```
{
Autor : Slawek Kolasinski
Data  : 03.2009
e-mail: skola@mimuw.edu.pl
}

Program BST;
type
   typ    = integer;
   drzewo = ^wezel;
   wezel  = record
               wartosc : typ;
               lewy    : drzewo;
               prawy   : drzewo;
            end;

function max(x,y : typ) : typ;
begin
   if x > y then
      max := x
   else
      max := y;
end; { max }

procedure inicjuj(var t : drzewo);
begin
   t := nil;
end; { inicjuj }

function puste(t : drzewo) : boolean;
begin
   puste := (t = nil);
end; { puste }

procedure dodaj(var t : drzewo; x : typ);
begin
   if puste(t) then begin
      new(t);
      t^.wartosc := x;
      { zainicjuj lewe i prawe poddrzewo, ¯…eby byly puste }
      inicjuj(t^.lewy);
      inicjuj(t^.prawy);
   end else if x < t^.wartosc then
      dodaj(t^.lewy, x)
   else
      dodaj(t^.prawy, x);
end; { dodaj }

function zbal_pom(r : drzewo; var wys : integer) : boolean;
var
   w1,w2 : integer;
begin
   if puste(r) then begin
      zbal_pom := true;
      wys := 0;
   end else begin
      zbal_pom := zbal_pom(r^.lewy, w1) and zbal_pom(
                           r^.prawy, w2) and (abs(w1 - w2) <= 1);
      wys := max(w1,w2) + 1;
   end;
```





```
end; { zbal_pom }

function zbalansowane(t : drzewo) : boolean;
var
   wys : integer;
begin
   zbalansowane := zbal_pom(t,wys);
end; { zbalansowane }

procedure kasuj(var t : drzewo);
begin
   if not puste(t) then begin
      kasuj(t^.lewy);
      kasuj(t^.prawy);
      dispose(t);
      t := nil;
   end;
end; { kasuj }

procedure predfs_print(t : drzewo);
begin
   if puste(t) then exit;
   write(t^.wartosc, ' ');
   predfs_print(t^.lewy);
   predfs_print(t^.prawy);
end; { predfs_print }

function max_pelne_wys(t : drzewo; var wys : integer) : drzewo;
var
   lt,pt : drzewo;
   lw,pw : integer;
begin
   if puste(t) then begin
      max_pelne_wys := t;
      wys := 0;
   end else begin
      lt := max_pelne_wys(t^.lewy, lw);
      pt := max_pelne_wys(t^.prawy, pw);
      if (lt = t^.lewy) and (pt = t^.prawy) and (
                                                pw = lw) then begin
         max_pelne_wys := t;
         wys := lw + 1;
      end else if lw > pw then begin
         max_pelne_wys := lt;
         wys := lw;
      end else begin
         max_pelne_wys := pt;
         wys := pw;
      end;
   end;
end; { max_pelne_wys }

function max_pelne(t : drzewo) : drzewo;
var
   wys : integer;
begin
   max_pelne := max_pelne_wys(t,wys);
end; { max_pelne }

function jest_bst(t : drzewo) : boolean;
```

```
119 begin
120    if puste(t) then jest_bst := true
121    else
122       jest_bst := (puste(t^.lewy) or (
                                t^.lewy^.wartosc < t^.wartosc)) and
123                   (puste(t^.prawy) or (
                               t^.prawy^.wartosc >= t^.wartosc)) and
124                   jest_bst(t^.lewy) and jest_bst(t^.prawy);
125 end; { jest_bst }

127 var
128    t1,pom : drzewo;

130 begin
131    inicjuj(t1);

133    dodaj(t1, 6); dodaj(t1, 3); dodaj(t1, 10); dodaj(t1, 1);
                                                      dodaj(t1, 12);
134    dodaj(t1, 2); dodaj(t1, 11); dodaj(t1, 4); dodaj(t1, 8);
                                                       dodaj(t1, 5);
135    dodaj(t1, 9); dodaj(t1, 13); dodaj(t1, 0); dodaj(t1, 7);

137    write('t1: '); predfs_print(t1); writeln;

139    writeln('t1 BST?: ', jest_bst(t1));
140    writeln('t1 zbalansowane: ', zbalansowane(t1));

142    writeln('maksymalne pelne poddrzewo zaczepione w ', max_pelne(
                                                      t1)^.wartosc);

144    { dodamy sztucznie jakis wezel i sprawdzimy czy jest_bst
                                                          dziala }
145    new(pom);
146    pom^.wartosc := -100;
147    pom^.prawy := nil;
148    pom^.lewy := t1;
149    writeln('pom BST?: ', jest_bst(pom));
150    dispose(pom);
151    pom := nil;

153    kasuj(t1);
154    write('po skasowaniu t1: '); predfs_print(t1); writeln;
155 end.
```