Estrutura de Dados

- Listas Lineares
- Listas Encadeadas
- Pilhas
- Filas

Danielle B. Colturato

Introdução

- **Estrutura de Dados** é um método particular de se implementar um **TAD**.
- A **implementação** de um **TAD** escolhe uma **ED** para representá-lo.
- Cada **ED** é construída dos tipos primitivos (inteiro, real, char,...) ou dos tipos compostos (array, registro,...) de uma linguagem de programação.

Exemplos de TAD

- **Lineares:**
  - Listas Ordenadas
  - Pilhas
  - Filas
  - Deques
- **Não Lineares:**
  - Árvores
  - Grafos

Listas

- São estruturas formadas por um conjunto de dados de forma a preservar a relação de ordem linear entre eles.
- Uma lista é composta por nós, os quais podem conter, cada um deles, um dado primitivo ou composto.
- Representação:

  - L_1 -> 1º elemento da lista
  - L_2 -> Sucessor de L_1
  - L_{n-1} -> Antecessor de L_n
  - L_n -> Último elemento da lista.

Exemplos de Lista:

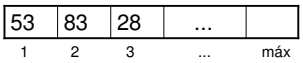
- Lista Telefônica
- Lista de clientes de uma agência bancária
- Lista de setores de disco a serem acessados por um sistema operacional
- Lista de pacotes a serem transmitidos em um nó de uma rede de computação de pacotes.

Operações Realizadas com Listas

- Criar uma lista vazia
- Verificar se uma lista está vazia
- Obter o tamanho da uma lista
- Obter/modificar o valor do elemento de uma determinada posição na lista
- Obter a posição de elemento cujo valor é dado
- Inserir um novo elemento após (ou antes) de uma determinada posição na lista
- Remover um elemento de uma determinada posição na lista
- Exibir os elementos de uma lista
- Concatenar duas listas

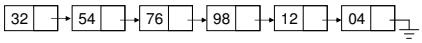
Formas de Representação

- **Seqüencial:**
  - Explora a sequencialidade da memória do computador;
  - Os nós de uma lista são armazenados em endereços seqüenciais, ou igualmente distanciados um do outro;
  - Pode ser representado por um vetor na memória principal ou um arquivo seqüencial em disco.



Formas de Representação

- **Encadeada:**
  - Esta estrutura é tida como uma seqüência de elementos encadeados por ponteiros
  - Cada elemento deve conter, além do dado propriamente dito, uma referência para o próximo elemento da lista.

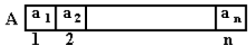


Lista Seqüencial

- Uma lista representada de forma seqüencial é um conjunto de registros onde estão estabelecidas regras de precedência entre seus elementos. O sucessor de um elemento ocupa posição física subsequente.
- A implementação de operações pode ser feita utilizando *array* e *registro*, associando o elemento a_i com o índice i (mapeamento seqüencial).

Lista Seqüencial - Características

- os elementos na lista estão armazenados fisicamente em posições consecutivas;
- a inserção de um elemento na posição a_i causa o deslocamento a direita do elemento de a_i ao último;
- a eliminação do elemento a_i requer o deslocamento à esquerda do a_{i+1} ao último;
- a_1 é o primeiro elemento, a_i precede a_{i+1}, e a_n é o último elemento.



Listas Seqüenciais

- A propriedades estruturadas da lista permitem responder a questões tais como:
  - se uma lista está vazia
  - se uma lista está cheia
  - quantos elementos existem na lista
  - qual é o elemento de uma determinada posição
  - qual a posição de um determinado elemento
  - inserir um elemento na lista
  - eliminar um elemento da lista

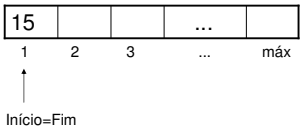
Listas Seqüenciais

- **Vantagem:**
  - 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

Inserção no Fim da Lista

Lista:Vetor[1..10] de inteiros;

Insere_lista(L; Fim; 15);



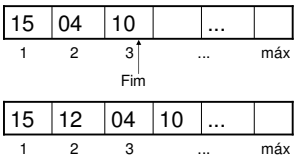

```
Constante Máx=10
Tipo Vetor=Array[1..Máx] de inteiros

Função Insere_lista(V:Lista;
Fim:inteiro; dado:inteiro): boleana;
Início
    se Fim=Máx
        então Insere_lista<-falso
        senão
            Fim<-Fim+1
            V[Fim]<-dado
            Insere_lista<-verdadeiro
Fim
```

Inserção em uma dada posição

Lista:Vetor[1..10] de inteiros;

