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
 pAlbero=^nodo;

  nodo=record
    info: integer;
    AlbSin: pAlbero;
    AlbDes: pAlbero;
  end;
```

Spiegazione: Aggiunge 1000 a tutte le informazioni contenute nei figli destri di un albero.

CODICE:

```
Procedure Add(var p:pAlbero; const plus:integer);
begin
if p<>NIL then
 begin
 if (p^.AlbDes<>NIL)then
   begin
   p^.AlbDes^.info:=p^.AlbDes^.info+plus;
   Add(p^.AlbDes,plus);
   end;
 if (p^.AlbSin<>NIL) then Add(p^.AlbSin,plus);
 end;
end;
```

Spiegazione: Verifica l'uguaglianza di due alberi.

CODICE:

```
Function alberi_uguali(p,p2:pAlbero):boolean;
begin
   if(p=nil)and(p2=nil)then
      alberi_uguali:=true
   else if(p=nil)and(p2<>nil)then
      alberi_uguali:=false
   else if(p<>nil)and(p2=nil)then
      alberi_uguali:=false
   else if(p^.info<>p2^.info)then
      alberi_uguali:=false
   else
      alberi_uguali:=alberi_uguali(p^.AlbSin,p2^.AlbSin)and
                     alberi_uguali(p^.AlbDes,p2^.AlbDes)
end;
```

AUTORE: PAOLO BIONDO

Spiegazione: Verifica se un albero è completo.

CODICE:

```
Function Completo(p:pAlbero):boolean;
begin
   if(p=nil)or((p^.AlbSin=nil)and(p^.AlbDes=nil))then
      Completo:=true
   else
      if(p^.AlbSin<>nil)and(p^.AlbDes<>nil)then
         Completo := Completo(p^.AlbSin) and Completo(p^.AlbDes)
      else
         Completo:=false

end;
```

Spiegazione: Eliminazione di un nodo di un albero.

CODICE:

```
{
La cancellazione di un elemento da un albero è un processo non semplice dal
momento
che cancellato un nodo dobbiamo fare in modo di tener "legato" tutto il resto
dell'albero.
Ci sono diversi modi per poter cancellare un elemento da un nodo.
1)Come abbiamo fatto qui. Si cerca il nodo da eliminare nell'albero e quando lo
si trova
  si chiama una funzione di appoggio "CancellaMin" che si occupa di cercare il
nodo
  con l'informazione più piccola del sottoAlbero destro, si prende tale nodo
  (spostandolo dalla vecchia posizione fino alla radice(se dobbiamo eliminare il
nodo
  alla radice).Alla fine di queste operazioni l'albero dovrebbe mantenere
intatta la sua forma di ABR.

2)Si agisce in maniera simile al tipo1, la funzione "CancellaMin2 viene
sostituita da
  "CancellaMax", questa volta si effettua la ricerca del nodo con informazione
più
  grande nel sottoAlbero sinistro e si rende tale nodo la nuova radice dell'ABR

}
Function CancellaMin(var p:pAlbero):integer;
begin
 if p^.AlbSin=NIL then
    begin
    CancellaMin:=p^.info;
    p:=p^.AlbDes
    end
 else
    CancellaMin:=CancellaMin(p^.AlbSin);
end;

Procedure Cancella(var p:pAlbero; valore: integer);
begin
 if p<>NIL then
   if valore<p^.info then Cancella(p^.AlbSin,valore)
   else if valore>p^.info then Cancella(p^.AlbDes,valore)
```

```
   else //if valore=p^.info
     if p^.AlbSin=NIL then p:=p^.AlbDes
     else if p^.AlbDes=NIL then p:=p^.AlbSin
     else  //se nessuno dei due figli è a Nil
      p^.info:=CancellaMin(p^.AlbDes);
end;
```

SPIEGAZIONE: Ricerca dell'altezza di un nodo di un albero.

CODICE:

