

INSTITUTO FEDERAL

Bahia

Campus Santo Antônio de Jesus

<http://www.portal.ifba.edu.br/santoantonio>

ESTRUTURA DE DADOS

Prof. George Pacheco Pinto

AGENDA

- ❑ Listas Lineares
 - ❑ Listas não sequenciais

LISTAS ESTÁTICAS - VETORES

❑ Vantagens

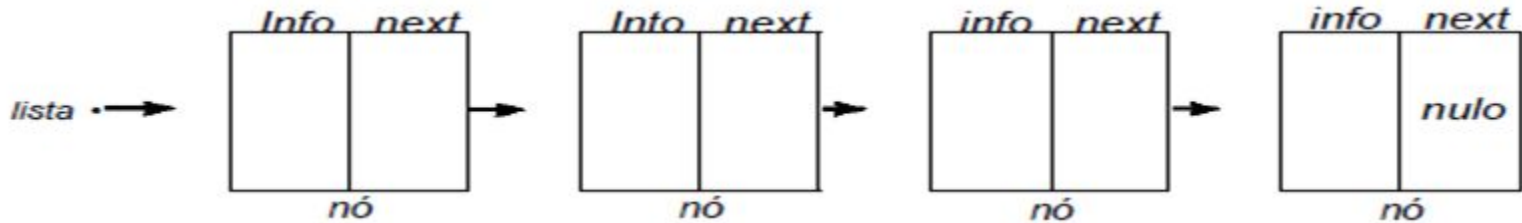
- ❑ Acesso direto indexado a qualquer elemento (facilita modificações dos conteúdos);
- ❑ Tempo constante para acessar os elementos (depende só do índice).

❑ Desvantagens

- ❑ Movimentação para a inserção e remoção de elementos;
- ❑ Conhecimento a priori do tamanho do número máximo da lista. Depois da alocação, não mais do que essa quantidade poderá ser solicitada;
- ❑ Quantidade fixa de memória permanece alocada, mesmo quando a estrutura estiver vazia.

LISTAS NÃO SEQUENCIAIS DINÂMICAS OU ENCADEADAS

Cada item da lista é encadeado com o seguinte através de uma variável que aponte o próximo item;



Cada item na lista é chamado de nó e contém basicamente dois campos, um de informação e outro com o endereço do seguinte.

LISTAS ENCADEADAS DINÂMICAS

❑ Desvantagens

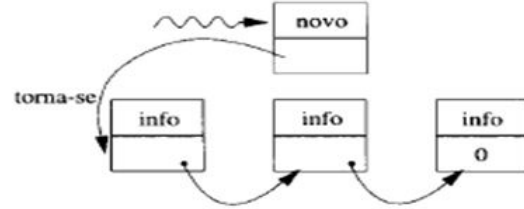
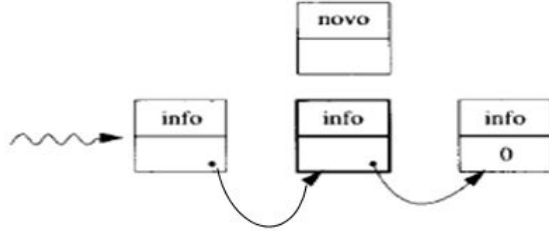
- ❑ Acesso indireto aos elementos
- ❑ Tempo variável para acessar os elementos (depende da posição do elemento)
- ❑ Gasto de memória maior pela necessidade de um novo campo para o apontador

❑ Vantagens

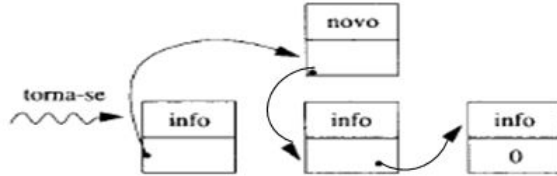
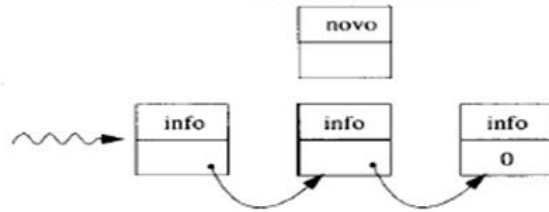
- ❑ A inserção e remoção de elementos podem ser feitas sem deslocar os itens seguintes da lista (custo constante)
- ❑ Não há necessidade de previsão do número de elementos da lista; o espaço necessário é alocado em tempo de execução
- ❑ Facilita o gerenciamento de várias listas (fusão, divisão,...)