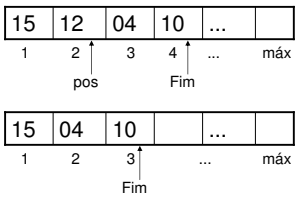
Inserir(L; Fim; 2; 12);




```
Função Inserir (L:Lista; Fim:inteiro;
    pos:inteiro; dado:inteiro):boleano;
Início
    i:inteiro
    se ( (Fim = Máx) ou (pos > Máx) )
        então Inserir<-falso
        senão
        para  i=Fim+1 até i=pos faça
            //decrescente
            L[i] <- L[i-1]
        L[i] <- dado
        Fim<-Fim+1
        Inserir<-verdadeiro
Fim
```

Remoção de um elemento da lista

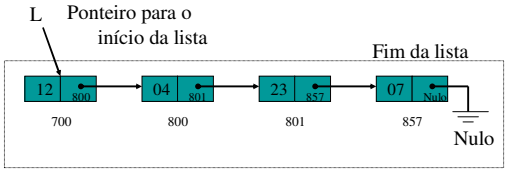
Remover(L; Fim; 2);




```
Função Remover (L:Lista; Fim:inteiro;
    pos:inteiro):boleano;
Início
    se ( pos > Fim ou pos <=0) então Remover<-falso
        senão se pos=Fim
                então Fim<-Fim-1
                    Remover<-verdadeiro
                senão
                    para i=pos até i=Fim faça
                        L[i]<- L[i+1]
                    Fim<-Fim-1
                    Remover<-verdadeiro
Fim
```

Listas Ligadas (Encadeadas)

- Os elementos da lista são registros com um dos componentes destinado a guardar o endereço do registro sucessor



Variáveis Dinâmicas e Ponteiros

- As linguagens de programação modernas tornaram possível explicitar não apenas o acesso aos dados, mas também aos endereços desses dados.
- Isso significa que ficou possível a utilização de ponteiros explicitamente, implicando que uma distinção notacional deve existir entre os dados e as referências (endereços) desses dados.
- A linguagem C utiliza a seguinte notação para a manipulação de ponteiros:

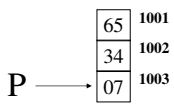
  tp *P
- Essa declaração expressa que a variável **P** é um ponteiro para uma variável do tipo **tp**.

Ponteiros

- Um ponteiro é uma variável cujo valor é um endereço de memória do computador, e cujo valor está armazenado neste endereço.
- A linguagem Pascal utiliza a seguinte notação para a manipulação de ponteiros:

var ponteiro: ^ **tipo**;

ex: var P: ^integer;



Ponteiros

- Alocação dinâmica de memória
  - as variáveis declaradas de um programa têm o seu endereço fixado durante a compilação;
  - as variáveis declaradas de uma subrotina têm o seu endereço fixado durante sua ativação;
  - os ponteiros permitem, durante a execução do programa, criar e desativar outras variáveis, denominadas *dinâmicas*.

Ponteiros - Alocação dinâmica de memória

Pascal		C	
New(p);	aloca uma área de memória e guarda seu endereço em p	malloc(n)	aloca dinamicamente n bytes e devolve um ponteiro para o início da memória alocada
Dispose(p);	libera a área de memória cujo endereço está em p	free(p)	libera a região de memória apontada por p
p	o endereço	p	o endereço
p^	o valor armazenado na posição de memória de endereço p	*p	o valor armazenado na posição de memória de endereço p
NILL	ponteiro que não aponta para nada (NULO)	NULL	ponteiro que não aponta para nada (NULO)
@A	endereço de uma variável	&A	endereço de uma variável

Ponteiros

```
#include <stdio.h>
main()
{
      int a, *pa;
      double b, *pb;
      char c, *pc;

     // atribuições de endereços
      pa = &a; pb = &b; pc = &c;

     // atribuição de valores
      a = 1; b = 2.34; c = '@';
      printf("\n valores:%5d  %5.2lf  %c", a, b, c);
      printf("\n ponteiros:%5d  %5.2lf  %c", *pa, *pb, *pc);
      printf("\n enderecos:%p  %p  %p", pa, pb, pc);

     // mais atribuições de valores usando os ponteiros
      *pa = 77; *pb = 0.33; *pc = '#';
      printf("\n valores :%5d  %5.2lf  %c", a, b, c);
      printf("\n ponteiros:%5d  %5.2lf  %c", *pa, *pb, *pc);
      printf("\n enderecos:%p  %p  %p", pa, pb, pc);
}
```

```
Resultados:
valores:   1   2.34  @
 ponteiros:   1   2.34  @
 enderecos:0063FDDC  0063FDD4  0063FDD3
 valores :  77   0.33  #
 ponteiros:  77   0.33  #
 enderecos:0063FDDC  0063FDD4  0063FDD3
```

