

Estruturas de dados com alocação dinâmica de memória

Estrutura de Dados e Operações - Lista Dinâmica

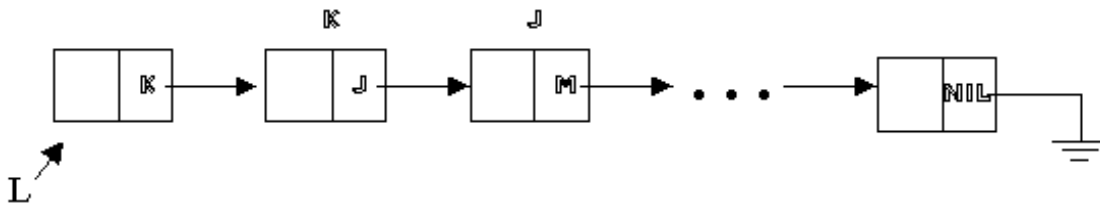
Instituto de Ciências Matemáticas de São Carlos
Departamento de Computação e Estatística
SCE182 - Algoritmos e Estruturas de Dados 1
Prof. Resp.: Graça Pimentel, Maria Cristina e Rosane

Definição da ED

```
Type prec = ^rec;  
Lista = prec;  
rec = record  
    info: T;      {seu tipo preferido}  
    lig: Lista  
End;  
  
Var p: Lista;    {ponteiro para qualquer elemento da lista}  
    L: Lista;    {ponteiro para o primeiro elemento da lista }
```

Visualização de uma lista encadeada

k, j e m são posições de memória não consecutivas e L é o ponteiro para o início da lista



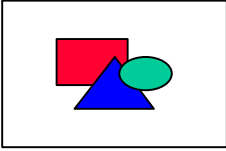
Operações sobre Listas Dinâmicas

1. [Criar lista vazia](#)
2. [Inserir primeiro elemento](#)
3. [Inserir no início de uma lista](#)
4. [Acessar primeiro elemento](#)
5. [Acessar último elemento](#)
6. [Tamanho da lista](#)
7. [Inserir valor v na posição pos](#)
8. [Eliminar elemento da posição pos](#)
9. [Eliminar primeiro elemento](#)
10. [Eliminar valor v](#)
11. [Inserir valor v antes do elemento apontado por p](#)
12. [Criar uma lista com registros numerados](#)
13. [Eliminar sucessor de p](#)
14. [Imprimir recursivamente](#)

Estruturas de dados com alocação dinâmica de memória

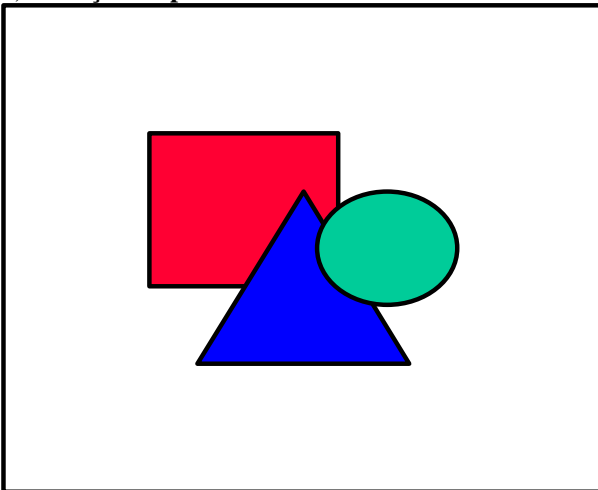
Implementação das Operações

1) Criação da lista vazia



```
Procedure Criar (Var L : Lista);  
Begin  
  L := nil;  
End;
```

2) Inserção do primeiro elemento



```
Procedure Insere_Prim(Var L: Lista; valor: T);  
Var p: Lista;  
Begin  
  new(p);  
  p^.info := valor;  
  p^.lig := nil;  
  L := p;  
End;
```

3) Inserção no início de uma lista

```
Procedure Insere_Inic(Var L: Lista; valor: T);  
Var p: Lista;  
Begin  
  new(p);  
  p^.info := valor;  
  p^.lig := L;  
  L := p;  
End;
```

Estruturas de dados com alocação dinâmica de memória

4) Acesso ao primeiro elemento da lista

```
Function Prim(L: Lista): T;  
Begin  
  Prim:= L^.info  
End;
```