INSERINDO UM NOVO NÓ

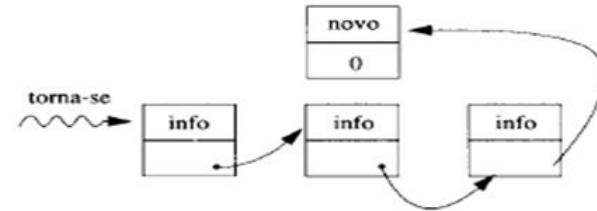
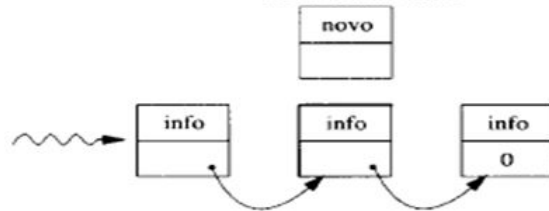
Novo Primeiro Item



Novo Item do Meio

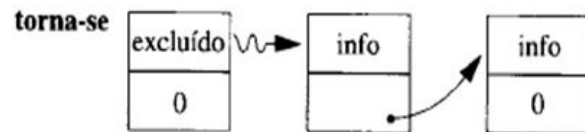
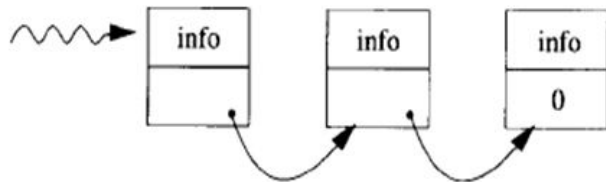


Novo Último Item

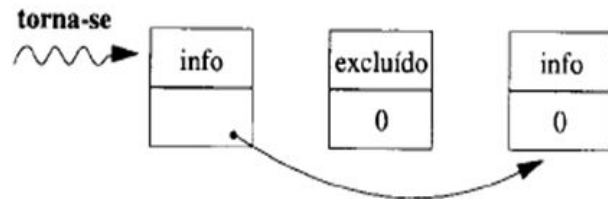
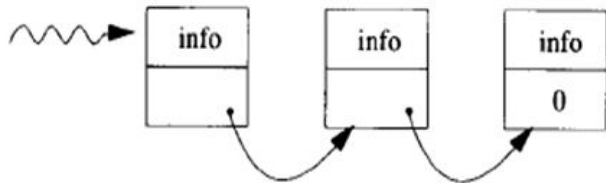


REMOVENDO UM NÓ

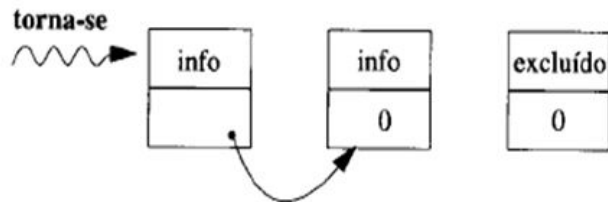
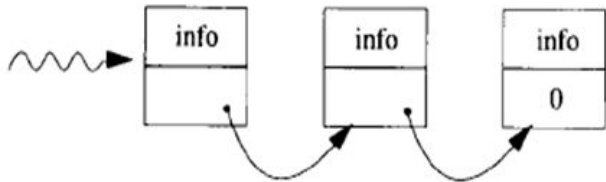
Apagando o primeiro item



Apagando o item do meio



Apagando o último item



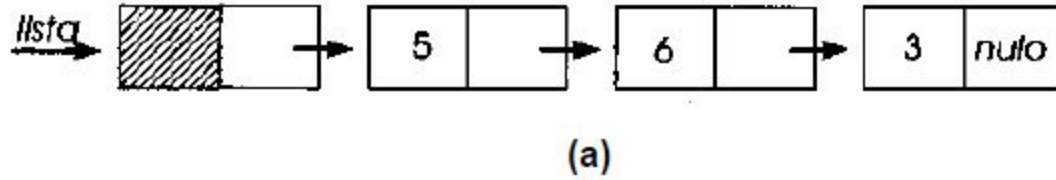
LISTAS ENCADEADAS

- ❑ Deve existir sempre uma indicação do **primeiro nó**: a partir dele a lista é percorrida
- ❑ Então, seria necessário um cuidado especial no tratamento do primeiro nó da lista:
 - ❑ Testes em algoritmos de inserção e remoção para verificar se este ponteiro é ou não igual a NULL
- ❑ Uma pequena variação na estrutura com a adição de um nó chamado nó-cabeça (sentinela) evita alguns testes com o primeiro nó e melhora o desempenho das operações na lista

