Práce se seznamy

Funkce pro zjištění prázdnosti seznamu

```
function LEmpty(L:TL):boolean;
begin
  First(L);
  LEmpty:=not Active(L)
end
```

Funkce pro zjištění délky jednosměrného seznamu.

```
function Length(L:TL):integer; (* délka seznamu*)
var Count:integer;
begin
   Count:=0;
   First(L);     (* Nastavení prvního jako aktivního *)
   while Active(L)do begin
     Count:=Count+1;
     succ(L)
   end; (* while *)
   Length:= Count
end
```

Práce s jednosměrnými seznamy

ListInit

```
procedure ListInit (var L:TList);
begin
  L.Act:=nil;
  L.Frst:=nil
end; (* ListInit *)
```

InsertFirst

```
procedure InsertFirst(var L:Tlist,El:TEl);
var
  UkPomEl:TUk;
begin
  new(UkPomEl);                (* Vytvoření nového prvku *)
  UkPomEl^.Data:=El;           (* Nastavení datové složky *)
  UkPomEl^.UkNasl:=L.Frst;     (* Uk tam, kam začátek *)
  L.Frst:=UkPomEl;             (* Zač. ukazuje na nový prvek *)
end; (* InsertFirst *)
```

PostInsert

```
procedure PostInsert (var L:Tlist; El:TEl);
var  PomUk:Tuk;
begin
    if L.Act <> nil then begin   (* jen pro aktivní seznam *)
       new(PomUk);
       PomUk^.Data:=El;
       PomUk^.UkNasl:=L.Act^.UkNasl;    (* Nový ukazuje
                                    tam, kam aktivní *)
       L.Act^.UkNasl:= PomUk        (* Aktivní ukazuje  na nového *)
end;
```

Copy

```
procedure Copy (L:Tlist; var El:TEl);
(*  Operace předpokládá ošetření aktivity seznamu ve tvaru: if Active(L) then begin
Copy(L,El); ...  Copy v neaktivním seznamu způsobí chybu *)
begin
   El:= L.Act^.Data
end;
```

Actualize

```
procedure Actualize (var L:Tlist; El:TEl);
begin
   if L.Act<> nil
   then begin


      L.Act^.Data:=El
    end
 end;
```

PostDelete

```
procedure PostDelete(var L:Tlist);
var
   PomUk:Tuk;
begin
    if (L.Act<>nil)
    then
       if L.Act^.UkNasl <> nil
       then  begin (* je co rušit *)
          PomUk:=L.Act^UkNasl;  (* Ukazatel na rušeneho *)
          L.Act^.UkNasl:=PomUk^.UkNasl;  (* Překlenutí rušeného*)
          Dispose(PomUk)
       end (*if L.Act^.UkNasl<> *)
     end (*if L.Act <> *)
end;
```

```
procedure CopyList(LOrig:TList; var LDupl:TList);
var
  El:char;
begin
  InitList(LDupl);
  First(LOrig);
  while Active(LOrig) do begin
    Copy(LOrig, El);
    Succ(LOrig);
    InsertLast(LDupl,El)
  end;
end;
```