5) Acesso ao último elemento da lista

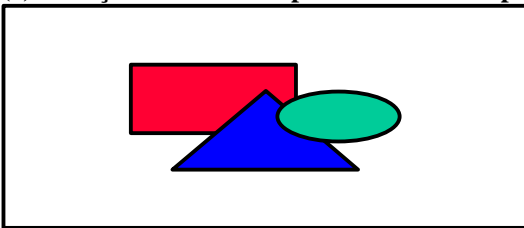
```
Function Ultimo(L: Lista): T;  
Begin  
  If L = nil Then  
    { lista vazia}  
  Else  
    Begin  
      p := L;  
      While p^.lig < nil do  
        p := p^.lig;  
      Ultimo := p^.info;      {p^.info é último elemento}  
    End;                      {p^ é o último registro}  
  End;  
End;
```

6) Qual o número de elementos da lista ?

```
Function Nelem(L: Lista): integer;  
Begin  
  If L = nil Then  
    Nelem := 0  
  Else  
    Begin  
      p := L;  
      Nelem := 1;  
      While p^.lig < nil do  
        Begin  
          Nelem := Nelem + 1;  
          p := p^.lig;  
        End;  
      End;  
    End;  
  End;  
End;
```

7)

(a) Inserção do valor v depois do elemento apontado por k



```
Procedure Insere_Depois(v: T; k: Lista);  
Var q: Lista;  
Begin  
  new(q);  
  q^.info := v;  
  q^.lig := k^.lig;  
  k^.lig := q;  
End;
```

OBS: Funciona para inserção após último elemento.

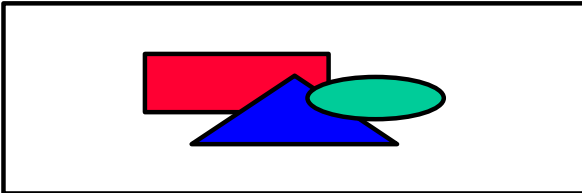
Estruturas de dados com alocação dinâmica de memória

(b) Inserção do valor v na posição pos

```
Function Insere_Pos (VAR L:Lista; v:T; pos:integer);
begin
  if L=nil then
    Insere_Prim (L,v)
  else
    if pos = 1 then
      Insere_Inic (L,v)
    else
      begin
        m = pos-1;
        p: = L;
        c=1;
        while (c<m) and (p^.lig < nil) do
          begin
            p:= p^.lig;
            c := c+1;
          end;
        if c=m then {p é ponteiro para o anterior}
          Insere_Deois (v,p)
        else
          writeln("nao existe posição pos na lista" )
      end;
    end;
  end;
```

8)

(a) Remoção do elemento apontado por j, que segue k



```
Procedure Elimina_Deois(k, j: Lista);
Begin
  k^.lig := j^.lig;
  dispose(j);
End;
```

(b) Remoção do elemento da posição pos

```
Function Remove_pos (VAR L:Lista; pos:integer);
begin
  if pos = 1 then
    Remove_Prim (L) {ver operação número 9}
  else
    begin
      m = pos-1;
      p: = L;
      c=1;
      while (c<=m) and (p^.lig < nil) do
        begin
          pa := p;
          p:= p^.lig;
          c := c+1;
        end;
    end;
  end;
```

Estruturas de dados com alocação dinâmica de memória

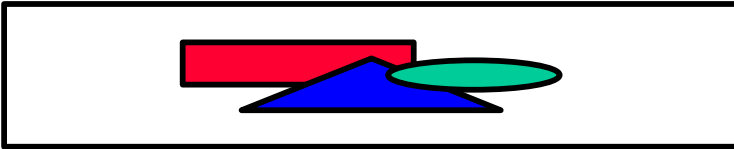
```
end;  
if (c=pos) and (p<nil) then {pa é ponteiro para o anterior}  
  Elimina_depois (pa,p)  
else  
  writeln("nao existe posição pos na lista" )  
end;  
end;  
end;
```

9) Remoção do primeiro elemento

```
Procedure Remove_Prim(Var L: Lista);  
Begin  
  p := L;  
  L:= L^.lig;  
  dispose(p)  
End;
```

OBS: funciona no caso de remoção em lista com um único elemento.

10) Eliminar um valor v de uma lista ordenada L



```
Procedure Elimina(v: T; Var L: Lista);  
Var  
  p, pa: Lista;  
  acabou: boolean;  
  
Begin  
  pa := nil;  
  p := L;  
  acabou := (p = nil);  
  While (not acabou) do  
    If p^.info < v Then  
      Begin  
        pa := p;  
        p := p^.lig;  
        acabou := (p = nil);  
      End  
    Else acabou := true;  
    {While acaba aqui!}  
    { p = nil ou p^.info = v ou p^.info > v }  
    If (p = nil) Then  
      "lista não contém v"  
    Else  
      If p^.info > v Then  
        Writeln("lista não contém v")  
      Else  
        Begin  
          If pa = nil Then  
            Remove_prim(L)  
          Else Elimina_depois(pa,p);  
        End;  
      End;  
  End;  
End;
```

