```
{
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 lisc(t : drzewo) : boolean;
begin
   lisc := puste(t^.lewy) and puste(t^.prawy);
end; { lisc }

function liczba_lisci(t : drzewo) : integer;
begin
   if puste(t) then
      liczba_lisci := 0
   else if lisc(t) then
      liczba_lisci := 1
   else
```





```
      liczba_lisci := liczba_lisci(t^.lewy) + liczba_lisci(
                                                   t^.prawy);
end; { liczba_lisci }

function liczba_wezlow(t : drzewo) : integer;
begin
   if puste(t) then
      liczba_wezlow := 0
   else
      liczba_wezlow := 1 + liczba_wezlow(
                              t^.lewy) + liczba_wezlow(t^.prawy);
end; { liczba_wezlow }

function znajdz_min(t : drzewo; var wsk : drzewo) : drzewo;
begin
   wsk := nil;
   if not puste(t) then begin
      while not puste(t^.lewy) do begin
         wsk := t;
         t := t^.lewy;
      end;
   end;
   znajdz_min := t;
end; { znajdz_min }

procedure usun_korzen(var t : drzewo);
var
   korzen,min,wsk : drzewo;
begin
   if puste(t) then begin
      writeln('usun_korzen: Drzewo puste!');
      halt;
   end else if lisc(t) then begin
      dispose(t);
      t := nil;
   end else begin
      korzen := t;
      if puste(t^.lewy) then begin
         t := t^.prawy;
         dispose(korzen);
      end else if puste(t^.prawy) then begin
         t := t^.lewy;
         dispose(korzen);
      end else begin
         min := znajdz_min(t^.prawy, wsk);
         t^.wartosc := min^.wartosc;
         usun_korzen(min);
         if wsk = nil then t^.prawy := min
         else wsk^.lewy := min;
      end;
   end;
end; { usun_korzen }

function znajdz_pom(t : drzewo; x : typ; var wsk : drzewo;
                                      var lewy : boolean) : drzewo;
begin
   wsk := nil;
   while (not puste(t)) and (t^.wartosc <> x) do begin
      wsk := t;
      if x < t^.wartosc then begin
```

```
118            t := t^.lewy;
119            lewy := true;
120         end else begin
121            t := t^.prawy;
122            lewy := false;
123         end;
124      end;
125      znajdz_pom := t;
126   end; { znajdz_pom }

128   procedure usun(var t : drzewo; x : typ);
129   var
130      wezel,wsk : drzewo;
131      lewy      : boolean;
132   begin
133      wezel := znajdz_pom(t, x, wsk, lewy);
134      if not puste(wezel) then begin
135         usun_korzen(wezel);
136         if puste(wsk) then t := wezel
137         else if lewy then wsk^.lewy := wezel
138         else wsk^.prawy := wezel;
139      end;
140   end; { usun }

142   procedure kasuj(var t : drzewo);
143   begin
144      if not puste(t) then begin
145         kasuj(t^.lewy);
146         kasuj(t^.prawy);
147         dispose(t);
148         t := nil;
149      end;
150   end; { kasuj }

152   procedure predfs_print(t : drzewo);
153   begin
154      if puste(t) then exit;
155      write(t^.wartosc, ' ');
156      predfs_print(t^.lewy);
157      predfs_print(t^.prawy);
158   end; { predfs_print }

160   var
161      t1 : drzewo;

163   begin
164      inicjuj(t1);

166      dodaj(t1, 6); dodaj(t1, 3); dodaj(t1, 10); dodaj(t1, 1);
                                                        dodaj(t1, 12);
167      dodaj(t1, 2); dodaj(t1, 11); dodaj(t1, 4); dodaj(t1, 8);
                                                         dodaj(t1, 5);
168      dodaj(t1, 9); dodaj(t1, 13); dodaj(t1, 0); dodaj(t1, 7);

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

172      usun_korzen(t1);
173      write('t1 po usunieciu korzenia: '); predfs_print(t1);
                                                              writeln;
```





```
175      usun(t1, 3);
176      write('t1 po usunieciu 3: '); predfs_print(t1); writeln;

178      usun(t1, 5);
179      write('t1 po usunieciu 5: '); predfs_print(t1); writeln;

181      kasuj(t1);
182      write('po skasowaniu t1: '); predfs_print(t1); writeln;
183   end.

185   {
186   WYNIK DZIALANIA PROGRAMU NA EKRANIE:

188   t1: 6 3 1 0 2 4 5 10 8 7 9 12 11 13
189   t1 po usunieciu korzenia: 7 3 1 0 2 4 5 10 8 9 12 11 13
190   t1 po usunieciu 3: 7 4 1 0 2 5 10 8 9 12 11 13
191   t1 po usunieciu 5: 7 4 1 0 2 10 8 9 12 11 13
192   po skasowaniu t1:
193   }
```