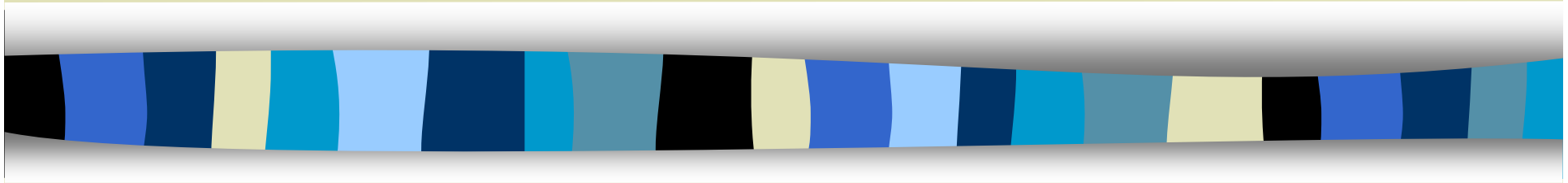
SCC122 - Estruturas de Dados



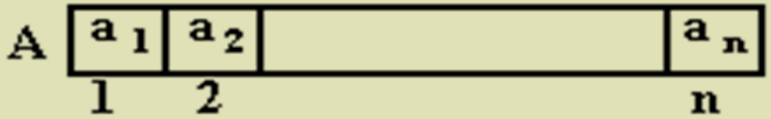
Lista Estática Seqüencial

Lista

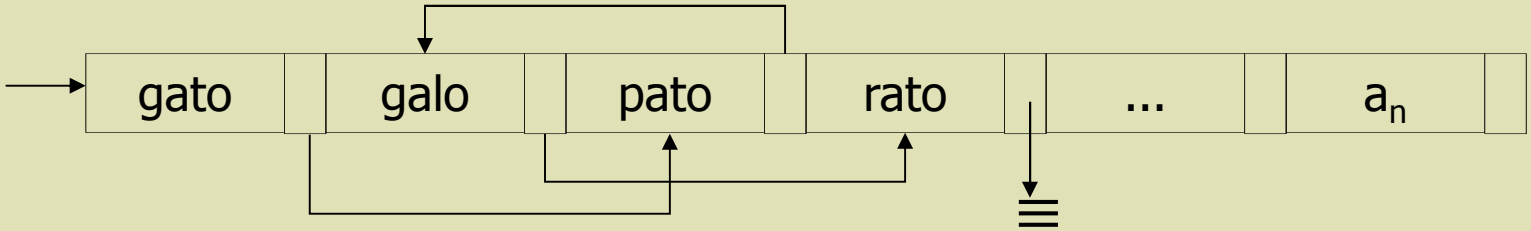
Uma lista é uma estrutura que armazena elementos de forma alinhada, ou seja, com elementos dispostos um após o outro.

Lista estática

- Implementação como array como seqüência de records;
  - Consecutiva – lista estática seqüencial.



  - Não consecutiva – lista estática encadeada.



- **Uma lista pode ser ordenada ou não.**

Lista dinâmica

C permite construir estruturas de dados avançadas – listas dinâmicas –, mais versáteis, utilizando **ponteiros** e **variáveis dinâmicas**.

Lista

Importante:

Os tipos de listas mencionados anteriormente são implementações diversas do mesmo tipo abstrato de dado, a lista.

Lista

As propriedades estruturadas da lista permitem responder a questões como:

- Qual é o primeiro elemento da lista
- Qual é o último elemento da lista
- Quais elementos sucedem um determinado elemento
- Quantos elementos existem na lista
- Inserir um elemento na lista
- Eliminar um elemento da lista

Lista Estática Seqüencial

- **Vantagens**:
  - Acesso direto indexado a qualquer elemento da lista
  - Tempo constante para acessar o elemento i - dependerá somente do índice.
- **Desvantagem**:
  - Movimentação quando eliminado/inserido elemento
  - Tamanho máximo pré-estimado
- **Quando usar:**
  - Listas pequenas
  - Inserção/remoção no fim da lista
  - Tamanho máximo bem definido

Operações do TAD Lista

```
#define  max 100;
typedef int posicao;
typedef struct tipo_elem{
                    chave: T1;
                    info: T2;
                    }
struct Lista{
  int nelem;
  tipo_elem A[max];
};
```

```
posicao Fim(const Lista *L){
/* Retorna a posição após o último  elemento de uma
   lista*/
 return ( L->nelem );
};

int Lista_vazia(const Lista *L){
/* Retorna true se lista vazia, false caso
   contrário*/
 return (L->nelem == 0);
};

int Lista_cheia(const Lista *L){
/* Retorna true se lista cheia, false caso
   contrário*/
   return ( L->nelem >= max );
};
```