Dvojsměrný seznam

DListInit

```
procedure DListInit(var L:TDList);
begin
   L.Frst:=nil;
   L.Lst:=nil;
   L.Act:=nil;
   (* PocPrv:=0; *)
end;
```

```
procedure DInsertFirst (var L:TDlist; El:TData);
var
   PomUk:TUkPrv;
begin
   new(PomUk);
   PomUk^.Data:=El;
   PomUk^.LUk:= nil;     (* levý nového ukazuje na nil *)
   PomUk^.PUk:= L.Frst;     (* Pravý nového ukazuje
                               na aktuálně prvního nebo nil *)
   if L.Frst<> nil
   then begin
     L.Frst^.LUk:=PomUk;        (* Aktuální první bude
                                   ukazovat doleva na nový prvek*)
   end else begin     (* Vkládám do prázdného seznamu *)
     L.Lst:=Pomuk
   end;
   L.Frst:=PomUk;              (* Korekce ukazatele začátku *)
```

```
procedure DDeleteFirst (var L:TDlist);
(* Nutno sledovat zda operace neruší aktivní prvek resp. jediný prvek *)
var
   PomUk:TUkPrv;
begin
  if L.Frst<>nil
  then begin
     PomUk:= L.Frst;
     if L.Act=L.Frst
     then L.Act:= nil;     (* první byl aktivní, ruší se aktivita seznamu *)

    if L.Frst=L.Lst
      then begin  (* ruší se jediný prvek sezn., vznikne prázdný*)
      L.Lst:=nil;
      L.Frst := nil
    end else begin
      L.Frst:= L.Frst^.PUk;  (*  Aktualizace začátku sezn.*)
      L.Frst^.LUk:= nil;   (* Není-li seznam prázdný, první prvek ukazuje
doleva uk. nil *)
    end; (* L.Frst= *)
    Dispose(PomUk)
  end; (* if L.Frst <> *)
end; (* procedure *)

procedure DPostInsert (var L:TDList; El:TData);
(*  Procedura musí dávat pozor, zda nevkládá za poslední prvek *)
var
   PomUk:TUkPrv;
begin
   if L.Act<>nil
   then begin
      new(PomUk);
      PomUk^.Data:= El;
      PomUk^.PUk:= L.Act^.PUk; (* *)
      PomUk^.LUk:= L.Act;
      L.Act^.Puk:= PomUk;   (*Navázání lev. souseda na vložený prv.*)
      if L.Act=L.Lst
      then  L.Lst:=PomUk   (* Korekce ukaz. na konec – nový poslední *)
      else PomUk^.PUk^.LUk:=PomUk     (*  navázání pravého
                                  souseda vložený prvek *)
    end; (* if *)
end; (* procedure *)
```

```
procedure DPostDelete(var L:DList);
(* Nutno sledovat, zda neruší poslední prvek *)
var
   PomUk: TUkPrv;
begin
   if (L.Act <> nil)
   then begin
        if L.Act^.PUk <> nil
(* Je co rušit? *)
        then begin      (* Rušený existuje *)
           PomUk:= L.Act^.PUk;     (* ukazatel na rušený*)
           L.Act^.PUk:= PomUk^.PUk;   (* překlenutí rušeného  *)
           if PomUk = L.Lst
           then L.Lst:=L.Act   (*  je-li rušený poslední, Act bude Lst *)
           else (* ruší se běžný prvek  *)
              PomUk^.PUk^.LUk:= L.Act;     (* prvek za zrušeným
                                    ukazuje doleva na Act *)
            Dispose(PomUk);
        end; (*if L.Act^.Puk <> nil *)
      end; (* if L.Act<> nil *)
end; (* procedure *)

procedure DPreDelete(var L:DList);
(* Nutno sledovat, zda neruší první prvek *)
var
  PomUk: TUkPrv;
begin
  if (L.Act <> nil)
  then begin
    if L.Act^.LUk <> nil (* Je co rušit? *)
    then begin  (* Rušený existuje *)
      PomUk:= L.Act^.LUk; (* ukazatel na rušený*)
      L.Act^.LUk:= PomUk^.LUk;(* překlenutí rušeného  *)
      if PomUk = L.Frst
      then L.Frst:=L.Act (* Je-li rušen první, stane se aktivní prvním *)
      else (* ruší se běžný prvek  *)
        PomUk^.LUk^.PUk:= L.Act;     (* prvek před zrušeným ukazuje
doprava na activní prvek *)
      Dispose(PomUk)
    end (*if L.Act^.Luk <> nil *)
  end (* if L.Act<> nil *)
end; (* procedure *)
```

Zásobník

```
procedeure SInit(var  S:TStack);
begin
   S.TopUk:=nil
   ...(* zde by byly inicializovány další atributy zásobníku, např. "pocet:=0"  *)
end;

procedure Push(var S:TStack; El:TElem);
var
   PomUk:TUk;
begin
   new(PomUk);
   PomUk^.Data:=El;
   PomUk^.UkDalsi:=S.TopUk;
   S.TopUk:=PomUk
end;

procedure Pop(var S:TStack);
var
    PomUk:TUk;
begin
    if S.TopUk<>nil
    then begin
       PomUk:=S.TopUk;
       S.TopUk:=S.TopUk^.UkDalsi;
       dispose(PomUk)
   end (* if *)
end;

function SEmpty(S:TStack):Boolean;
begin
   SEmpty:=S.TopUk=nil
end;
```

Fronta

```
procedure QInit(var Q:TQueue);
begin
  Q.QZacUk:=nil;
  Q.QKonUk:=nil
end;

procedure QueUp(var Q;TQueue; El:TElem);
var
  PomUk:TUk;
begin
  new (PomUk);
  PomUk^.Data:= El;     (* naplnění nového prvku *)
  PomUk^.UkDalsi :=nil;   (* ukončení nového prvku *)

  if Q.ZacUk = nil
  then (* fronta je prázdná, vlož nový jako první a jediný*)
    Q.QZacUk:=PomUk
  else (* fronta obsahuje nejméně jeden prvek, přidej na konec*)
    Q.QKonUk^.UkDalsi:=Pomuk;
  Q.QKonUk:=PomUk   (* korekce konce fronty *)
end;
```

```
procedure Front (Q;TQueue; var El: TElem);
begin
  El:=Q.QZacUk^.Data
end;


procedure Remove (var Q:TQueue);
var
  PomUk: TUk;
begin
  if Q.QZacUk <> nil
  then  begin (* Fronta je neprázdná *)
    PomUk := Q.QZac;
    if Q.QZac = Q.QKon
  then Q.QKon:=nil;   (* Zrušil se poslední a jediný prvek fronty*)
  Q.QZac:=Q.QZac^.UkDalsi;
  dispose (PomUk);
 end (* if Q.QZac <> *)
end;
```

