AUTORE:PAOLO BIONDO

NOTA DELL'AUTORE:

Sebbene la maggiorparte del codice presente in queste pagine sia stato provato e verificato (ahimè quando ho studiato Programmazione e Laboratorio ;)) non è certo esente da errori. Molte di queste procedure o funzioni non saranno le più performanti che esistano, ma in genere vanno bene per qualsiasi programma. Buono studio a tutti!!!

Dichiarazioni:

```
type puntatore=^elemento

   elemento=record
    info: integer;       //spesso anche char
    next: puntatore;
end;
```

Spiegazione: *Legg*end*o alternativamente gli elementi da 2 liste, ne viene creata una terza.*

CODICE:

```
Procedure ListaAlternata(lis,lis2: puntatore; var lis3:puntatore);
begin
if (lis=NIL) OR (lis2=NIL) then lis3:=NIL
else
   begin
   new(lis3);
   lis3^.info:=lis^.info;
   new(lis3^.next);
   lis3^.next^.info:=lis2^.info;
   ListaAlternata(lis^.next,lis2^.next,lis3^.next^.next);
   end;
end;
```

Spiegazione: *Cancellazione della prima occorrenza di un elemento in una lista.*

CODICE:

```
Procedure CancellaElementoLista (var p_testa: puntatore; valore:integer);
{ Cancella la prima occorrenza dell'elemento valore dalla lista}
var
  prec,                { puntatore all'elemento precedente }
  corr    : puntatore; { puntatore all'elemento corrente }
  trovato : boolean;   { booleano usato per terminare la scansione }
begin
write('Inserisci il valore da eliminare: ');
readln(valore);

  if p_testa <> NIL then
    if p_testa^.info=valore then
    begin { cancellazione del primo elemento }
      prec := p_testa;
      p_testa := p_testa^.next;
      dispose(prec)
    end
    else
    begin { scansione della lista e cancellazione dell'elemento }
      prec := p_testa;
```

```
      corr := prec^.next;
      trovato := FALSE;
      while (corr <> NIL) and (not trovato) do
        if corr^.info= valore then
        begin { cancellazione dell'lemento }
          trovato := TRUE;            { forza l'uscita dal ciclo }
          prec^.next := corr^.next;
          dispose(corr)
        end
        else
        begin { avanzamento dei due puntatori }
          prec := prec^.next;
          corr := prec^.next
        end
    end
end; { CancellaElementoLista }
```

Spiegazione: *Cancellazione di tutte le occorrenze di un elemento tranne la prima.*

CODICE:

```
Procedure CancOccorrenze(var lis1:puntatore; valore:integer);
(*Se i=1 significa che è stata incontrata la prima occorrenza e quindi non si
 può procedere all'eliminazione.Quando sarà trovata nuovamente una occorrenza
 i<>1 (ad esempio i=2 alla seconda occorrenza) si può quindi procedere all'
 eliminazione*)
var
  paux:puntatore;
begin
if lis1<>NIL then
 begin
 if lis1^.info=valore then
  begin
  i:=i+1;
  if i<>1 then
    begin
    paux:=lis1;
    lis1:=lis1^.next;
    dispose(paux);
    CancOccorrenze(lis1,valore); //si richiama la procedura per eventuali altre
occ.
    end
  else  //se i=1
    CancOccorrenze(lis1^.next,valore);
  end
 else //se lis1^.info<>valore
    CancOccorrenze(lis1^.next,valore);
 end;
end;
```

Spiegazione: *Cancellazione di tutte le vocali da una lista che ha per elementi lettere dell'alfabeto.*

CODICE:

```
Procedure CancellaTutteVocali(var p_testa:puntatore);
var
```

```
  paux: puntatore;
begin
if p_testa<>NIL then
  { Si può fare così:
  if (p_testa^.info='a')OR(p_testa^.info='e')OR(p_testa^.info='i')OR
      (p_testa^.info='o')OR(p_testa^.info='u') then....Oppure}
  if p_testa^.info in ['a','e','i','o','u'] then
      begin
      paux:=p_testa;
      p_testa:=p_testa^.next;
      dispose(paux);
      CancellaTutteVocali(p_testa);
      end
   else CancellaTutteVocali(p_testa^.next);
end;
```

