LISTAS LINEARES

Lista Linear

- Definição: seqüência de zero ou mais elementos $a_1, a_2, ..., a_n$ sendo

  a_i elementos de um mesmo tipo

  n o tamanho da lista linear
- Propriedade fundamental: os elementos têm relações de ordem na lista

  a_i precede a_{i+1} (e a_i sucede a_{i-1});

  a_1 é o primeiro elemento da lista

  a_n é o último elemento da lista

Lista: Operações Usuais

- Para criar o TAD Lista, deve-se definir o conjunto de operações sobre os objetos do tipo Lista:
  1. Iniciar/Criar uma lista linear vazia
  2. Inserir um novo elemento após o *i*-ésimo elemento
  3. Retirar *i*-ésimo elemento
  4. Buscar o *i*-ésimo elemento
  5. Combinar duas ou mais listas em uma lista única
  6. Dividir uma lista em duas ou mais listas
  7. Fazer uma cópia da lista
  8. Ordenar os itens da lista em ordem crescente ou decrescente

Implementação de Listas

- Diferentes implementações do mesmo TAD Lista: elementos consecutivos (seqüencial) ou não consecutivos (encadeada)
- Representações mais utilizadas:
  1. Através de arranjos (*arrays*) – listas estáticas
  2. Através de apontadores ou ponteiros – listas dinâmicas
- Restrições nas operações → diferentes listas (pilhas, filas, deques, etc)
- Operações sobre listas dependem ainda da organização: ordenada ou desordenada.

TAD Lista: Operações Básicas

1. FLVazia

   parâmetros: TipoLista L;

   pós-condição: Lista L vazia;

   funcionalidade: cria uma lista vazia

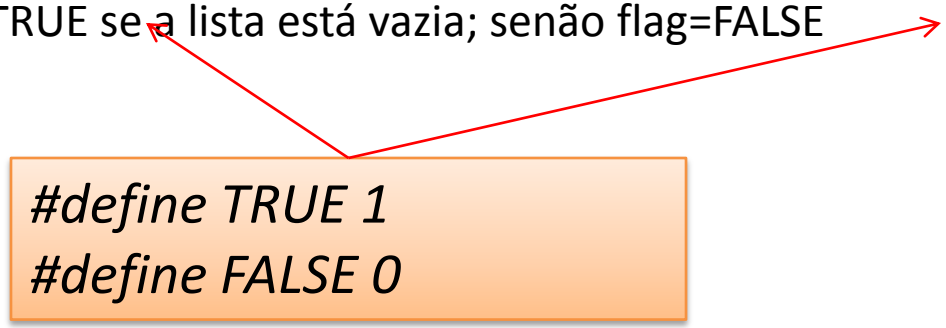
   resultado: retorna de uma lista vazia criada

2. Vazia

   parâmetros: TipoLista L, int flag;

   funcionalidade: Testa se a lista está vazia ou não

   resultado: flag=TRUE se a lista está vazia; senão flag=FALSE



TAD Lista: Operações Básicas

3. Retira

   parâmetros: TipoLista L, TipoItem x, int pos; int flag;

   pré-condição: Lista L tem n>1 elementos e pos <= posição do último;

   pós-condição: Lista L tem n – 1 elementos;

   funcionalidade: Retorna o item x que está na posição pos da lista mantendo a ordenação dos demais

   resultado: flag=TRUE se item removido com sucesso; senão flag=FALSE

4. Insere

   parâmetros: TipoItem x, TipoLista L, int flag;

   pré-condição: Lista L tem n>= 0 elementos e n <TamMax;

   pós-condição: Lista L tem n+1 elementos;

   funcionalidade: Insere o elemento x após o último elemento da lista

   resultado: flag=TRUE se item inserido com sucesso; senão flag=FALSE

TAD Lista: Operações Básicas

5. Imprime

   parâmetros: TipoLista L;

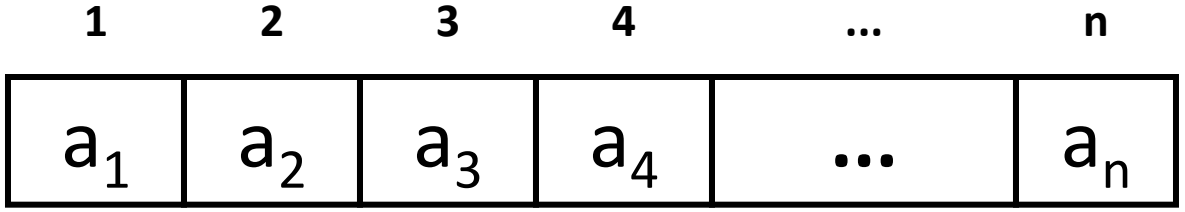
   funcionalidade: Imprime os itens da lista

   resultado: impressão dos itens da lista na ordem de ocorrência

- Outras ?