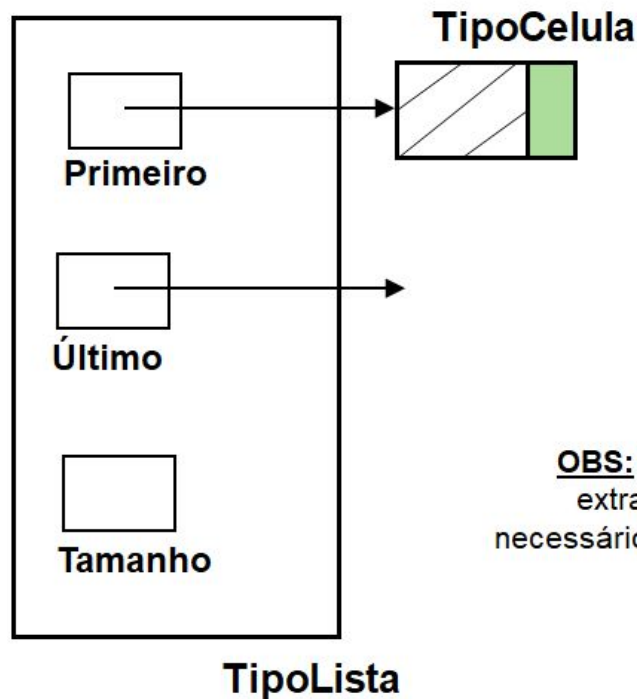
NÓS DE CABEÇALHO

- ❑ O nó-cabeça não contém informações relacionadas aos dados da lista e nunca é removido;
- ❑ A parte de informações, pode ser usada para manter informações globais sobre a lista.

NÓS DE CABEÇALHO



IMPLEMENTAÇÃO DE LISTAS ENCADEADAS USANDO PONTEIROS



OBS: Vamos considerar uma variável extra "tamanho" para evitar que seja necessário percorrer a lista para descobrir o seu tamanho

ALOCACÃO DINÂMICA

ALOCÇÃO DE MEMÓRIA

- ❑ Uso da memória em C
 - ❑ uso de variáveis globais (e estáticas). O espaço reservado para uma variável global existe enquanto o programa estiver sendo executado;
 - ❑ uso de variáveis locais. Neste caso, o espaço existe apenas enquanto a função que declarou a variável está sendo executada;
 - ❑ reservar memória requisitando ao sistema, em tempo de execução, um espaço de um determinado tamanho – Alocação Dinâmica de Memória

ALOCAÇÃO DINÂMICA DE MEMÓRIA

- ❑ Muitas vezes a quantidade de memória a se alocar só é conhecida em tempo de execução;
- ❑ Além disso, definir um tamanho máximo para suas estruturas de dados gera desperdício de memória;
- ❑ A solução é alocar a memória necessária, quando realmente precisar dela – **Alocação dinâmica**

ALOCACÃO DINÂMICA DE MEMÓRIA

- ❑ O espaço alocado dinamicamente permanece reservado até que seja explicitamente liberado pelo programa.
- ❑ A partir do momento que liberarmos o espaço, ele fica disponível para outros usos e não podemos mais acessá-lo.

ALOCACÃO DINÂMICA - INSTRUÇÕES

- ❑ C define 4 instruções para gerenciar alocação dinâmica de memória. Disponíveis na biblioteca `<stdlib.h>`

São elas:

- ❑ `malloc`
- ❑ `calloc`
- ❑ `realloc`
- ❑ `free`

ALOCAÇÃO DINÂMICA - INSTRUÇÕES

- ❑ malloc - permite alocar blocos de memória em tempo de execução

```
void *malloc (int tamanho)
```



Número de bytes
alocados


retorna um ponteiro void para n bytes de memória não iniciados. Se não há memória disponível malloc retorna NULL

ALOCAÇÃO DINÂMICA - INSTRUÇÕES


❑ `calloc`

```
void * calloc(n, size);
```

Tamanho em bytes
de cada elemento.



Número de
elementos a ser
alocado



`calloc` retorna um ponteiro para um array com `n` elementos de tamanho `size` cada um ou `NULL` se não houver memória disponível.