Aloca dinamicamente um inteiro e depois o libera

```
#include <stdlib.h>
  int *pi;
  pi = (int *) malloc (sizeof(int));
  ...
  free(pi);
```

Alocação Dinâmica de Vetores

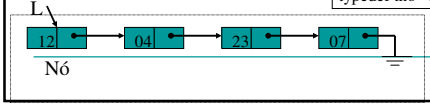
```
#include <stdlib.h>
main()
{
     int *v;
     int i, n;
     scanf("%d", &n); // le n
     // aloca n elementos para v
     v = (int *) malloc(n*sizeof(int));
     // zera o vetor v com n elementos
     for (i = 0; i < n; i++) v[i] = 0;
     ...
     // libera os n elementos de v
     free(v);
}
```

Listas Encadeadas – Definição da lista (nós)

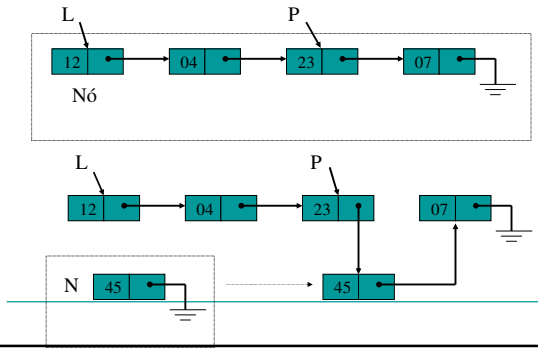
```
Tipo     No = registro
                elemento: inteiro;
                prox: campo do ponteiro
                para o próximo nó;
         end;
Variável L: ponteiro para uma
            variável do tipo No;
```

```
Pascal
Type     No = record
                elemento: integer;
                prox: ^No;
         end;
Var      L: ^No;
```

```
C
typedef int telem;
typedef struct no {
                telem dado;
                struct no* prox;
} tno;
typedef tno* tlista;
```



Inserção de um novo nó N na posição apontada por P

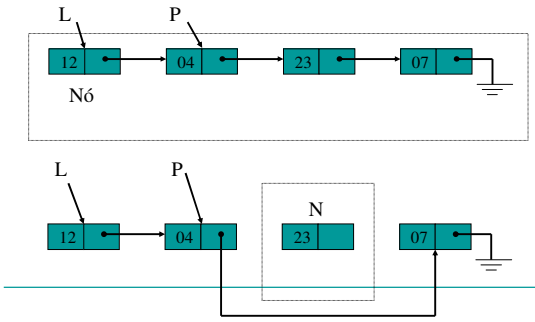


Inserção de um novo nó N na posição apontada por P

- Inserção
  - Guarda o endereço do próximo nó;
  - O nó que está sendo apontado por P aponta para N.

Pascal	C
New(N); N^.elemento:=45; N^.prox:=NIL;	N = (tlista) malloc(sizeof(tno)); N->dado = 45; N->prox =NULL;
N^.prox := P^.prox; P^.prox := N;	N->prox = p->prox; p->prox = N;

Remoção de um nó logo após a posição apontada por P



Remoção de um nó

- Remoção
  - Guarda o endereço do que será removido em um ponteiro auxiliar N;
  - Fazer o nó apontar para o que o N aponta;
  - Liberar N.

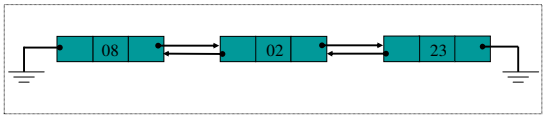
Pascal	C

N := P^.prox;
P^.prox := N^.prox;
Dispose (N);
	N=p->prox;
	p->prox = N->prox;
	free(N);

Listas Duplamente Encadeadas

- **Características**
  - Listas foram percorridas do início ao final.
  - Ponteiro "anterior" necessário para muitas operações.
  - Em alguns casos pode-se desejar percorrer uma lista nas duas direções indiferentemente.
  - Nestes casos, o gasto de memória imposto por um novo campo de ponteiro pode ser justificado pela economia em não reprocessar a lista toda.