Listas Estáticas Seqüenciais

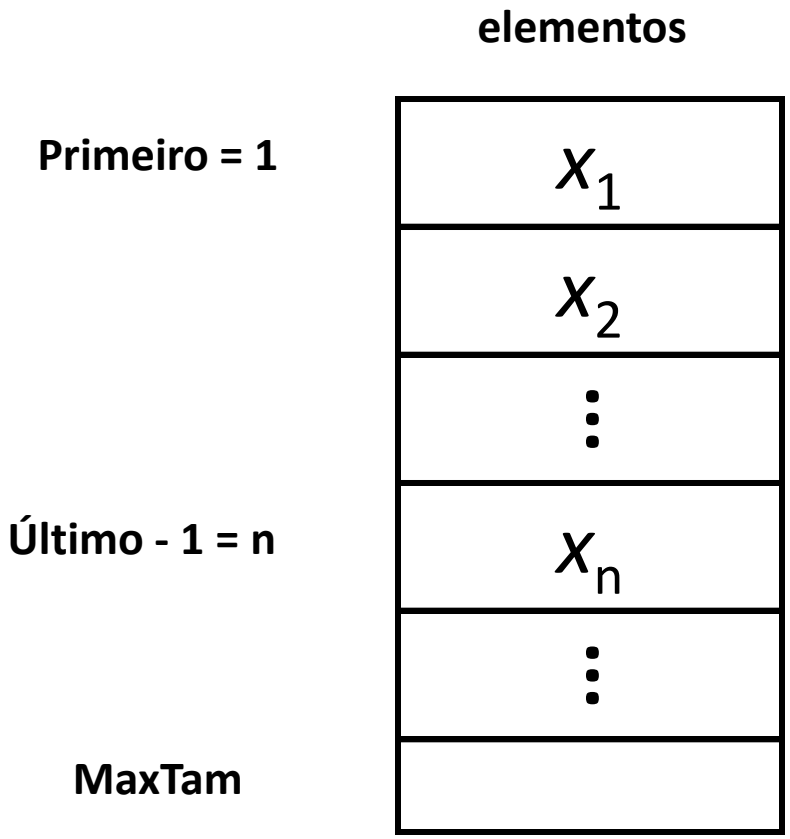
- Lista seqüencial: arranjo de registros de mesmo tipo na qual o sucessor de um elemento ocupa a posição física subseqüente a este elemento
- Implementação das operações em C pode ser feita usando ARRAY e STRUCT



Vantagens e Desvantagens

- Vantagens
  1. Acesso direto indexado a qualquer elemento (facilita modificações dos conteúdos)
  2. Tempo constante para acessar os elementos (depende só do índice)
- Desvantagens
  1. Movimentação para a inserção e remoção de elementos
  2. Conhecimento *a priori* do tamanho do número máximo da lista (aplicações em que não existe previsão sobre o crescimento da lista não é indicado)

Implementação da Lista Estática usando vetores



Implementação da Lista Estática usando vetores

```
#define InicioArranjo   1
#define MaxTam          1000

typedef int TipoChave;
typedef int Apontador;

typedef struct {
  TipoChave Chave;
  /* outros componentes */
} TipoItem;

typedef struct {
  TipoItem Item[MaxTam];
  Apontador Primeiro, Ultimo;
} TipoLista;
```

Implementação: FLVazia e Vazia

```
void FLVazia(TipoLista *Lista)
{
  Lista->Primeiro = InicioArranjo;
  Lista->Ultimo = Lista->Primeiro;
}  /* FLVazia */

int Vazia(TipoLista Lista)
{
  return (Lista.Primeiro == Lista.Ultimo);
}  /* Vazia */
```

Implementação: Insere

```
void Insere(TipoItem x, TipoLista *Lista)
{
  if (Lista->Ultimo > MaxTam)
  printf("Lista esta cheia\n");
  else { Lista->Item[Lista->Ultimo - 1] = x;
         Lista->Ultimo++;
       }
}  /* Insere */
```

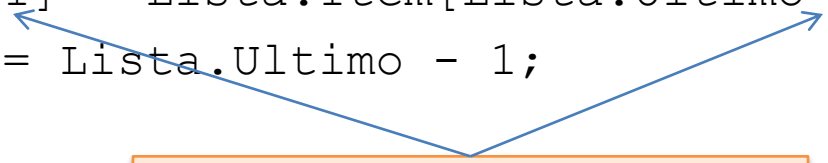
Implementação: Retira

```
void Retira(Apontador p, TipoLista *Lista, TipoItem *Item)
{
  int Aux;

  if (Vazia(*Lista) || p >= Lista->Ultimo)
  { printf(" Erro   Posicao nao existe\n");
    return;
  }
  *Item = Lista->Item[p - 1];
  Lista->Ultimo--;
  for (Aux = p; Aux < Lista->Ultimo; Aux++)
    Lista->Item[Aux - 1] = Lista->Item[Aux];
}  /* Retira */
```

Retira: Outra solução

```
{ dado que a lista é não ordenada, basta substituir o
   item retirado pelo ultimo da lista }
void Retira(Apontador p, TipoLista *Lista, TipoItem
   *Item) {
   int Aux;
     if(Vazia(*Lista)|| p >= Lista->Ultimo) {
       printf ("Erro: Elemento não existe\n");
       return;
     }
     *Item = Lista->Item[p-1];
     Lista.Item[p-1] = Lista.Item[Lista.Ultimo-2];
     Lista.Ultimo = Lista.Ultimo - 1;
}
```



Adaptação da posição absoluta na lista para a posição no array do C ([p])

Implementação: Imprime_Lista

```
void Imprime(TipoLista Lista)
{
  int Aux;

  for (Aux = Lista.Primeiro - 1; Aux <=
    (Lista.Ultimo - 2); Aux++)
    printf("%d\n", Lista.Item[Aux].Chave);
}  /* Imprime */
```

Listas Não-Seqüenciais ou Encadeadas

- Lista seqüencial: cada item da lista é encadeado com o seguinte através de uma variável do tipo Ponteiro