```
(* Program za URAVNOTEZENO VECSMERNO ZLIVANJE kot je zapisan v knjigi *)
(* Niklaus Wirth: Racunalnisko programiranje-2.del, (str.159-161).    *)
(* Pozor, program je zapisan v STANDARDNEM Pascalu!                   *)

Program BalancedMerge(output);
(* balanced n-way tape merge sort *)

  Const n=6;
        nh=3;   (* number of input tapes *)

  Type item=record
         key:integer;
       end;
       tape = file of item;
       tapeno = 1..n;

  Var leng,rand:integer; (* used to generate file *)
      eot:boolean;  (* end of tape *)
      buf:item ;
      f0:tape; (* f0 is input tape with random numbers *)
      f:array[1..n] of tape;

  Procedure List(var f:tape; n:tapeno);

  var z:integer;

  begin
    writeln('TAPE ',n:2);
    z:=0;
    while not eof(f) do
    begin
      read(f,buf);
      write(output,buf.key:5);
      z:=z+1;
      if z = 25 then
      begin
         writeln(output);
         z:=0;
      end;
    end;
    if z<>0 then
      writeln(output);
    reset(f);
  end; (* List *)

  Procedure TapeMergeSort;

  var i,j,mx,tx:tapeno;
      k1,k2,l:integer;
      x,min:integer;
      t,ta: array[tapeno] of tapeno;
```

```
begin
  (* distribute initial runs to t[1]..t[nh] *)
  for i := 1 to nh do
    rewrite(f[i]);
  j:=nh;
  l:=0;
  repeat
    if j<nh then
      j:=j+1
    else
      j:=1;
    (* copy one run from f0 to tape j *)
    l:=l+1;
    repeat
      read(f0,buf);
      write(f[j],buf);
    until (buf.key>f0^.key) or eof(f0)
  until eof(f0);
  for i:=1 to n do
    t[i]:=i;
  repeat
    (* merge from t[1]…t[nh] to t[nh+1]…t[n] *)
    if l<nh then
      k1:=l
    else
      k1:=nh;
      (* k1 = number of input tapes in this phase *)
    for i:=1 to k1 do
    begin
      reset(f[t[i]]);
      List(f[t[i]],t[i]);
      ta[i]:=t[i];
    end;
    l:=0;     (* l = number of runs merged *)
    j:=nh+1;  (* j = index of output tape *)
    repeat
      (* merge a run from t[1]…t[k1] to t[j] *)
      k2:=k1;  (* k2 = number of active input tapes *)
      l:=l+1;
      repeat
        (* select minimal element *)
        i:=1; mx:=1;
        min:=f[ta[1]]^.key;
        while i<k2 do
        begin
          i:=i+1;
          x:=f[ta[i]]^.key;
          if x<min then
          begin
            min:=x;
            mx:=i;
          end;
```

```
        end;
        (* ta[mx] has te minimal element, move it to t[j] *)
        read(f[ta[mx]],buf);
        eot:=eof(f[ta[mx]]);
        write(f[t[j]],buf);
        if eot then
        begin
          rewrite(f[ta[mx]]);  (* eliminate tape *)
          ta[mx]:=ta[k2];
          ta[k2]:=ta[k1];
          k1:=k1-1;
          k2:=k2-1;
        end
        else
          if buf.key>f[ta[mx]]^.key then
          begin
            tx:=ta[mx];
            ta[mx]:=ta[k2];
            ta[k2]:=tx;
            k2:=k2-1
          end
      until k2=0;
      if j<n then
        j:=j+1
      else
        j:=nh+1;
    until k1=0;
    for i:=1 to nh do
    begin
      tx:=t[i];
      t[i]:=t[i+nh];
      t[i+nh]:=tx;
    end
  until l=1;
  reset(f[t[1]]);
  List(f[t[1]],t[1]);
  (* sorted output is on t[1] *)
end; (* TapeMergeSort *)

Begin
  (* generate random file *)
  leng:=200; rand:=7789;
  rewrite(f0);
  repeat
    rand:=(131071*rand) mod 2147483647;
    buf.key:= rand div 2147484;
    write(f0,buf);
    leng:=leng-1;
  until leng=0;
  reset(f0);
  List(f0,1);
  TapeMergeSort;
```

End.