Estruturas de dados com alocação dinâmica de memória

11) Inserção do valor v antes do elemento apontado por p

Uma alternativa para evitarmos percorrer a lista para encontrar o antecessor de **p** é inserirmos o valor dado, de fato, após o elemento apontado por **p**, após realizar uma cópia de conteúdo entre ponteiros....

```
Procedure Insere_antes(v: T; p: Lista);
Var q: Lista;
Begin
  new(q);
  q^ := p^;
  p^.info := v;
  p^.lig := q;
End;
```

12) Criação de uma lista L com registros numerados de 1 .. n

```
Procedure Cria_Lista_num(Var L: Lista);
  Var L, q: Lista;

  Begin
    L := nil; { começa com lista vazia }
    While n > 0 do
      Begin
        new(q);
        q^.lig := L;
        q^.info := n;
        L := q;
        n := n-1;
      End;
    End;
```

13) Remoção do registro sucessor de p e inserção do mesmo como cabeça em uma lista apontada por q

```
Procedure Remove_Insere(Var q: Lista; p: Lista);
  Begin
    r := p^.next; {r aponta o sucessor de p^}
    p^.lig := r^.lig; {r^ sai da lista}
    r^.lig := q; {r^ passa a apontar q^}
    q := r; {r^ passa a ser o primeiro elemento da lista}
  End; {apontada por q}
```

14) Impressão recursiva de uma lista

```
Procedure Printlist(p: Lista);
  Begin
    If (p < nil) Then
      Begin
        write(p^.info);
        Printlist(p^.lig);
      End;
  End;
```

Estruturas de dados com alocação dinâmica de memória

Busca em Lista Dinâmica

1) Problema com nil ?

Se estivermos buscando um elemento x em uma lista encadeada, podemos ter algo similar a:

```
while (p <> nil) and (p^.info <> x) do
  p := p^.lig;
```

Que **problema** temos no comando acima ?

No caso de $p = nil$ o comando pode ser inválido pois $p^.info$ não existe!

OBS.: Se o compilador não realiza o segundo teste no caso do primeiro falhar, não haveria problema. Em nível de algoritmo, é melhor resolver o problema evitando a possibilidade desse tipo de erro ocorrer, ao invés de usar soluções que dependem do compilador.

O que fizemos nos nossos algoritmos foi algo similar a:

```
acabou := false;
While (p <> nil) and (not acabou) do
  If p^. info = x Then
    acabou := true
  Else p := p^.lig;
```

2) Uso de sentinela na busca em lista encadeada

Um elemento [sentinela](#) pode ser utilizado na busca em lista linear encadeada: usamos para isso um registro no final da lista.

Supondo que inicializamos uma lista apontada por L com a lista vazia, faremos:

```
Procedure Create(L: Lista);
Var sentinela: Lista;
Begin
  new(sentinela);
  L := sentinela;
End;
```

O algoritmo seguinte implementa uma busca por um elemento x na lista apontada por L **não ordenada**.

Retorna $p = nil$ se o elemento não for encontrado; caso contrário p aponta o registro que contém o elemento.

```
Procedure busca(x: T; var L, p: Lista);
  var p: Lista;

Begin
  p := L;
  sentinela^.info := x;
  While p^.info <> x do
    p := p^.lig;
  If (p = sentinela) Then
    p := nil; {elemento não encontrado}
End;
```

Estruturas de dados com alocação dinâmica de memória

3) Busca com sentinela e inserção na cabeça da lista não ordenada

Caso o elemento não seja encontrado e quisermos inseri-lo no final da lista, teremos o procedimento abaixo:

```
Procedure busca_insere(x: T; Var L, p: Lista);
  var p: Lista;

Begin
  p := L;
  sentinela^.info := x;
  While p^.info <> x do
    p := p^.lig;
  If (p = sentinela) Then
    Begin
      { inserção no começo da lista}
      p := L;           {aponta primeiro}
      new(L);          {novo registro é primeiro}
      L^.info := x;
      L^.lig := p;
    End;
  End;
```

4) Sentinela em busca com lista encadeada ordenada

No algoritmo acima, ao realizar uma busca sequencial numa lista encadeada anulamos a vantagem de utilizar listas encadeadas. Neste caso, esta implementação só pode ser considerada para listas com um número pequeno de elementos.

No caso da lista estar **ordenada**, a busca termina assim que encontrarmos um elemento maior que aquele que buscamos. A ordenação é obtida ao inserirmos cada novo elemento na sua posição correta, ao invés de na cabeça da lista. Essa "ordenação" na realidade não tem custo algum, porque fazemos uso das características da lista encadeada. Isso não ocorre com estruturas estáticas como o *array* e os *arquivos sequenciais*.

