



Estruturas de Dados  
Aula 10: Listas (parte 2)

## Fontes Bibliográficas



- Livros:
  - Projeto de Algoritmos (Nivio Ziviani): **Capítulo 3;**
  - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): **Capítulo 10;**
  - Estruturas de Dados e seus Algoritmos (Szwarcfiter, et. al): **Capítulo 2;**
  - Algorithms in C (Sedgewick): **Capítulo 3;**
- Slides baseados nas transparências disponíveis em:  
<http://www.dcc.ufmg.br/algoritmos/transparencias.php>

### Listas com alocação não sequencial e dinâmica

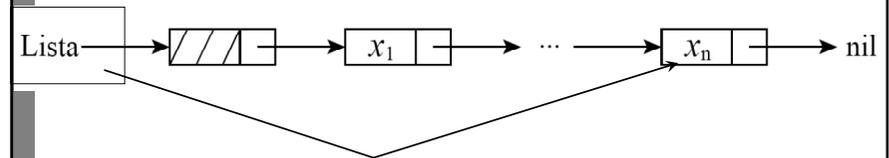


- Cada item é encadeado com o seguinte mediante uma variável do tipo Ponteiro.
- Permite utilizar posições não contíguas de memória.
- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.
- Há uma célula cabeça para simplificar as operações sobre a lista
- **Estrutura Encadeada**

### Listas com alocação não sequencial e dinâmica



- Cada item é encadeado com o seguinte mediante uma variável do tipo Ponteiro.
- Permite utilizar posições não contíguas de memória.
- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.



## Estrutura da Lista com Alocação não Sequencial e Dinâmica



- A lista é constituída de células.
- Cada célula contém um item da lista e um ponteiro para a célula seguinte.
- O registro (struct) TipoLista contém um ponteiro para a célula cabeça e um ponteiro para a última célula da lista.

## Estrutura da Lista com Alocação não Sequencial e Dinâmica (2) – lista.h



```
typedef int Posicao;  
typedef struct tipoitem TipoItem;  
typedef struct tipolista TipoLista;  
  
TipoLista* InicializaLista();  
int Vazia (TipoLista* Lista);  
void Insere (TipoItem* x, TipoLista* Lista);  
void Retira (TipoLista* Lista, int v);  
void Imprime (TipoLista* Lista);  
TipoItem* InicializaTipoItem();  
void ModificaValorItem (TipoItem* x, int valor);  
void ImprimeTipoItem(TipoItem* x);
```

## Estrutura da Lista com Alocação não Sequencial e Dinâmica (2) – arquivo.c



```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

struct tipoitem{
    int valor;
    /* outros componentes */
};

typedef struct celula_str Celula;

struct celula_str {
    TipoItem Item;
    Celula* Prox;
};

struct tipolista{
    Celula* Primeiro, Ultimo;
};
```

## Implementação TAD Lista com Ponteiros



```
TipoLista* InicializaLista()
{
    TipoLista* lista =
        (TipoLista*)malloc(sizeof(TipoLista));
    lista->Ultimo = NULL;
    lista->Primeiro = NULL;
    return lista;
}
```

## Implementação TAD Lista com Ponteiros



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

## Implementação TAD Lista com Ponteiros (2)



```
void Insere (TipoItem* x, TipoLista
    *Lista){
    Celula* novo = (Celula*)
    malloc(sizeof(Celula));
    if (lista->Ultimo == NULL)
        lista->Primeiro = lista->Ultimo =
        novo;
    else
    { Lista->Ultimo->Prox = novo;
      Lista->Ultimo = Lista->Ultimo->Prox;}
    Lista->Ultimo->Item =*x;
    Lista->Ultimo->Prox = NULL;
}
```

```

void Retira (TipoLista *Lista, int v)
{
    Celula* ant = NULL;
    Celula* p = Lista->Primeiro;
    while (p != NULL && p->Item.valor != v)
    { ant = p;
      p = p->Prox;}
    if (p == NULL)
        return;
    if (p == Lista->Primeiro && p == Lista->Ultimo){
        Lista->Primeiro = Lista->Ultimo = NULL;
        free (p);
        return; }
    if (p == Lista->Ultimo){
        Lista->Ultimo = ant; ant->Prox = NULL; free (p);
        return;}
    if (p == Lista->Primeiro)
        Lista->Primeiro = p->Prox;
    else
        ant->Prox = p->Prox;
    free(p);
}

```



## Implementação TAD Lista com Ponteiros(4)



```

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

```

Lista com alocação não sequencial e dinâmica:  
vantagens e desvantagens



- Vantagens:
  - Permite inserir ou retirar itens do meio da lista a um custo constante (importante quando a lista tem de ser mantida em ordem).
  - Bom para aplicações em que não existe previsão sobre o crescimento da lista (o tamanho máximo da lista não precisa ser definido *a priori*).
- Desvantagem: utilização de memória extra para armazenar os ponteiros.