ALOCAÇÃO DINÂMICA - INSTRUÇÕES

❑ Exemplos

❑ Código que aloca memória para um inteiro

```
int *p;  
p = (int *) malloc (sizeof(int));
```

❑ Código que aloca memória para um vetor de 50 inteiros

```
int *ai = (int *) calloc (50, sizeof(int));
```

ALOCAÇÃO DINÂMICA - INSTRUÇÕES

- ❑ `realloc` - usado para redimensionar o espaço alocado previamente com `malloc` ou `calloc`

Exemplo

```
int *A;  
A = calloc (n,sizeof(double));  
...  
RA = realloc (A,2*n);
```

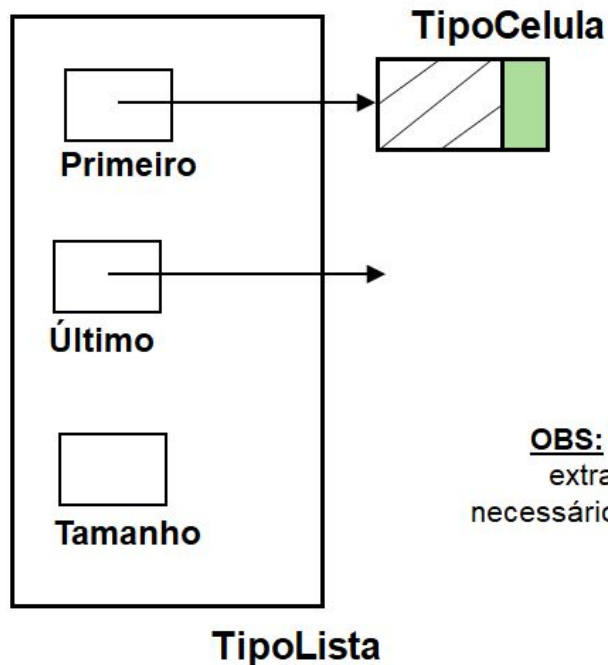
ALOCAÇÃO DINÂMICA - INSTRUÇÕES

- ❑ Toda memória não utilizada deve ser liberada. Para isso usa-se a instrução `free()`.

```
int *p;  
p = (int * )malloc (sizeof(int));  
free (p);
```

IMPLEMENTANDO LISTAS ENCADEADAS EM C

IMPLEMENTAÇÃO DE LISTAS ENCADEADAS USANDO PONTEIROS



OBS: Vamos considerar uma variável extra "tamanho" para evitar que seja necessário percorrer a lista para descobrir o seu tamanho

LISTA ENCADEADA - TAD

```
typedef int TipoChave;

typedef struct
{
    TipoChave chave;
} TipoItem;

typedef struct TipoCelula
*Apontador;
```

```
struct TipoCelula
{
    TipoItem Item;
    Apontador Prox;
} celula;

typedef struct
{
    Apontador Primeiro, Ultimo;
    int tamanho;
} TipoLista;
```


IMPLEMENTAÇÃO OPERAÇÕES BÁSICAS

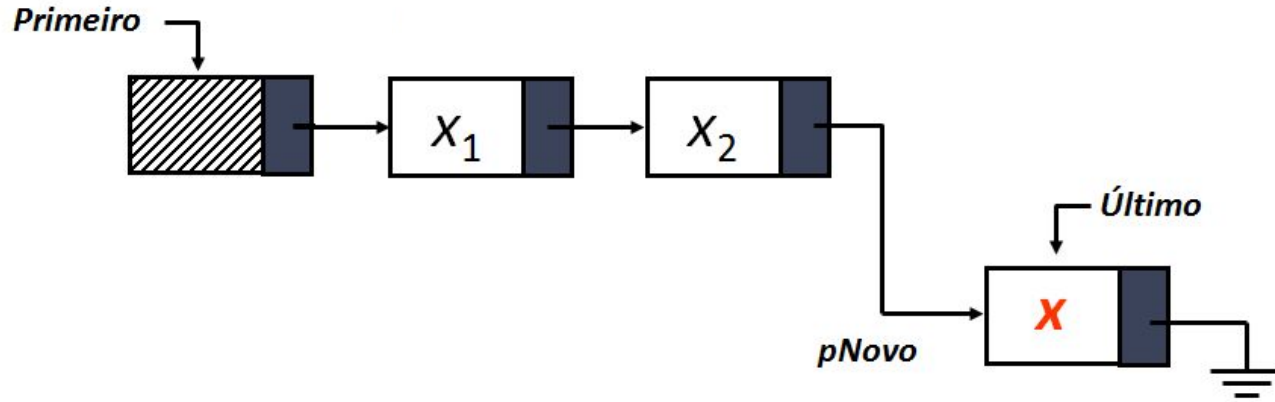
```
void Inicia(TipoLista *Lista)
{
    Lista->Primeiro = (Apontador) malloc(sizeof(celula));
    Lista->Ultimo = Lista->Primeiro;
    Lista->Primeiro->Prox = NULL;
    Lista->tamanho = 0;
}
```

```
int Vazia(TipoLista *Lista)
{
    return (Lista->Primeiro == Lista->Ultimo);
}
```