O procedimento abaixo **elimina** um registro que contém o elemento *v* de uma lista apontada por *L*. O bloco de busca pelo elemento realiza dois testes para cada elemento da lista. Esse esforço pode ser reduzido introduzindo o recurso de **sentinela**. Depois da **busca**, testes devem ser efetuados para verificar se: o elemento foi encontrado ou não se encontrado, se ele era o primeiro da lista ou não

```
Procedure elimina(v: integer; Var Prim: Lista);

Var
  p, pa: Lista;
  acabou: boolean;

Begin
  pa := nil;
  p := L;
  acabou := (p = nil);
  While (not acabou) do
    If p^.info = v }
  If (p = nil) Then
    "lista não contém v"
  Else
    If p^.info = v Then
      "lista não contém v"
    Else
```

Estruturas de dados com alocação dinâmica de memória

```
Begin
  If pa = nil Then
    L := p^.lig
  Else
    pa^.lig := p^.lig;
    dispose(p);
  End;
End;
```

Dada a seguinte **inicialização**:

```
new(root);
new(sentinel);
root^.lig := sentinel;
```

Temos acima um lista com dois elementos: o elemento apontado por root vai ser sempre um [dummy](#) (elemento sem conteúdo), e o sentinela sempre marca o final da lista.

O algoritmo abaixo realiza a **busca** de um elemento e o **insere** na posição correta caso ele não seja encontrado.

```
Procedure busca_insere(x: T; Var root: Lista);
  Var w1, w2, w3: Lista;

Begin
  w2 := root;
  w1 := w2^.lig;
  sentinel^.info := x;
  While w1^.info <> sentinel) Then
    {elemento já existe}
  Else
    Begin
      new(w3); {insere w3 entre w1 e w2}
      w3^.info := x;
      w3^.lig := w1;
      w2^.lig := w3;
    End;
  End;
```

Estruturas de dados com alocação dinâmica de memória

Lista Duplamente Encadeada

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.
-



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

```
Var pont: Lista;
```

- Como consequência, podemos realizar as operações de inserção e eliminação à esquerda ou à direita de um campo no interior de uma lista sem a necessidade de ponteiros "anteriores".

Implementação de algumas operações de lista duplamente encadeada com alocação dinâmica

As operações 1 a 4 abaixo são atômicas, isto é, elas realizam a reserva de espaço (quando necessária) além do 'acerto dos ponteiros'. Pré-condição para todas elas é que se conheça o ponteiro de um nó ou de um vizinho. Além disso, elas não tratam casos especiais. Assim, elas são operações de suporte para as inserções e eliminações do TAD Lista, quando este é implementado por lista duplamente encadeada. Tais operações não ficariam disponíveis para o 'usuário' da TAD. Apenas o projetista do tipo abstrato de dados as usa no desenvolvimento dos algoritmos genéricos de inserção e eliminação (isto é, aqueles que tratam todos os casos, além das condições de erro).

1) 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;
```

2) Inserção à esquerda de pont

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

Estruturas de dados com alocação dinâmica de memória

```
j^.dir:=pont;  
j^.esq:=pont^.esq;  
j^.esq^.dir:=j;  
pont^.esq:=j;  
End;
```

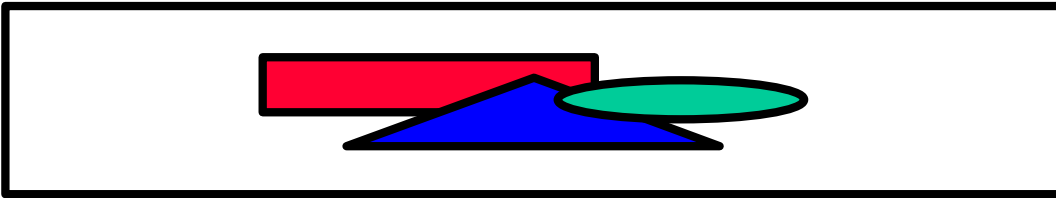
3) Eliminação à direita de pont

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

4) Eliminação do próprio pont

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

5) Busca em uma lista circular



```
Function Busca_Dup_Ord(ptlista: Lista; x: T):Lista;  
{Lista Duplamente Encadeada Ordenada com sentinela apontado por ptlista}  
  
Var pont, ultimo: Lista;  
Begin  
  ultimo:=ptlista^.esq;  
  If x<=ultimo^.info then  
    Begin  
      pont:=ptlista;  
      While pont^.info < x do  
        pont:=pont^.dir;  
      Busca_Dup_Ord:=pont;  
    End  
  Else  
    Busca_Dup_Ord:=ptlista;  
End;
```