Spiegazione: *Cancellazione dell'ultima occorrenza di un elemento.*

CODICE:

```
Procedure CancUltimaOccorrenza(var lis1:puntatore; valore:integer);
var
  paux,prec:puntatore;
  i,j:integer;
begin
i:=0;
j:=0;
paux:=lis1;
//Controlliamo quante sono le occorrenze del valore nella lista
while (paux<>NIL) do
 begin
 if paux^.info=valore then
  begin
  i:=i+1;
  paux:=paux^.next
  end
 else paux:=paux^.next;
 end;
//Creazione Record generatore
new(prec);
prec^.next:=lis1;
lis1:=prec;
//Cerchiamo l'ultima occorrenza e la eliminiamo
while (prec^.next<>NIL) do
  begin
  if prec^.next^.info=valore then
   begin
    j:=j+1;
   if j=i then
    begin
    paux:=prec^.next;
    prec^.next:=paux^.next;
    dispose(paux);
    end
   else //if j<>i
     prec:=prec^.next;
   end
  else // if prec^.next^.info<>valore
   prec:=prec^.next;
  end;
//Eliminazione Record generatore
```

```
prec:=lis1;
lis1:=prec^.next;
dispose(prec);
end;
```

Spiegazione: *Cancellazione dell'elemento che contiene come informazione il massimo valore tra tutti gli altri elementi.*

CODICE:

```
Procedure EliminaMax (var lis1:puntatore);
var
  max,prec,p:puntatore;

begin
  new (max);
  max^.info:=-MAXINT-1;
  max^.next:=lis1;
  prec:=max;
  if lis1<>nil then begin
                      p:=lis1;
                      repeat
                        if p^.info>max^.next^.info then max:=prec;
                        prec:=p;
                        p:=p^.next;
                      until p=nil;
                      if max^.next^.info=lis1^.info then begin
                                                           prec:=lis1;
                                                           lis1:=lis1^.next;
                                                           dispose (prec);
                                                         end
                      else begin
                             prec:=max^.next;
                             max^.next:=max^.next^.next;
                             dispose (prec);
                           end;
                    end;
end;


//Altra versione, fa uso di una funzione locale
Procedure EliminaMax(var lis1: puntatore; max: integer);
var
  paux: puntatore; //globale ad EliminaMax

(*TrovaMax è una funzione locale ad EliminaMax*)
Function TrovaMax(lis1: puntatore; max: integer):integer;
begin
if lis1<>NIL then
  begin
  if lis1^.info > max then max:=lis1^.info;
max:=TrovaMax(lis1^.next, max);
  end;
  TrovaMax:=Max;
end;{Fine della funzione TrovaMax}

(*Da qui in poi implementiamo EliminaMax*)
begin
if lis1<>NIL then
  if lis1^.info=TrovaMax(lis1,max) then
```

```
     begin
     paux:=lis1;
     lis1:=lis1^.next;
     dispose(paux);
     end
   else
     EliminaMax(lis1^.next,TrovaMax(lis1,max));
end;

//Un ulteriore versione….
Procedure EliminaMax(var lis1:puntatore);
var
  paux,prec:puntatore;
  mass: integer;
begin
paux:=lis1;
mass:=paux^.info;

//Ricerchiamo il valore massimo
while paux^.next<>NIL do
 begin
 if paux^.next^.info>mass then
  begin
  mass:=paux^.next^.info;
  paux:=paux^.next;
  end
 else
  paux:=paux^.next
 end;

//Creazione Record Generatore
new(prec);
prec^.next:=lis1;
lis1:=prec;

//Eliminazione dell'elemento contenente il valore massimo
while prec^.next<>NIL do
 begin
 if prec^.next^.info=mass then
  begin
  paux:=prec^.next;
  prec^.next:=paux^.next;
  dispose(paux);
  end
 else
  prec:=prec^.next;
 end;

//eliminazione Record generatore
prec:=lis1;
lis1:=lis1^.next;     //lis1:=prec^.next;
dispose(prec);
end;
```

Spiegazione: *Cancellazione dell'ultima occorrenza di elemento di una lista.*

CODICE:

```
Procedure CancUltimaOccorrenza(var lis1:puntatore; valore:integer);
var
```

```
  paux,prec:puntatore;
  i,j:integer;
begin
i:=0;
j:=0;
paux:=lis1;
//Controlliamo quante sono le occorrenze del valore nella lista
while (paux<>NIL) do
 begin
 if paux^.info=valore then
  begin
  i:=i+1;
  paux:=paux^.next
  end
 else paux:=paux^.next;
 end;
//Creazione Record generatore
new(prec);
prec^.next:=lis1;
lis1:=prec;
//Cerchiamo l'ultima occorrenza e la eliminiamo
while (prec^.next<>NIL) do
  begin
  if prec^.next^.info=valore then
   begin
    j:=j+1;
   if j=i then
    begin
    paux:=prec^.next;
    prec^.next:=paux^.next;
    dispose(paux);
    end
   else //if j<>i
     prec:=prec^.next;
   end
  else // if prec^.next^.info<>valore
   prec:=prec^.next;
  end;
//Eliminazione Record generatore
prec:=lis1;
lis1:=prec^.next;
dispose(prec);
end;
```