IMPLEMENTAÇÃO OPERAÇÕES BÁSICAS

```
void Insere(TipoItem x, TipoLista *Lista)
/*A inserção é feita à direita do ponteiro Último*/
{
    Lista->Ultimo->Prox = (Apontador) malloc(sizeof(celula));
    Lista->Ultimo = Lista->Ultimo->Prox;
    Lista->Ultimo->Item = x;
    Lista->Ultimo->Prox = NULL;
    Lista->tamanho = Lista->tamanho + 1;
}
```

IMPLEMENTAÇÃO OPERAÇÕES BÁSICAS

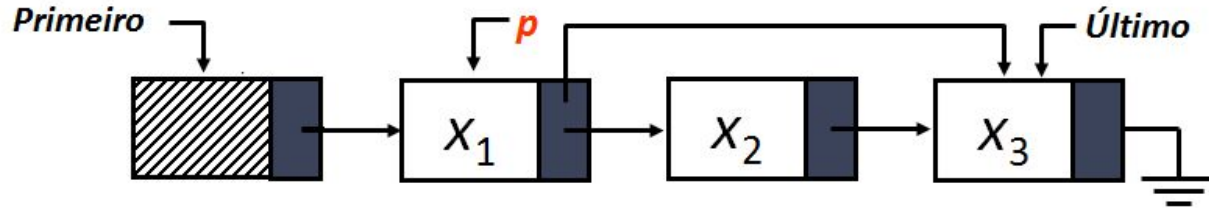


```
Lista->Ultimo->Prox = (Apontador) malloc(sizeof(TipoCelula));  
Lista->Ultimo = Lista->Ultimo->Prox;  
Lista->Ultimo->Item = x;  
Lista->Ultimo->Prox = NULL;
```

IMPLEMENTAÇÃO OPERAÇÕES BÁSICAS

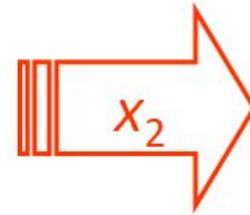
```
void Remove(Apontador p, TipoLista *Lista, TipoItem *Item)
{
    /* Obs.: o item a ser retirado é o seguinte ao apontado por p */
    Apontador pAux;
    if (Vazia(Lista) || p == NULL || p->Prox == NULL)
    {
        printf(" Erro Lista vazia ou posicao nao existe\n");
        return;
    }
    pAux = p->Prox;
    *Item = pAux->Item;
    p->Prox = pAux->Prox;
    if (p->Prox == NULL) Lista->Ultimo = p;
    free(pAux);
    Lista->tamanho = Lista->tamanho - 1;
}
```

IMPLEMENTAÇÃO OPERAÇÕES BÁSICAS



p : ponteiro para o nó anterior ao que será removido

```
pAux = p->Prox;  
*Item = pAux->Item;  
p->Prox = pAux->Prox;  
if (p->Prox == NULL) Lista->Ultimo = p;  
free(pAux);
```



IMPLEMENTAÇÃO OPERAÇÕES BÁSICAS

```
void Imprime(TipoLista Lista)
{
    Apontador Aux;
    Aux = Lista.Primeiro->Prox;
    while (Aux != NULL)
    {
        printf("%d\n", Aux->Item.chave);
        Aux = Aux->Prox;
    }
}
```

EXERCÍCIO

Seja uma lista de alunos da disciplina X

Registro para cada aluno:

Chave :1..999; {número de inscrição/matricula}

Nota :0..10; {média final}

Curso :1..20; {código do curso}

Problema: a partir da lista de alunos, gerar a lista dos alunos aprovados e reprovados na disciplina

Aprovado: Nota ≥ 7.0

Reprovado: caso contrário

Imprimir a lista dos aprovados e dos reprovados

REFERÊNCIAS

- ❏ Consultar ementário.