

{ Operações elementares em Listas Ligadas Lineares }

```
type ponteiro    = ^elemento;  
      elemento = record  num : integer;  
                      prox : ponteiro  
                      end;  
var ponta : ponteiro;
```

- * Registrar os números contidos numa Lista referida por ponta no ficheiro numeros.txt, um por linha e pela mesma ordem.

{ Versão Iterativa }

```
procedure registrar(ponta : ponteiro; var A : text);  
var p : ponteiro;  
  
begin  assign(A, 'numeros.txt');  
      rewrite(A);  
  
      p:= ponta;  
      while p <> nil do  
          begin writeln(A, p^.num);  
                p:= p^.prox  
          end;  
  
      close(A)  
  
end{ registrar };
```

{ Versão Recorrente – encapsulamento }

```
procedure registar(ponta : ponteiro; var A : text);

    procedure listar(p : ponteiro);
        begin if p <> nil
            then begin writeln(A, p^.num);
                    listar(p^.prox)
            end;
        end{ listar };

begin assign(A, 'numeros.txt');
    rewrite(A);
    listar(ponta);
    close(A)
end{ registar };
```

✳ Registar por ordem inversa.

```
procedure registarR(ponta : ponteiro; var A : text);

    procedure listarR(p : ponteiro);
        begin if p <> nil
            then begin listarR(p^.prox);
                    writeln(A, p^.num)
            end;
        end{ listarR };

begin assign(A, 'numeros.txt');
    rewrite(A);
    listarR(ponta);
    close(A)
end{ registarR };
```



Construir uma Lista cujos nós contêm os números inteiros registados no ficheiro `numeros.txt`, por ordem inversa.

```
procedure recuperarR(var ponta : ponteiro; var A : text);  
var p : ponteiro;  
    n : integer;  
  
begin assign(A, 'numeros.txt');  
    reset(A);  
  
    ponta:= nil;  
    while not eof(A) do  
        begin readln(A, n);  
            new(p);  
            p^.prox:= ponta;  
            p^.num:= n;  
            ponta:= p  
        end;  
  
    close(A)  
  
end{ recuperarR };
```

Recuperar pela mesma ordem.

```
procedure recuperar(var ponta : ponteiro; var A : text);  
var p, fim : ponteiro;  
    n : integer;  
  
begin assign(A, 'numeros.txt');  
    reset(A);  
  
    if eof(A)  
    then ponta:= nil  
    else begin readln(A, n);  
                new(ponta);  
                ponta^.num:= n;  
                ponta^.prox:= nil;  
                fim:= ponta;  
  
                while not eof(A) do  
                    begin readln(A, n);  
                        new(p);  
                        p^.num:= n;  
                        p^.prox:= nil;  
                        fim^.prox:= p;  
                        fim:= p  
                    end  
  
                end;  
  
    close(A)  
  
end{ recuperar };
```

Outra versão, utilizando um nó provisório:

```
procedure recuperar2(var ponta : ponteiro; var A : text);  
var p, fim, provisório : ponteiro;  
    n : integer;  
  
begin assign(A, 'numeros.txt');  
    reset(A);  
  
    new(provisório);  
    provisório ^.prox:= nil;  
    fim:= provisório;  
  
    while not eof(A) do  
        begin readln(A, n);  
            new(p);  
            p ^.num:= n;  
            p ^.prox:= nil;  
            fim ^.prox:= p;  
            fim:= p  
        end;  
  
    ponta:= provisório ^.prox;  
    dispose(provisório);  
  
    close(A)  
  
end{ recuperar2 };
```



Pesquisa do elemento maximo, identificado pelos ponteiros pmax e pantmax.

```
procedure DarMax( ponta : ponteiro; var pmax, pantmax : ponteiro) ;  
  
var maximo : integer;  
    este, ant : ponteiro;  
  
begin if ponta = nil  
    then begin pmax:= nil;  
              pantmax:= nil  
    end  
    else begin maximo:= ponta^.num;  
              pantmax:= nil;  
              pmax:= ponta;  
              ant:= nil;  
              este:= ponta^.prox;  
  
              while este <> nil do  
                  begin if este^.num > maximo  
                      then begin maximo:= este^.num;  
                                pantmax:= ant;  
                                pmax:= este;  
                      end;  
                      ant:= este;  
                      este:= este^.prox  
                  end  
    end  
  
end; { DarMax }
```

*** Remover o nó que contém o elemento maximo, numa só passagem.**

```
procedure RemoverMax(var ponta : ponteiro) ;

var maximo : integer;
    este, ant, pmax, pantmax : ponteiro;

begin   if ponta <> nil
        then begin maximo:= ponta^.num;
                    pantmax:= nil;
                    pmax:= ponta;
                    ant:= nil;
                    este:= ponta^.prox;

                    while este<>nil do
                        begin   if este^.num > maximo
                                then begin maximo:= este^.num;
                                            pantmax:= ant;
                                            pmax:= este
                                end;
                                ant:= este;
                                este:= este^.prox
                        end;

                    if ponta = pmax
                    then ponta:= ponta^.prox
                    else pantmax^.prox:= pmax^.prox;

                    dispose(pmax)

        end

end; { RemoverMax }
```



Remover e devolver o nó com o elemento maximo.

```
procedure RemoverDarMax(var ponta, pmax : ponteiro) ;  
  
var maximo : integer;  
    este, ant, pantmax : ponteiro;  
  
begin    if ponta = nil  
        then pmax:= nil  
        else begin maximo:= ponta^.num;  
                    pantmax:= nil;  
                    pmax:= ponta;  
                    ant:= ponta;  
                    este:= ponta^.prox;  
  
                    while este <>nil do  
                        begin if este^.num > maximo  
                            then begin maximo:= este^.num;  
                                        pantmax:= ant;  
                                        pmax:= este  
                            end;  
                            ant:= este;  
                            este:= este^.prox;  
                        end;  
  
                        if ponta = pmax  
                        then ponta:= ponta^.prox  
                        else pantmax^.prox:= pmax^.prox;  
  
                        pmax^.prox:= nil  
                    end  
  
end; { RemoverDarMax }
```