Spiegazione: Cancellazione dell'ultimo elemento da una lista (versione iterativa e ricorsiva)

CODICE:

```
{//Versione iterativa
Procedure EliminaUltimo(var lis1:puntatore);
var
 paux,prec:puntatore; //paux punta sempre l'elemento successivo puntato da prec
begin
(*Se la lista è composta da un solo elemento è necessario un controllo (perchè
 successivamente l'istruzione paux:=prec^.next potrebbe potrebbe dare prob dal
 momento che prec:=lis1*)
if lis1^.next=NIL then
 begin
```

```
Dispose(lis1);
 lis1:=NIL
 end;
if lis1<>NIL then
 begin
 prec:=lis1;
 paux:=prec^.next;
  while paux^.next<>NIL do
  begin
  paux:=paux^.next;
  prec:=prec^.next;
  end;
dispose(paux);          //si è chiuso il ciclo while, si può eliminare l'ultimo
elem
prec^.next:=NIL         //Chiudiamo la lista
 end;
end;
}
//Versione Ricorsiva
Procedure EliminaUltimo(var lis1:puntatore);
begin
if lis1<>NIL then
 begin
 if lis1^.next=NIL then
  begin
  dispose(lis1);
  lis1:=NIL
  end
 else
  EliminaUltimo(lis1^.next);
  end
else
  writeln('Lista gia'' vuota');
end;
```

Spiegazione: Conta le ripetizioni di un elemento in una lista.

CODICE:

```
{
//Versione iterativa
Function ContaRip(lis1: puntatore; valore:integer):integer;
var
  Ripetizioni:integer;
begin
Ripetizioni:=0;
while lis1<>NIL do
 begin
 if lis1^.info= valore then lis1:=lis1^.next
 else
   begin
   Ripetizioni:=Ripetizioni+1;
   lis1:=lis1^.next;
   end;
 end;
ContaRip:=Ripetizioni;
end;
}
//Versione ricorsiva
Function ContaRip(lis1: puntatore; valore:integer):integer;
```

```
begin
if lis1=NIL then ContaRip:=0
else
  begin
  if lis1^.info<>valore then ContaRip:=ContaRip(lis1^.next, valore)
  else
    ContaRip:=ContaRip(lis1^.next, valore)+1;
  end;
end;
```

Spiegazione: Effettua la copia di una lista.

CODICE:

```
Procedure Copia(var lis1: puntatore; var copia:puntatore);
begin
if lis1=NIL then copia:=NIL
else
  begin
  new(copia);
  copia^.info:=lis1^.info;
  Copia(lis1^.next,copia^.next);
  end;
end;
```

Spiegazione: *Da una lista, ne viene creata una nuova contenente gli elementi che hanno per campo informazione numeri divisibili per un dato valore.*

CODICE:

```
Procedure DivisibiliInLista(p_testa:puntatore;var p_testa2:puntatore;
valore:integer);
begin
 if (p_testa<>NIL) then
 begin
 if (p_testa^.info mod valore)=0 then
  begin
  new(p_testa2);
  p_testa2^.info:=p_testa^.info;
 DivisibiliInLista(p_testa^.next,p_testa2^.next,valore);
  end
 else
  DivisibiliInLista(p_testa^.next,p_testa2,valore);
 end;
end;
```

Spiegazione: *Cancella in una lista gli elementi il cui campo informazione non è divisibile per un dato valore.*

CODICE:

```
Procedure DivisibiliInLista(var p_testa:puntatore; valore:integer);
var
  paux: puntatore;
begin
if p_testa=NIL then warning:='Lista vuota'
else
if p_testa<>NIL then
```

```
(*Scartiamo gli elementi il cui campo info non è divisibile per valore. Alla
fine ci rimarrà la lista dei divisibili per valore*)
  if (p_testa^.info mod valore)<>0 then
      begin
      paux:=p_testa;
      p_testa:=p_testa^.next;
      dispose(paux);
      DivisibiliInLista(p_testa, valore);
      end
   else DivisibiliInLista(p_testa^.next, valore);
end;
```

SPIEGAZIONE: Cancella gli elementi i cui campi informazione sono numeri pari.