Listas Duplamente Encadeadas



Pascal

```
Type tpont = ^ trec;
     trec = record
               info:T;
               esq, dir: tpont;
End;
     Lista = tpont;
Var pont: Lista;
```

Listas Duplamente Encadeadas

Inserção à direita de pont

```
Procedure ins_dir (pont: lista; x: T);
Var   j: Lista;

Begin

        new(j);
        j^.info:=x;
        j^.dir:=pont^.dir;
        j^.dir^.esq:=j;
        j^.esq:=pont;
        pont^.dir:=j;
End;
```

Inserção à esquerda de pont

```
Procedure ins_esq (pont: Lista; x: T);
Var j: Lista;
Begin
        new(j);
        j^.info:=x;
        j^.dir:=pont;
        j^.esq:=pont^.esq;
        j^.esq^.dir:=j;
        pont^.esq:=j;
End;
```

Listas Duplamente Encadeadas

Eliminação à direita de pont

```
Procedure elim_dir (pont: Lista);
Var j: Lista;
Begin
        j:=pont^.dir;
        pont^.dir:=j^.dir;
        j^.dir^.esq:=pont;
        dispose(j);
End;
```

Eliminação do próprio pont

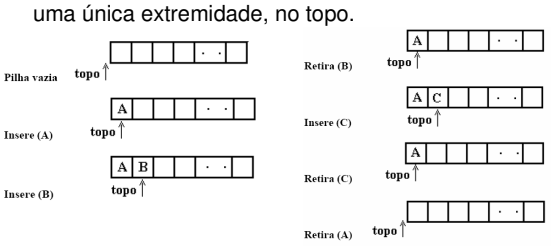
```
Procedure elim (Var pont: Lista);

Begin
        pont^.dir^.esq:=pont^.esq;
        pont^.esq^.dir:=pont^.dir;
        dispose(pont);

End;
```

Pilhas

- **Pilhas** são listas onde a inserção de um novo item ou a remoção de um item já existente se dá em uma única extremidade, no topo.



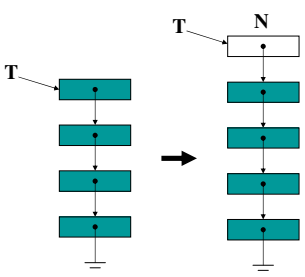
Pilha – Definição e operações associadas

- Dada uma pilha P = (a_1, a_2, ..., a_n), dizemos que a_1 é o elemento da pilha; e a_{i+1} está acima de a_i.
- Pilhas são também conhecidas como listas **LIFO** (*Last In, First Out*).
- **Operações Associadas:**
  1. Criar uma pilha P vazia
  2. Testar se P está vazia
  3. Obter o elemento do topo da pilha (sem eliminar)
  4. Inserir um novo elemento no topo de P (empilhar - PUSH)
  5. Remover o elemento do topo de P (desempilhar - POP)

Implementação de Pilhas

- Como lista **Seqüencial** ou **Encadeada**?
  - No caso geral de listas ordenadas, a maior vantagem da alocação encadeada sobre a seqüencial - se a memória não for problema - é a eliminação de deslocamentos na inserção ou eliminação dos elementos.
  - No caso das pilhas, essas operações de deslocamento não ocorrem.
  - Portanto, podemos dizer que a alocação sequencial é mais vantajosa na maioria das vezes.

Pilha - Inserção



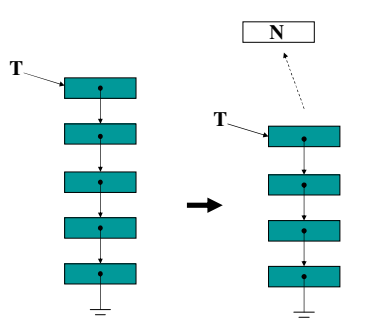
Pascal

```
N^.dado:=valor;
N^.prox := T;
T := N;
```

C

```
N->dado = valor;
N->prox = T;
T = N;
```

Pilha - Remoção



Pascal

```
N := T;
T := N^.prox;
Dispose (N);
```

C

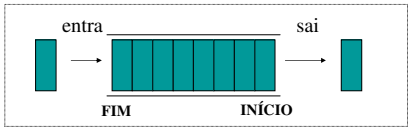
```
N = T;
T = N->prox;
free(N);
```

Filas

- É uma lista linear em que a inserção é feita numa extremidade e a eliminação na outra.
  - inserção: no fim da fila
  - deleção: primeiro da fila
- Conhecida com estrutura FIFO (*First In, First Out*).

$(a_1, a_2, \ldots, a_n)$

eliminações no início — inserções no final



Filas - Exemplos

- Escalonamento de "Jobs": fila de processos aguardando os recursos do sistema operacional.
- Fila de pacotes a serem transmitidos numa rede de comutação de pacotes.
- Simulação: fila de caixa em banco.