```
(*altAus è l'altezza attuale dell'albero,serve da var. ausiliaria per trovare
l'altezza
del nodo cercato*)
Procedure AltezzaN(p :pAlbero; valore :integer; altAus :integer; var altezza
:integer; found : boolean);
begin
  if found = false then
  begin
    if p <> nil then
    begin
      if p^.info = valore then AltezzaN(p,valore,0,altezza,true)
      else
      begin
        AltezzaN(p^.AlbSin, valore,0,altezza,false);
        AltezzaN(p^.AlbDes, valore,0,altezza,false);
      end;
    end;
  end
  else  //if found=true
  begin
    if p<>nil then
    begin
      if altezza<altAus then altezza:=altAus;
      AltezzaN(p^.AlbSin,valore,altAus+1,altezza,true);
      AltezzaN(p^.AlbDes,valore,altAus+1,altezza,true);
    end;
  end;
end;
```

SPIEGAZIONE: Conta i nodi di un albero.

CODICE:

```
Function ContaNodi(p:pAlbero):integer;
begin
   if(p=nil)then
      ContaNodi:=0
   else
   ContaNodi:=ContaNodi(p^.AlbSin)+ContaNodi(p^.AlbDes)+1;
end;
```

SPIEGAZIONE: Dealloca il sottoalbero che ha per nodo radice "valore".

CODICE:

```
Procedure DeallocaSottoAlbero( var p:pAlbero; valore: integer);
begin
```

```
  if ( p<>NIL ) then
  begin
    if valore=p^.info then //questo è il caso in cui dobbiamo deallocare tutto
l'albero
      begin                //poichè il valore cercato è uguale a quello del nodi
radice
      DeallocaSottoAlbero(p^.AlbSin,valore);
      DeallocaSottoAlbero(p^.AlbDes, valore);
      dispose(p);
      p:=NIL
      end
else
   if valore<p^.info then DeallocaSottoAlbero(p^.AlbSin,valore)
else
   if valore>p^.info then DeallocaSottoAlbero(p^.AlbDes,valore)
  end;
end;
```

SPIEGAZIONE :Conta il numero di foglie, di nodi con un figlio e nodi con due figli.

CODICE:

```
(*foglie,unFiglio,dueFigli sono tutti passati per indirizzo, questo perchè
andiamo a modificare di volta in volta il loro valore*)
Procedure AllInOne(p:pAlbero;var foglie,unFiglio,dueFigli: integer);
begin
if p<>NIL then
 begin
  if (p^.AlbSin=NIL) AND (p^.AlbDes=NIL) then Inc(foglie);
 if ((p^.AlbSin=NIL) AND (p^.AlbDes<>NIL)) OR ((p^.AlbSin<>NIL) AND
(p^.AlbDes=NIL)) then Inc(unFiglio);
 if (p^.AlbSin<>NIL) AND (p^.AlbDes<>NIL) then Inc(dueFigli);

 AllInOne(p^.AlbSin, foglie,unFiglio,dueFigli);
 AllInOne(p^.AlbDes, foglie,unFiglio,dueFigli);

 end;
end;
```

SPIEGAZIONE: Verifica che un albero è lineare.

CODICE:

```
(*Un albero si dice lineare se ogni nodo ha al più un figlio*)
Function Lineare(p:pAlbero):boolean;
begin
 if p=NIL then Lineare:=TRUE
 else
  begin
   if ((p^.AlbSin=NIL) AND (p^.AlbDes=NIL)) then Lineare:=TRUE
   else Lineare:=FALSE;

    if ((p^.AlbSin=NIL) AND (p^.AlbDes<>NIL))OR ((p^.AlbSin<>NIL) AND
(p^.AlbDes=NIL))
     then Lineare:=Lineare(p^.AlbSin) AND Lineare(p^.AlbDes);

  end;
end;
```

AUTORE: PAOLO BIONDO

SPIEGAZIONE: Trova il max e il min tra i nodi di un albero.

CODICE:

