



## Estruturas de Dados Aula 10: Listas (parte 2)

19/04/2010

## 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

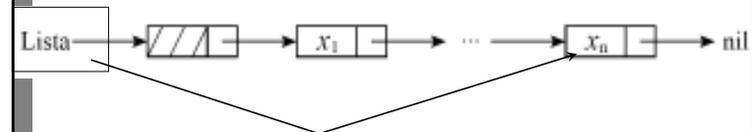


- Cada item é encadeado com o seguinte mediante uma variável do tipo Ponteiro.
- Permite utilizar posições não contíguas de memória.
- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.
- Há uma célula cabeça para simplificar as operações sobre a lista
- **Estrutura 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.
- Há uma célula cabeça para simplificar as operações sobre a 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();  
void FLVazia (TipoLista* Lista);  
int Vazia (TipoLista* Lista);  
void Insere (TipoItem* x, TipoLista* Lista);  
TipoItem* Retira (Posicao p, TipoLista* Lista);  
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));  
    return lista;  
}
```

## Implementação TAD Lista com Ponteiros



```
void FLVazia (TipoLista *Lista)
{
    Lista->Primeiro = (Celula*) malloc (sizeof
    (Celula));
    Lista->Ultimo = Lista->Primeiro;
    Lista->Primeiro->Prox = NULL;
}

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

## Implementação TAD Lista com Ponteiros (2)



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

## Implementação TAD Lista com Ponteiros (3)



```
// retorna o tamanho da lista, tirando célula
// cabeça
int tamanho (TipoLista* Lista)
{
    Celula* p;
    int i = 0;
    for (p=Lista->Primeiro->Prox; p!=NULL; p=p-
    >Prox)
        i++;
    return i;
}
```

## Implementação TAD Lista com Ponteiros (3)



```
/* O item retirado é o seguinte apontado por p */
TipoItem* Retira (Posicao p, TipoLista *Lista)
{
    Ponteiro q, q2; TipoItem* item;
    item = (TipoItem*)malloc(sizeof(TipoItem));
    int i;
    int t = tamanho (Lista);
    if (Vazia(Lista) || p>=t)
    { printf ("ERRO: Lista vazia ou posicao nao existe\n");
      return 0;
    }
    q = Lista->Primeiro;
    //encontra o elemento antes da posicao desejada
    for (i=0; i<p; i++)
        q = q->Prox;
    q2 = q->Prox;
    *item = q2->Item;
    q->Prox = q2->Prox;
    if (q2->Prox == NULL) Lista->Ultimo = q2;
    free (q2);
    return item;
}
```

## Implementação TAD Lista com Ponteiros(4)

```
void Imprime (TipoLista* Lista)
{
    Celula* Aux;
    Aux = Lista->Primeiro->Prox;
    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.