Utilizando RemoveDarMax ordenar a Lista, sem criação de espaço de memória.

```
procedure OrdenarLista (var ponta : ponteiro);  
var pmax, novaponta : ponteiro;  
  
begin while ponta <> nil do  
    begin pmax:= nil;  
        RemoveDarMax(ponta, pmax);  
        if novaponta = nil  
        then novaponta:= pmax  
        else begin pmax^.prox:= novaponta;  
                novaponta:= pmax  
        end  
    end;  
  
    ponta:= novaponta  
  
end; { OrdenarLista }
```



A partir de uma Lista Ordenada, criar outra sem nós repetidos.

```
function SemRep(ponta : ponteiro) : ponteiro;
var p, aux, fim, novo : ponteiro;

begin if ponta = nil
    then SemRep:= nil
    else begin new(aux);
                fim:= aux;

                p:= ponta;
                while p <> nil do
                    begin if p^.num <> fim^.num
                        else begin new(novo);
                                    novo^.num:= p^.num;
                                    novo^.prox:= nil;
                                    fim^.prox:= novo;
                                    fim:= novo
                                end;
                                p:= p^.prox
                            end;

                    SemRep:= aux^.prox;
                    dispose(aux);
                end;

end; { SemRep }
```



Remover as repetições de uma dada Lista Ordenada.

✳ Fusão de duas Listas Ordenadas.

```
function Fusao (pA, pB : ponteiro) : ponteiro;
var fim, p, aux : ponteiro;

begin if (pA = nil) and (pB = nil)
    then Fusao:= nil
    else   begin new(aux);
            fim:= aux;
            while (pA <> nil) and (pB <> nil) do
                begin   if pA^.num < pB^.num
                    then begin new(p);
                                p^.num:= pA^.num;
                                p^.prox:= nil;
                                fim^.prox:= p;
                                fim:= p;
                                pA:= pA^.prox;
                            end
                    else begin new(p);
                                p^.num:= pB^.num;
                                p^.prox:= nil;
                                fim^.prox:= p;
                                fim:= p;
                                pB:= pB^.prox;
                            end
                end
            end;

            if pA = nil
            then while pB <> nil do
                begin   new(p);
                        p^.num:= pB^.num;
                        p^.prox:= nil;
                        fim^.prox:= p;
                        fim:= p;
                        pB:= pB^.prox;
                end
            end
```

```
        else while pA <> nil do
            begin
                new(p);
                p^.num:= pA^.num;
                p^.prox:= nil;
                fim^.prox:= p;
                fim:= p;
                pA:= pA^.prox;
            end;
        Fusao:= aux^.prox;
        dispose(aux);
    end;

end; { Fusao }
```

União de Conjuntos.

```
function Unir(pA,pB:ponteiro):ponteiro;
var fim,p,aux:ponteiro;

begin if (pA=nil) and (pB=nil)
    then Unir:=nil
    else   begin new(aux);
           fim:=aux;

           while (pA<>nil) and (pB<>nil) do
               begin if pA^.num<pB^.num
                   then   begin new(p);
                           p^.num:=pA^.num;
                           p^.prox:=nil;
                           fim^.prox:=p;
                           fim:=p;
                           pA:=pA^.prox;
                   end
                   else if pA^.num>pB^.num
                       then begin new(p);
                               p^.num:=pB^.num;
                               p^.prox:=nil;
                               fim^.prox:=p;
                               fim:=p;
                               pB:=pB^.prox;
                       end
                   else { pA^.num=pB^.num }
                       begin   new(p);
                               p^.num:=pA^.num;
                               p^.prox:=nil;
                               fim^.prox:=p;
                               fim:=p;
                               pA:=pA^.prox;
                               pB:=pB^.prox;
                       end;
               end;
           end;
    end;
```

```
if pA=nil
then while pB<>nil do
begin new(p);
      p^.num:=pB^.num;
      p^.prox:=nil;
      fim^.prox:=p;
      fim:=p;
      pB:=pB^.prox;
end
else while pA <> nil do
begin new(p);
      p^.num:=pA^.num;
      p^.prox:=nil;
      fim^.prox:=p;
      fim:=p;
      pA:=pA^.prox;
end;

Unir:=aux^.prox;
dispose(aux);

end;
```

end; { Unir – faz a fusão ordenada, sem repetições, de duas listas previamente ordenadas e sem repetições – União de Conjuntos }

* Intersecção de Conjuntos.

```
function Intersectar(pA,pB:ponteiro):ponteiro;
var fim,p,aux:ponteiro;
```

```
begin if (pA=nil) or (pB=nil)
    then Intersectar := nil
    else begin new(aux);
        fim:=aux;
        while (pA<>nil) and (pB<>nil) do
            begin if pA^.num<pB^.num
                then pA:=pA^.prox
                else if pA^.num>pB^.num
                    then pB:=pB^.prox
                    else { pA^.num=pB^.num }
                        begin new(p);
                            p^.num:=pA^.num;
                            p^.prox:=nil;
                            fim^.prox:=p;
                            fim:=p;
                            pA:=pA^.prox;
                            pB:=pB^.prox;
                        end;
            end;

        Intersectar:=aux^.prox;
        dispose(aux);
    end;
```

end; { Intersectar – devolve uma lista ordenada cujos elementos são os elementos comuns a duas listas previamente ordenadas e sem repetições – Intersecção de conjuntos }