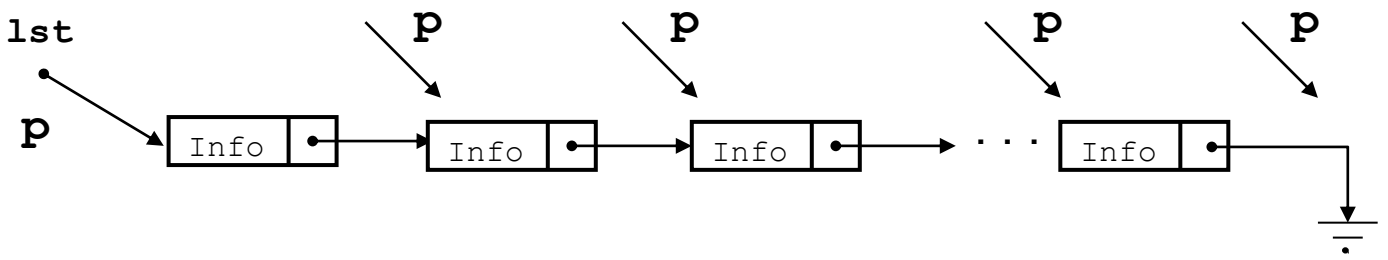
a variável que representa a lista é atualizada a cada inserção de um novo elemento.



Listas Encadeadas: impressão

- Imprime os valores dos elementos armazenados

```
/* função imprime: imprime valores dos elementos */
void lst_imprime (TNo* lst)
{
    TNo* p;
    for (p = lst; p != NULL; p = p->prox)
       printf("info = %d\n", p->info);
}
```



Listas Encadeadas: Teste de vazia

- Retorna 1 se a lista estiver vazia ou 0 se não estiver vazia

```
/* função vazia: retorna 1 se vazia ou 0 senão vazia */
int lst_vazia (TNo* lst)
{
    return (lst == NULL);
}
```

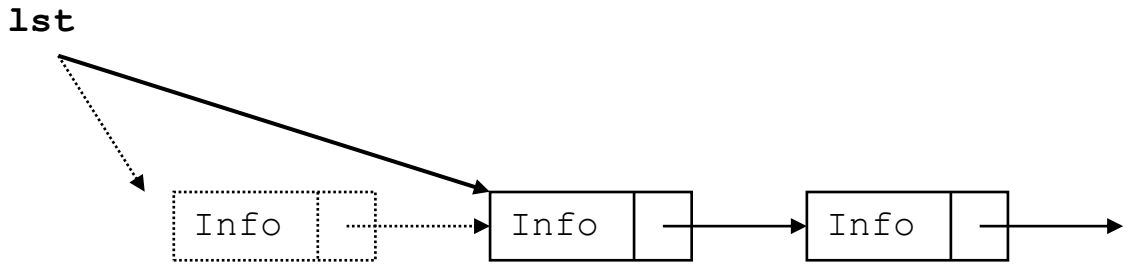
Listas Encadeadas: Busca

- recebe a informação referente ao elemento a pesquisar
- retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

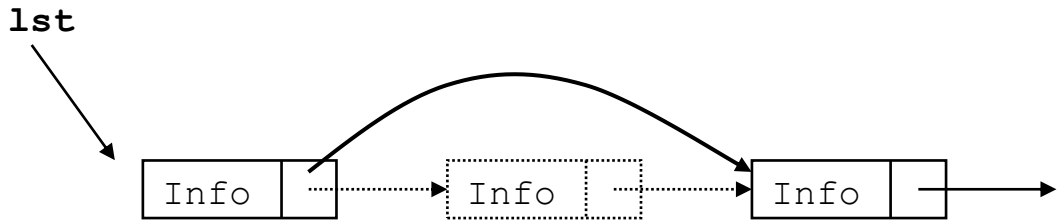
```
/* função busca: busca um elemento na lista */
TNo* busca (TNo* lst, int v)
{
   TNo* p;
   p = lst;
   while ((p!= NULL) && (p->info != v))
      p = p->prox
   return p;  /* se p == NULL, não encontrou o elemento */
}
```

Listas Encadeadas: remover um elemento

- recebe como entrada a lista e o valor do elemento a retirar
- atualiza o valor da lista, se o elemento removido for o primeiro



- caso contrário, apenas remove o elemento da lista



```c
/* função que retira um elemento val da lista */
TNo* lst_retira (TNo* lst, int val)
{
   TNo* a = NULL;   /* ponteiro para elemento anterior */
   TNo* p = lst;    /* ponteiro para percorrer a lista */
   /* procura elemento na lista, guardando anterior */
   while (p != NULL && p->info != val)
   {
      a = p;
      p = p->prox;
   }
   if (p != NULL) /* encontrou o elemento */
   {
      if (a == NULL) /* retira elemento do inicio */
         lst = p->prox;
      else    /* retira elemento do meio da lista */
         a->prox = p->prox;
      free(p);
   }
   return lst; /* retorna lista atualizada */
}
```