Operações do TAD Lista

```
void Definir (Lista *L){
/* Cria uma lista vazia. Este procedimento deve ser
   chamado para cada nova lista antes da execução de
   qualquer operação.*/
    L->nelem = 0;
};

void Apagar (Lista *L){
/*Apaga uma lista.*/
    L->nelem = 0;
};
```

```
int Inserir(tipo_elem *x, posicao p, Lista *L){
/* Insere novo elemento na posição p da Lista. Se L
   = a1, a2,... an então temos a1, a2,...ap-1 x ap+1
   ... an. Se p = Fim(L) temos a1, a2,...an,x.
   Devolve 1 (true) se sucesso, 0 (false) caso
   contrário (L não tem nenhuma posição p ou Lista
   cheia)*/
posicao q;
if (Lista_cheia(L)) return ( 0 );
else if ((p > Fim(L)) || (p < 0))
     return ( 0 );//posição não existe
     else {
          for (q=L->nelem; q > p; q--)
            L->A[q] = L->A[q-1];
          L->nelem = L->nelem + 1;
          L->A[p] = *x;
          return ( 1 );
         }
};(Nelem – p+1) Movimentos
```

```
int Inserir_ord(tipo_elem *x, Lista *L){
/*Insere novo elemento de forma a manter a Lista
   ordenada. Devolve true se sucesso, false caso
   contrário */
 posicao i;
 /*acha posição de inserção*/
 i = 0;
 while ((i < L->nelem) && (x->chave > L->A[i].chave))
  i++;
 return ( Inserir(x,i,L) );
};
```

```
posicao Busca_Linear (tipo_elem *x, const Lista *L){
/*Retorna a posição de x na Lista. Se x ocorre mais de uma
   vez, a posição da primeira ocorrência é retornada. Se x
   não aparece retorna Fim(L)*/
/* Primeira implementação com busca linear simples*/
 int i; //posicao
 i = 0;
 while ((i < L->nelem) && (x->chave != L->A[i].chave))
   i++;
 return ( i );
};
```

```
posicao Busca_Sentinela (tipo_elem *x, Lista *L){
/*Retorna a posição de x na Lista. Se x ocorre mais de uma vez, a
   posição da primeira ocorrência é retornada. Se x não aparece
   retorna Fim(L)*/
/* Segunda implementação com busca linear e sentinela: insere x no
   final da lista, como sempre encontrará podemos eliminar o teste
   de fim de lista*/
  int i; //posição
  L->A[Fim(L)].chave = x->chave; //sentinela
  i = 0;
  while (x->chave != L->A[i].chave)
    i++;
  return ( i );
}
```

Nro Max de Comparações: Nelem + 1

Obs.: Faça uma adaptação no algoritmo para que ele funcione mesmo com a lista estando cheia.

```
posicao Busca_binaria (tipo_elem *x, const Lista *L){
/*Retorna a posição de x na Lista. Se x não aparece retorna Fim(L)*/
/* o vetor deve estar ordenado, neste caso em ordem crescente.*/
 posicao inf, sup, meio;
 if ( ! Lista_vazia(L) )
 {
  inf = 0; sup = L->nelem - 1;
  while (inf <= sup)
  {
    meio = (inf + sup) / 2;
    if (L->A[meio].chave == x->chave)
    {
     return ( meio );//sai da busca
    }
    else if (L->A[meio].chave < x->chave)
          inf = meio + 1;
         else sup = meio - 1;
  } //while
 }//if
 return( Fim(L) );
}; //function
```

```
void Ler_registro (posicao p, const Lista *L, tipo_elem *Reg){
/* Recupera o elemento da posição p da Lista L. Se p = Fim(L) ou não
   existe posição p válida emite mensagem.*/
 if ((p >= Fim(L)) || (p < 0))
   printf("posição inexistente/n");
 else *Reg = L->A[p]; //se posição for válida
};

int Remover (posicao p, Lista *L){
/* Remove o elemento na posição p da Lista. Se L = a1,a2,...an então
   temos a1, a2, ...ap-1, ap+1,... an. Devolve true se sucesso, false
   caso contrário (L não tem nenhuma posição p ou se p = Fim(L))*/
posicao i;
if ( ( p >= Fim(L) ) || ( p < 0 ) ) return ( 0 );
else if (Lista_vazia(L)) return ( 0 );
     else {
            for (i = p+1; i < L->nelem; i++)
              L->A[i-1] = L->A[i];
            L->nelem = L->nelem - 1;
            return ( 1 );
          }
};
```

Nro de Mov = (nelem – p)

```
posicao Prox(const posicao p, Lista *L){
/*Retorna p + 1. Se p é a última posição de L então
   Prox(p,L) = Fim(L). Prox(p,L) retorna 0 se p = Fim(L)*/
  if (p == L->nelem) return ( Fim(L) );
  else return ( p + 1 );
};

posicao Ant(const posicao p, Lista *L){
/* Retorna p - 1. Ant(p,L) retorna -1 se p < 0*/
  if (p == 0) return ( -1 );
  else return ( p - 1 );
};

posicao Primeiro(const Lista *L){
/* Retorna 0. Se L , vazia retorna -1*/
  if (L->nelem == 0 ) return ( -1 );
  else return ( 0 );
};
```

```
void Imprimir (const Lista *L){
/* Imprime os elementos na sua ordem de precedência.*/
 posicao i;
 if (! Lista_vazia(L) )
  for (i = 0; i < L->nelem; i++)
     printf("%d - %s\n",L->A[i].chave,L->A[i].info);
};

int Tamanho (const Lista *L){
/* Retorna o tamanho da Lista. Se L , vazia retorna 0*/
  return ( L->nelem );
};
```

Lista de Exercícios

1) Verifique se L está ordenada (pode ser crescente ou decrescente)
2) Faça uma cópia de Lista L1 em outra L2
3) Faça uma cópia da Lista L1 em L2 , eliminando repetidos
4) Inverta L1 colocando o resultado em L2
5) Inverta a própria L1
6) Intercale L1 com L2 gerando L3, considere L1 e L2 ordenadas
7) Elimine de L1 todas as ocorrências de um dado elemento, L1 está ordenada

- Este material didático foi revisado e adaptado por Danilo Medeiros Eler e pelo prof. Adenilso da Silva Simão a partir do material da profa. Roseli Ap. Francelin Romero.
- Revisado pela profa. Roseli no 2o./2009