```
Function Minimo(p:pAlbero): integer;
begin
if p<>NIL then
 begin
 if p^.info<piccolo then piccolo:=p^.info;
 piccolo:=Minimo(p^.AlbSin);
 if piccolo<p^.info then Minimo:=piccolo //di sicuro se c'è un minimo si trova
nel sottAlb sin
 else Minimo:=p^.info
 end
else Minimo:=MAXINT;
end;

Function Massimo(p:pAlbero):integer;
begin
 if p<>NIL then
  begin
  if p^.info>max then max:=p^.info;
  max:=Massimo(p^.AlbDes); //di sicuro se c'è un massimo si trova nel sottAlb
destro
  if max>p^.info then Massimo:=max
  else Massimo:=p^.info
  end
 else Massimo:=-MAXINT-1;
end;
```

SPIEGAZIONE: Pota un albero, ossia elimina le foglie di un albero.

CODICE:

```
Procedure Pota(var p:pAlbero);
begin
if p<>NIL then
 begin
 if (p^.AlbSin=NIL) AND (p^.AlbDes=NIL) then
  begin
  dispose(p);
  p:=NIL
  end
 else
   begin
   pota(p^.AlbSin);
   pota(p^.AlbDes);
   end;
 end;
end;
```

SPIEGAZIONE: Verifica la presenza di un nodo nell'albero. (versione base ed ottimizzata)

CODICE:

```
Function Presente(p:pAlbero; valore: integer): boolean;
begin
```

```
 if p=NIL then Presente:=FALSE //Se l'albero è vuoto sicuramente l'informazione
non è presente
  else if p^.info=valore then Presente:=TRUE //la radice contiene l'info cercata
  else
    Presente:=Presente(p^.AlbSin, valore) OR Presente(p^.albDes, valore);
end;

(*Versione ottimizzata, se l'informazione è maggiore del campo info della
radice,
ricerchiamo l'eventuale presenza dell'elemento solo nel sottoalbero destro (se è
 minore nel sottalb sinistro).In questo modo risparmiamo diversi controlli *)
Function Presente(p:pAlbero; valore: integer): boolean;
begin
 if p=NIL then Presente:=FALSE //Se l'albero è vuoto sicuramente l'informazione
non è presente
  else if p^.info=valore then Presente:=TRUE //la radice contiene l'info cercata
  else
    if valore > p^.info then
    begin
    Presente:=Presente(p^.AlbDes,valore);
    i:=i+1;
    end
    else
      begin
      Presente:=Presente(p^.AlbSin,valore);
      i:=i+1;
      end;
end;
```

SPIEGAZIONE: Cerca la profondità di un nodo scelto dall'utente.

CODICE:

```
Function profonditaNodo(p:pAlbero; valore:integer):integer;
begin
if p=NIL then profonditaNodo:=-1
else
 begin
 if valore=p^.info then profonditaNodo:=0
 else
 if valore<p^.info then ProfonditaNodo:=profonditaNodo(p^.AlbSin,valore)+1
 else
 if valore>p^.info then ProfonditaNodo:=profonditaNodo(p^.AlbDes,valore)+1
 end;
end;
```

SPIEGAZIONE: Somma delle "info" dei nodi.

CODICE:

```
Function somma_valori_nodi(p:pAlbero):integer;
begin
   if(p=nil)then
      somma_valori_nodi:=0
   else
      somma_valori_nodi:=somma_valori_nodi(p^.AlbSin)+
                         somma_valori_nodi(p^.AlbDes)+p^.info;
end;
```

AUTORE:PAOLO BIONDO

SPIEGAZIONE: Somma delle “info” delle foglie.

CODICE:

```
Function SommaF(p:pAlbero):integer;
begin
if p<>NIL then
  begin
  if (p^.AlbSin=NIL) AND (p^.AlbDes=NIL) then SommaF:=p^.info   //CASO BASE
  else
    SommaF:=SommaF(p^.AlbSin)+SommaF(p^.AlbDes);
  end;
end;
```


ULTIMA MODIFICA: 31/10/2003