Filas - Operações associadas:

1. **Criar** - cria uma fila vazia
2. **Vazia** - testa se um fila está vazia
3. **Primeiro** - obtém o elemento do início de uma fila
4. **Inserir** - insere um elemento no fim de uma fila
5. **Remover** - remove o elemento do início de uma fila, retornando o elemento removido.

Implementação de Filas

- Como lista **Seqüencial** ou **Encadeada** ?
  - Pelas suas características, as filas têm as eliminações feitas no seu início e as inserções feitas no seu final.
  - A implementação **encadeada dinâmica** torna mais simples as operações (usando uma lista de duas cabeças).
  - Já a implementação **seqüencial** é um pouco mais complexa (teremos que usar o conceito de fila circular), mas pode ser usada quando há previsão do tamanho máximo da fila.

Problema na implementação seqüencial

- O que acontece com a fila considerando a seguinte seqüência de operações sobre um fila:

I E I E I E I E ...
(I - inserção e E - eliminação)

- Note que a fila vai se deslocando da esquerda para a direita do vetor. Chegará a condição de "overflow" (cheia), porém estando vazia, ou seja, sem nenhum elemento.
- **Alternativa:**
  - No algoritmo de remoção, após a atualização de **inicio**, verificar se a fila ficou vazia. Se este for o caso, reinicializar

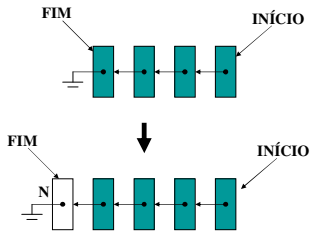
inicio = 0 e **final = -1**

Problema na implementação seqüencial

- **O que aconteceria se a seqüência fosse:**

  I I E I E I E I E I ...
- A lista estaria com no máximo dois elementos, mas ainda ocorreria overflow com a lista quase vazia.
- **Alternativa:**
  - Forçar **final** a usar o espaço liberado por **inicio** (**Fila Circular**)

Fila – Inserção de um elemento



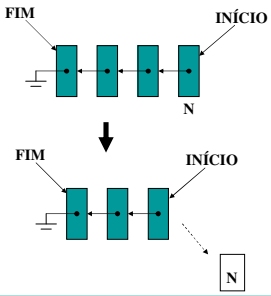
Pascal

```
FIM^.prox:=N;
N^.prox:=NIL;
FIM:=N;
```

C

```
FIM->prox=N;
N->prox = NULL;
FIM=N;
```

Fila – Remoção de um elemento



Pascal

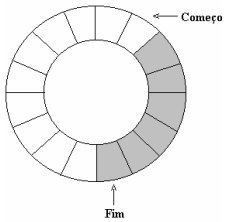
```
N:=INICIO;
INICIO:=N^.prox;
Dispose(N);
```

C

```
N=INICIO;
INICIO=N->prox;
Free(N);
```

Fila Circular

- Para permitir a reutilização das posições já ocupadas, usa-se o conceito de "Fila Circular".



Fila Circular

- Índices do array: *0 .. m-1*
- Ponteiros de controle: *Começo* e *Fim*
- Inicialmente *Começo = Fim = 0*
- Quando *Fim = m-1*, o próximo elemento será inserido na posição 0 (se essa posição estiver vazia!)
- Condição de fila vazia *Começo = Fim*

Fila Circular - Inserção

```
Procedure Inserir (Var Fim: indice);
Begin
  If Fim = m-1
     Then    Fim := 0
          Else    Fim := Fim + 1;
                  {ou  Fim := ( Fim + 1 ) mod m}
End;
```

Fila Circular - Remoção

```
Procedure Eliminar (Var Começo: indice);
Begin
   Começo := (Começo + 1) mod m;
End;
```

- **Problema:** Nesta representação, como teremos fila cheia ???

  Começo = Fim

- Para resolver esse problema, utilizaremos apenas m-1 posições do anel. Deste modo, existe sempre um elemento vazio e, portanto, *Fim* não coincide com *Começo*.

Fila Circular - Inserção

```
Procedure Inserir (Var Fim: indice; Começo: indice; F:
   fila);
Begin
   If (Fim + 1) mod m = Começo
        Then    {FILA CHEIA}
              Else    Begin
                      Fim := (Fim + 1) mod m;
                      {um registro fica vazio}
                      F[Fim] := x;
                      End;
End;
```

Fila Circular - Remoção

```
Procedure Eliminar (Var Comeco: indice; Fim:
   indice);
Begin
   If Começo = Fim
        Then    {FILA VAZIA}
        Else    Começo := (Começo + 1) mod m;
End;
```