Binární stromy

```
procedure PreOrder(var L:TList;RootPtr:TPtr);
  (* Seznam L byl inicializován před voláním *)
begin
  if RootPtr  <> nil
  then begin
    DInsertLast(L,RootPtr^.Data);
    PreOrder(L,RootPtr^.LPtr);
    PreOrder(L,RootPtr^.RPtr)
  end;
end; (* procedure *)
```

Záměnou pořadí rekurzívního volání v podmíněném příkazu if získáme průchody InOrder a PostOrder

```
(* Inorder *)
    Inorder(L,RootPtr^.LPtr);
    DInsertlast(L,RootPtr^.Data);
    Inorder(L,RootPtr^.RPtr);

 (* Postorder *)
    Postorder(L,RootPtr^.LPtr);
    Postorder(L,RootPtr^.RPtr);
    DInsertlast(L,RootPtr^.Data).
```

```
procedure NejlevPre(Uk:TUk; var L:TList);
(* Procedura Nejlev pro Preorder. Používá globální ADT zásobník ukazatelů "S" *)
begin
   while Uk<>nil do begin
       Push(S,Uk);
       InsertLast(L,Uk^.Data)  (* vložení na konec seznamu*)
          Uk:= Uk^.LUk
   end
end;

procedure PreOrder(Uk:TUk; var L:TList);
begin
   ListInit(L); SInit(S);(* inicializace seznamu a zásobníku *)
   NejlevPre(Uk,L);
   while not SEmpty(S) do begin
      Top(S,Uk); Pop(S);
      NejlevPre(Uk^.PUk,L)
   end
end;
```

```
procedure NejlevPost(Uk:TUk);
(* Globální ADT zásobník ukazatelů S a zásobník Booleovských hodnot SB *)
begin
  while Uk<>nil do begin
    Push(S,Uk);
    PushB(SB, true);
    Uk:=Uk^.LUk
end;
procedure PostOrder(Uk:TUk; L:TList);
(* Vlastní průchod PostOrder se dvěma zásobníky *)
var
  Zleva:Boolean;   (* indikátor návratu zleva *)
begin
  ListInit(L); SInit(S); SInitB(SB);
  NejlevPost(Uk);
  while not SEmpty(S) do begin
    Top(S,Uk);
    TopB(SB, Zleva); PopB(SB);
    if Zleva
    then begin (* přichází zleva, půjde doprava *)
      PushB(SB, false);
      NejlevPost(Uk^.Puk)
    end else begin (* přichází zprava, odstraní a zpracuje otcovský uzel *)
      Pop(S);
      InsertLast(L,Uk^.Data)
    end (* if *)
  end (* while *)
end;
```

```
procedure ZrusStrom(var Kor:TUkUz);
(* předpokládá se rušení neprázdného stromu, tedy kor<>nil *)
begin
  InitStack(S); (* inicializace zásobníku S pro ukazatele na uzel *)
  repeat  (* cyklus rušení *)
    if Kor=nil
    then begin
      if not SEmpty(S)
      then begin
        Kor:=TOP(S);
        POP(S);
      end; (* if not SEmpty *)

    end else begin (* jde doleva, likviduje a pravé strká do zásobníku *)
      if Kor^.PUK <>nil
      then PUSH(S,Kor^.PUK);
      PomPtr:=Kor; (* uchování dočasného kořene pro pozdější zrušení *)
      Kor:=Kor^.LUK; (* posud doleva *)
      dispose(PomPtr); (* zrušení starého uchovaného kořene *)
    end; (* Kor = nil *)
  until (Kor=nil) and SEMPTY(S)
end
```

Řazení

Select Sort

```
procedure SelectSort(var A:TA);
var i,j,PInd, PMin:integer;
begin
   for i:=1 to N-1 do begin
     PInd:=i;    (* Poloha pomocného minima*)
     PMin:=A[i];  (* Pomocné minimum*)
      for j:=i+1 to N do
          if PMin>A[j]
          then begin
              PMin:=A[j];
              PInd:=j
          end;
       A[i] :=: A[PInd]
  end (* for *)
end;
```