CODICE:

```
Procedure EliminaPari(var p_testa:puntatore);
var
  paux: puntatore;
begin
if p_testa<>NIL then
  if (p_testa^.info mod 2)=0 then
      begin
      paux:=p_testa;
      p_testa:=p_testa^.next;
      dispose(paux);
      EliminaPari(p_testa);
      end
   else EliminaPari(p_testa^.next);
end;
```

SPIEGAZIONE: Elimina gli elementi di posto pari.

CODICE:

```
Procedure EliminaPostoPari(p_testa:puntatore);
var
  paux: puntatore;
begin
   while (p_testa<>nil) do
       begin
           paux:=p_testa^.next;
           if paux<>nil then
               begin
                    p_testa^.next:=p_testa^.next^.next;
                    dispose (paux);
                end;
           p_testa:=p_testa^.next;
       end;
end;
```

SPIEGAZIONE: Elimina gli elementi di posto dispari.

CODICE:

```
Procedure EliminaPostoDisp(var p_testa: puntatore);
var
  paux: puntatore;
  p: puntatore;
begin
  if (p_testa<>NIL) then
  begin
    paux:=p_testa;
    p_testa:=p_testa^.next;
    dispose(paux);
  end;
  p := p_testa;
  while (p<>NIL) do
  begin
    paux:=p^.next;
    if paux<>nil then
    begin
      p^.next:=p^.next^.next;
      dispose (paux);
    end;
    p :=p^.next;
  end;
end;
```

SPIEGAZIONE: Verifica l'esistenza di un elemento in una lista ordinata.

CODICE:

```
//Versione iterativa
Function EsisteOrdinata(p_testa:puntatore; valore:integer):boolean;
var
  trovato:boolean;
begin
trovato:=FALSE;

while (p_testa<>NIL)AND not(trovato)AND not(valore<p_testa^.info) do
  begin
  if valore=p_testa^.info then trovato:=TRUE
  else
    p_testa:=p_testa^.next;
  end;
EsisteOrdinata:=trovato
end;

//Versione ricorsiva
Function EsisteOrdinata(p_testa:puntatore; valore:integer):boolean;
begin
if p_testa=NIL then EsisteOrdinata:=FALSE
else
  begin
  if valore<p_testa^.info then EsisteOrdinata:=FALSE
  else
  if valore=p_testa^.info then EsisteOrdinata:=TRUE
  else
  if valore>p_testa^.info then
EsisteOrdinata:=EsisteOrdinata(p_testa^.next,valore);
  end;
end;
```

AUTORE: PAOLO BIONDO

SPIEGAZIONE: Inserisce un elemento in una lista dopo un dato elemento scelto dall'utente.

CODICE:

```
Procedure InsDopo(var lis1:puntatore; dopoValore,qualeValore: integer);
var
 pNuovo:puntatore;
begin
if lis1<>NIL then
 begin
 if lis1^.info=dopoValore then
  begin
  new(pNuovo);
  pNuovo^.info:=qualeValore;
  pNuovo^.next := lis1^.next;
  lis1^.next:=pNuovo;
  end
 else
   InsDopo(lis1^.next,dopoValore,qualeValore);
 end;
end;
```

SPIEGAZIONE: Creazione di una lista ordinata.

CODICE:

```
procedure InserisciInListaOrdinata (var p_testa: puntatore; elem: integer);
{ Inserisce l'elemento elem nella lista lis ordinata per elementi crescenti,
  mantenendo l'ordinamento. Versione iterativa. Utilizza record generatore. }
var
  corr,                { usato per la scansione della lista }
  paux    : puntatore;
  trovato : boolean;   { indica se la posizione corretta e` stata trovata }
begin
//Immettiamo il valore che vogliamo ordinare in lista
write('Inserisci l''elemento nella lista ordinata: ');
read(elem);

if p_testa=NIL then     //Se la lista è vuota,semplicemente si inserisce in
testa l'elemento
               begin
               new(p_testa);
               p_testa^.inf:=elem;
               p_testa^.pun:=NIL;
               end
else  //altrimenti si procede cercando la posizione di inserimento, etc...
begin
{ creazione del record generatore }
new(paux);
paux^.pun := p_testa;
p_testa:= paux;

  { ricerca della posizione in cui inserire il nuovo elemento }
  corr := p_testa;
  trovato := FALSE;
  while (corr^.pun <> NIL) and (not trovato) do
```

```
    if corr^.pun^.inf < elem then
```



```
      corr := corr^.pun
    else  { l'elemento in testa e` maggiore o uguale a elem }
      trovato := TRUE;      { forza il termine della scansione }

  //Se un elemento è già in lista non lo rimettiamo
  if trovato then
        if NOT(corr^.pun^.inf= elem) then  //inserisce elem SE E SOLO SE elem è
diverso da corr^.pun^.inf
  begin
  { concatenazione del nuovo elemento }
  new(paux);                { 1 }  { alloca un record per il nuovo elemento }
  paux^.inf := elem;       { 2 }
  paux^.pun := corr^.pun; { 3 }
  corr^.pun := paux;       { 4 }
  end;
  { eliminazione del record generatore }
  paux := p_testa;
  p_testa := paux^.pun;
  dispose(paux)
end;
end; { InserisciInListaOrdinata }
```