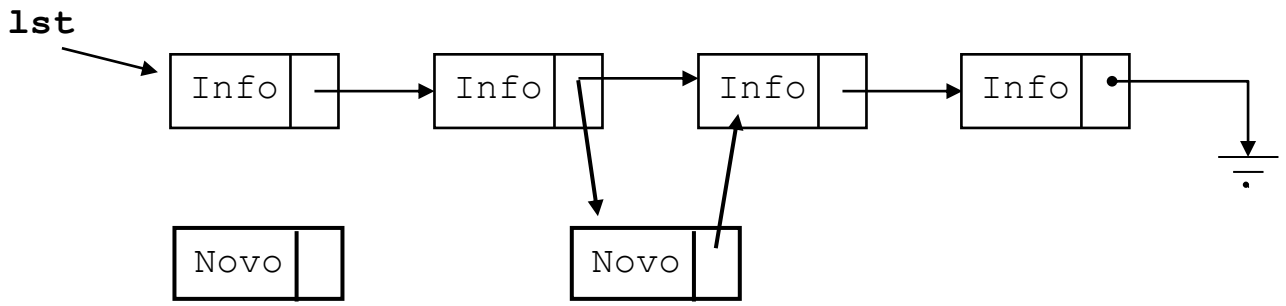
Listas Encadeadas: Esvazia a lista

- Elimina a lista, liberando todos os elementos alocados

```
void lst_libera (TNo* lst)
{
    TNo* p = lst;
    TNo* t;
    while (p != NULL) {
        t = p->prox;    /* guarda referência p/ próx. TNo    */
        free(p);        /* libera a memória apontada por p   */
        p = t;          /* faz p apontar para o próximo      */
    }
}
```

Listas Encadeadas

- Manutenção da lista ordenada:
  - função de inserção percorre os elementos da lista até encontrar a posição correta para a inserção do novo.



```c
/* função insere_ordenado: insere elemento em ordem */
TNo* lst_insere_ordenado (TNo* lst, int val)
{
    TNo* novo;
    TNo* a = NULL;    /* ponteiro para elemento anterior  */
    TNo* p = lst;     /* ponteiro para percorrer a lista  */
    while (p != NULL && p->info < val) /* procura posição de inserção */
    {
        a = p;
        p = p->prox;
    }
    novo = (TNo*) malloc(sizeof(TNo)); /* cria novo elemento */
    novo->info = val;
    /* encadeia elemento */
    if (a == NULL) /* insere elemento no início     */
    {
        novo->prox = lst;
        lst = novo;
    }
    else  /* insere elemento no meio da lista */
    {
        novo->prox = a->prox;
        a->prox = novo;
    }
    return lst;
}
```

Listas de Tipos Estruturados

- Lista de tipo estruturado:
  - a informação associada a cada nó de uma lista encadeada pode ser mais complexa, sem alterar o encadeamento dos elementos;
  - as funções apresentadas para manipular listas de inteiros podem ser adaptadas para tratar listas de outros tipos.

Listas de Tipos Estruturados

- Lista de tipo estruturado (cont.):
  - o campo da informação pode ser representado por um ponteiro para uma estrutura, em lugar da estrutura em si;
  - independente da informação armazenada na lista, a estrutura do nó é sempre composta por:
    - um ponteiro para a informação e
    - um ponteiro para o próximo nó da lista.

Listas de Tipos Estruturados

- Exemplo – Lista de retângulos

```
typedef struct elemento TRetangulo;

struct elemento {
    float b;
    float h;
    TRetangulo *prox;
};
```

Listas de Tipos Estruturados

- Exemplo – Lista de retângulos (outra possibilidade)

```
struct retangulo {
    float b;
    float h;
};
typedef struct retangulo TRetangulo;

struct elemento {
    TRetangulo * info;
    struct elemento *prox;
};
typedef struct elemento TNo;
```

campo da informação representado por um ponteiro para uma estrutura, em lugar da estrutura em si

Listas de Tipos Estruturados

- Exemplo – Função auxiliar para alocar um nó

```
static TNo* aloca (float b, float h)
{
   TRetangulo *r;
   TNo *p;
   r = (TRetangulo *) malloc(sizeof(TRetangulo));
   p = (TNo*) malloc(sizeof(TNo));
   r->b = b;
   r->h = h;
   p->info = r;
   p->prox = NULL;
   return p;
}
```

Para alocar um nó, são necessárias duas alocações dinâmicas:
uma para criar a estrutura do retângulo e outra para criar a estrutura do nó.

O valor da base associado a um nó **p** seria acessado por: **p->info->b**.

Exercício de Fixação

1) Faça o algoritmo de uma função que retorne a quantidade de elementos da lista.

Exercício de Fixação

2) Faça o algoritmo de uma função que inclua um elemento no final de uma lista.

Exercício de Fixação

3) Faça o algoritmo de um procedimento que exclua de uma lista de inteiros, todos os elementos que tenham o mesmo valor passado como parâmetro.