Bubble Sort

```
procedure BubbleSort(var A:TA);
var
   i,j: integer;
   Konec:Boolean;
begin
  i:=2;
  repeat
    Konec:=true;
    for j:=N downto i do (* porovnávání sousedních dvojic *)
      if A[j-1] > A[j]
      then begin
         A[j-1]:=: A[j]; (* výměna *)
         Konec:=false
      end;
      i:=i+1
  until Konec or (i=(N+1));
end;
```

Heap Sort

```
procedure HeapSort(var A:TA);
var
  i,Left,Right:integer;
begin (* Ustavení hromady *)
  Left:= N div 2; (* index nejpravějšího uzlu na nejnižší úrovni *)
  Right:=N;
  for i:= Left downto 1 do SiftDown(A,i,Right);
 (* Vlastní cyklus Heap-sortu *)
  for Right:=N downto 2 do begin
    A[1]:=:A[Right]; (* Výměna kořene s akt. posledním prvkem *)
    SiftDown(A,1,Right-1)   (* Znovuustavení hromady *)
  end; (* for *)
end; (* procedure *)
```

K Heap sort patří i:

```
procedure SiftDown(var A:TArr;Left,Right:integer);
(* Left je kořenový uzel porušující pravidla heapu, Right je velikost pole *)
var
  i,j:integer;
  Cont:Boolean;   (* Řídicí proměnná cyklu *)
  Temp:integer;  (* Pomocná proměnná téhož typu jako položka pole *)
begin
  i:=Left;
  j:=2*i;   (* Index levého syna *)
  Temp:=A[i];
  Cont:=j<=Right;
  while Cont do begin
    if j<Right
    then  (* Uzel má oba synovské uzly *)
      if A[j]< A[j+1]
      then  (* Pravý syn je větší *)
        j:=j+1;  (* nastav jako většího z dvojice synů *)
    if Temp >= A[j]
    then  (* Prvek Temp již byl posunut na své místo; cyklus končí *)
      Cont:=false
    else begin (* Temp propadá níž,  A[j] vyplouvá o úroveň výš *)
      A[i]:=A[j];   (*   *)
      i:=j;          (* syn se stane otcem pro příští cyklus"*)
      j:=2*i;        (* příští levý syn *)
      Cont:=j<=Right;  (* podmínka : "cyklus pokračuje" *)
    end (* if *)
  end;  (* while *)
  A[i]:=Temp;  (* konečné umístění prosetého uzlu *)
end;  (* procedure *)
```

Insert Sort

```
procedure BubbleInsertSort(var A:TArray);
var
  i,j:integer;
  Tmp:integer;
begin
  for i:=2 to N do begin
    Tmp:=A[i];
    A[0]:=Tmp; (* Ustavení zarážky *)
    j:=i-1;
    while Tmp<A[j] do begin (* hledej místo a posouvej prvek *)
      A[j+1]:=A[j];
      j:=j-1
    end; (* while *)
    A[j+1]:=Tmp; (* konečné vložení na místo *)
  end (* for *)
end; (* procedure *)
```

Quick Sort

```
procedure QuickSort(var A:TArr; left, right:integer);
(* Při volání má left hodnotu 1 a right hodnotu N *)
var i,j:integer;
begin
  partition(A,left,right,i,j);
  if left<j then QuickSort(A,left,j); (* Rekurze doleva *)
  if i<right then QuickSort(A,i,right);(* Rek. doprava *)
end;
```

```
procedure partition(var A:TArr; left,right:integer; var
i,j:integer);
var
  PM:integer; (* pseudomedián *)
begin
  i:=left;     (* inicializace i *)
  j:=right;    (* inicializace j *)
  PM:=A[(i+j) div 2]; (* ustavení pseudomediánu *)
  repeat
    while A[i] < PM do i:=i+1;
                        (* hledání prvního i zleva, pro A[i]>=PM  *)
    while A[j] > PM do j:=j-1;
                        (* hledání prvního j zprava pro A[j]<=PM  *)
    if i<=j
    then begin
      A[i]:=:A[j]; (* výměna nalezených prvků *)
      i:=i+1;
      j:=j-1
    end
  until i>j; (* cyklus končí, když se indexy i a j překříží *)
end; (* procedure *)
```

Shell Sort

```
procedure ShellSort(var A:TArr, N: integer);
var
  step,I,J:integer;
begin
  step:=N div 2; (* první krok je polovina délky pole *)
 while step > 0 do begin  (* cykluj, pokud je krok větší než 0 *)
   for I:=step to N-1 do begin (*cykly pro paralelní n-tice *)
     J:=I-step+1;
     while (J>=1) and (A[J]>A[J+step])do begin
          (* bublinový průchod *)
       A[J]:=:A[J+step];
       J:=J-step;   (* snížení indexu o krok *)
     end; (* while *)
   end; (* for *)
   step:=step div 2;   (* půlení kroku *)
 end; (* while *)
end; (* procedure *)
```