SPIEGAZIONE: Verifica se due liste sono uguali.

CODICE:

```
Function Uguali(lis,lis2:puntatore):boolean;
begin
if (lis<>NIL) AND (lis2<>NIL) then
   begin
   if lis^.info<>lis2^.info then Uguali:=FALSE
    else //if lis^.info=lis2^.info then
       Uguali:=Uguali(lis^.next,lis2^.next)
   end
else if ((lis=NIL)AND(lis2<>NIL))OR((lis<>NIL)AND(lis2=NIL)) then
 Uguali:=False;
end;
```

SPIEGAZIONE: Verica se una lista è sottolista di un'altra lista.

CODICE:

```
Function subLista (lis1,lis2: puntatore):boolean;
var p1,p2: puntatore;
      b:boolean;
begin
  p1:=lis1;
  p2:=lis2;
  while (p2<>nil) and (p1<>nil) do
  begin
      p1:=lis1;
      b:=true;
         while b do
          begin
            if p1^.info<>p2^.info then   b:=false
            else p1:=p1^.next;
```

```
            p2:=p2^.next;
            if ((p1=nil) or (p2=nil))  then b:=false;
          end;
```

AUTORE:PAOLO BIONDO

```
  end;
  if ((p1=nil) and (lis1<>nil)) then subLista:=true
          else sublista:=false;
end;
```

SPIEGAZIONE: Verica se una lista è prefisso di un'altra lista.

CODICE:

```
(*La lista lis1 è prefisso della lis2 se, tutti gli elementi di lis1 coincidono
 con la parte iniziale di lis2, segue che una lista vuota è prefisso di
qualsiasi
 lista, se due liste sono uguali significa che una è prefisso dell'altra*)
Function Prefisso(lis1,lis2:puntatore):boolean;
begin
if (lis1<>NIL)AND(lis2=NIL) then Prefisso:=FALSE
else
if (lis1=NIL) then Prefisso:=TRUE
else
 if (lis1^.info=lis2^.info) then
 Prefisso:=Prefisso(lis1^.next,lis2^.next)
 else
  Prefisso:=FALSE;
end;
```

SPIEGAZIONE: Verica se una lista è postfisso di un'altra lista.

CODICE:

```
Function Postfisso(lis1,lis2:puntatore):boolean;
var
 paux:puntatore;
begin
paux:=lis1;
while (lis1<>NIL)AND(lis2<>NIL) do
 begin
 if lis1^.info=lis2^.info then
  begin
  lis1:=lis1^.next;
  lis2:=lis2^.next;
  end
 else
   begin
   lis1:=paux;
   lis2:=lis2^.next;
   end;
 end;
if (lis1=NIL)AND(lis2=NIL) then PostFisso:=TRUE
else
 PostFisso:=FALSE;
end;
```

SPIEGAZIONE: Calcola la somma,la media dei campi informazione di tutti gli elementi di una lista.

CODICE:

AUTORE: PAOLO BIONDO

```
Function SommaValoriLista(p_testa: puntatore): integer;
var
  Somma: integer;
begin
Somma:=0;
 while p_testa<>NIL do
   begin
   Somma:=Somma + p_testa^.info;
   p_testa:=p_testa^.next;
   end;
SommaValoriLista:=Somma;
end;

Function MediaValoriLista(p_testa: puntatore): real;
(*Contiamo quante volte viene eseguito il ciclo while,questo ci darà il numero
di elementi della lista, poi calcoleremo la media*)
var
  i,Somma:integer;
begin
Somma:=0;
i:=0;
 while p_testa<>NIL do
   begin
   Somma:=Somma + p_testa^.info;
   p_testa:=p_testa^.next;
   i:=i+1;
   end;
 MediaValoriLista:=Somma/i;
end;
```


ULTIMA MODIFICA: 31/10/2003