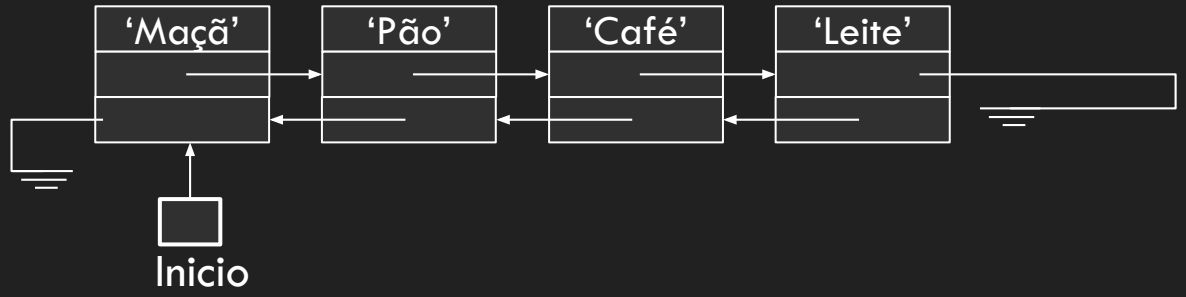
Listas Duplamente Ligadas

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Baseado nos slides do Prof. Rudinei Goularte

Listas Duplamente Ligadas

➔ Nesta aula vamos implementar as operações do TAD Listas utilizando Listas Duplamente Ligadas



Listas Duplamente Ligadas

- ➔ Nas listas duplamente ligadas, cada nó mantém um ponteiro para o nó anterior e posterior
- ➔ A manipulação da lista é mais complexa, porém algumas operações são diretamente beneficiadas
- ➔ Por exemplo, as operações de inserção e remoção em uma dada posição

Listas Duplamente Ligadas

- Aplicações
  - Em geral, qualquer aplicação que necessite “navegação” em dois sentidos.
  - Exemplo: “Playlist” (com respectiva reprodução) de músicas.
    - Skip e Back.

TAD Listas Duplamente Ligadas

- Principais operações
  - Criar lista
  - Apagar lista
  - Verificar se a lista está vazia
  - Imprimir lista
  - Inserir item
  - Remover item (dado uma chave)
  - Recuperar item (dado uma chave)

Interface (.h) da lista é a mesma!!!

Listas Duplamente Ligadas

```
/*listaDupla.c*/
typedef strcut no_ NO;
struct no_{
    ITEM *item;
    NO *anterior;
    NO *proximo;
};

struct lista_{
    NO *inicio;
    NO *fim;
    int tamanho;  /*tamanho da lista*/
};
```

Operações Básicas

➔ As operações de criar e apagar a lista são simples

```
LISTA *lista_criar(void){
/*pré-condição: existir espaço na memória.*/
    LISTA *lista = (LISTA *) malloc(sizeof(LISTA));
    if(lista != NULL) {
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho = 0;
    }
    return (lista);
}
```

Operações Básicas

- As operações de criar e apagar a lista são simples

```
/*recebe o inicio da lista como argumento e esvazia a mesma*/
void lista_esvazia (NO *ptr){
     if (ptr != NULL){
          if(ptr->proximo != NULL)
               lista_esvazia(ptr->proximo);
          item_apagar(&ptr->item);
          ptr->anterior = NULL;
          free(ptr); /* apaga o nó*/
          ptr = NULL;
     }
}
```

Operações Básicas

- As operações de criar e apagar a lista são simples

```
void lista_apagar(LISTA **ptr){
     if (*ptr == NULL)
          return;
     lista_esvazia((*ptr)->inicio);
     free(*ptr);
     *ptr = NULL;
}
```

Operações Básicas

- Inserção
  - Em listas não ordenadas
    - No início ou no fim da lista.
  - Em listas ordenadas
    - Inserir ordenadamente.
- Remoção
  - Em qualquer posição da lista

Operações Básicas

➔ Essas implementação são ligeiramente diferentes das implementações para listas simplesmente encadeadas.

Inserir Item (Primeira Posição)

```
/*Insere um novo nó no início da lista. PARA LISTAS NÃO ORDENADAS*/
boolean lista_inserir_inicio(LISTA *lista, ITEM *i){
     if ((lista != NULL) && (!lista_cheia(lista)) ) {
          NO *pnovo = (NO *) malloc(sizeof (NO));
          pnovo->item = i;
          if (lista->inicio == NULL){
               //lista->inicio = pnovo;
               lista->fim = pnovo;
               pnovo->proximo = NULL;
          }
          else {
               lista->inicio->anterior = pnovo;
               pnovo->proximo = lista->inicio;
          }
          pnovo->anterior = NULL;
          lista->inicio = pnovo;
          lista->tamanho++;
          return (TRUE);
     }
     return (FALSE);
}
```

Inserir Item (Última Posição)

```
/*Insere um novo nó no fim da lista. PARA LISTAS NÃO ORDENADAS*/
boolean lista_inserir_fim(LISTA *lista, ITEM *item){
     if ((lista != NULL) && (!lista_cheia(lista)) ) {
          NO *pnovo = (NO *) malloc(sizeof (NO));
          pnovo->item = item;
          if (lista->inicio == NULL){
               lista->inicio = pnovo;
               pnovo->anterior = NULL;
          }
          else {
               lista->fim->proximo = pnovo;
               pnovo->anterior = lista->fim;
          }
          pnovo->proximo = NULL;
          lista->fim = pnovo;
          lista->tamanho++;
          return (TRUE);
     }
     return (FALSE);
}
```

Remover Item (dado uma chave)

```
boolean lista_remover(LISTA *lista, int chave){
     NO *p=NULL;
     if ( (lista != NULL) && (!lista_vazia(lista)) ){
          p = lista->inicio;
          while(p != NULL && (item_chave(p->item) != chave) ) /*Percorre a lista em
               p = p->proximo;                              busca da chave*/
          if(p != NULL){ /*Se a lista não acabou significa que encontrou a chave*/
               if(p == lista->inicio) /*Se é o 1° da lista basta acertar o ptr inicio*/
                    lista->inicio = p->proximo;
               else  /*Se não é o 1° da lista, há alguém antes dele para acertar o ptr*/
                    p->anterior->proximo = p->proximo;
               if(p == lista->fim) /* Ideia do if/else anterior para o fim da lista */
                    lista->fim = p->anterior;
               else
                    p->proximo->anterior = p->anterior;
               p->proximo = NULL;  p->anterior = NULL;
               free(p); lista->tamanho--;
               return(TRUE);
          }
     }
     return(FALSE); /*elemento (chave) não está na lista ou lista vazia*/
}
```

Exercício

- Implementar a função para inserir **ordenadamente** itens no TAD lista dinâmica duplamente encadeada

Referências

- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.