IAL-Algoritmy
Obsah 4. přednášky

- Seznamy - práce s ukazateli.
- ATD seznam - jednosměrný, dvousměrný, kruhový
- Seznamy s hlavičkou, bez hlavičky, operace,
- Principy implementace.
- Seznam a rekurze

Zdroje ke studiu

1. **Soubor ADS04.pdf na adrese**

2.Soubory: SeznamJednosmresene.pdf,SeznamyDvojsmResene.pdf, Seznamy-Kruh aj.pdf

V souborech jsou pascalovské a textové podoby modulů s operacemi nad jednosměrným seznamem (Let_un), dvojsměrným seznamem DLet_un) a modul s některými ostatními operacemi (OsLet_un). Uvedeny jsou i řídicí (hlavní) programy využívající tyto tři moduly a jejich výsledky při spuštění.

```
type
  TUk=^TElem; (* Typ uk. na prvek jednosm. sezn. *)
  TElem = record     (* Typ prvku seznamu *)
     Data:TEl;     (* Datové složky elementu *)
     UkNasl:TUk;   (* Ukazatel na následníka *)
  end;
  TList=record       (* ADT seznam *)
    Act,             (* Ukazatel na aktivní prvek seznamu *)
    Frst : TUk       (* Ukazatel na první prvek seznamu *)
  end;
```

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

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 *)
```

```
procedure First(var L:TList);
begin
  L.Active:=L.First  (* První se stane aktivním;
         v prázdném seznamu beze  změny *)
end; (* First *)
```

```
function Active(L:TList):Boolean;
begin
  Active:=L.Act<>nil
end; (* Active *)
```

```
procedure PostInsert (var L:Tlist; El:TEl);
var
   PomUk:Tuk;
begin
   if L.Act <> nil then begin  (* Operace se
        provede 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;
```

```
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;

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

```
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; (* Ukaz na ruseneho *)
         L.Act^.UkNasl:=PomUk^.UkNasl; (* Preklenuti
                                ruseneho*)
         Dispose(PomUk)
      end (*if  L.Act^.UkNasl<> *)

     end (*if  L.Act <> *)
end;
```

Seznam s hlavičkou

- Hlavička je první prvek seznamu, který má pomocnou funkci a není skutečným prvkem seznamu.
- Prázdný seznam obsahuje pouze hlavičku
- Seznam lze dočasně opatřit hlavičkou, která se v závěru odstraní
- Hlavička odstraňuje zbytečný prolog pro první prvek, před cyklem opakovaných operací nad ostatními prvky seznamu

```
procedure CopyList (LOrig:TList; var LDupl:TList);
(* Procedura s využitím ADT TList nad prvky integer a jeho operací *)
var
   El:integer;
begin
   InitList(LDupl);
   First(LOrig);
   if  Active(LOrig)
   then begin     (* vytvoření prvního prvku se provede  před cyklem *)
       Copy(LOrig, El);
       InsertFirst(LDupl,El);
       succ(LOrig);
        First(LDupl);
      (* vytvoření zbytku seznamu se provede v cyklu *)
        while Active(LOrig) do begin
            Copy(LOrig,El);
            PostInsert(LDupl,El);
            Succ(LOrig);
            Succ}LDupl);
         end (* while *)
    end (* if *)
end; (* procedure *)
```

```
procedure CopyList( LOrig:TList; var LDupl:TList);
(* Procedura s využitím ADT TList s hlavičkou
    nad prvky integer a jeho operací *)
var
  El:char;
begin
  InitList(LDupl); (* prázdná kopie *)
  InsertFirst(LDupl,0); (* vložení hlavičky s hodnotou 0 *)
  First(LOrig);  (* první originálu nastav na aktivní *)
  First(LDupl);  (* nastav hlavičku na aktivní *)
    (* vytvoření celého seznamu se provede v cyklu *)
     while Active(LOrig) do begin
        Copy(LOrig,El);
        PostInsert(LDupl,El);
        Succ(LOrig);
        Succ(LDupl);
      end (* while *)
  DeleteFirst(LDupl); (* odstranění hlavičky  *)
end; (* procedure *)
```

Pozn: Všimněme si, že bez použití aktivity
v duplikátním seznamu lze následujícím cyklem
vytvořit kopii, v níž jsou prvky v obráceném pořadí:

```
while Active(LOrig) do begin
      Copy(LOrig,El);
      InsertFirst(LDupl,El);
       Succ(LOrig)
 end;
```

Dvojsměrný seznam

Symetrické operace :

DInsertFirst	DInsertLast
DDeleteFirst	DDeleteLast
DFirst	DLast
DSucc	DPred
DCopyFirst	DCopyLast
DPostInsert	DPreInsert
DPostDelete	DPreDelete

Ostatní operace:

DListInit, DCopy, DActualize, DActive

```
procedure CopyDoubleList (DLOrig:TDList; var DLDupl:TDList);
var
   El:TData;
begin
   DInit(DLDupl);
   DFirst(DLOrig);
   while DActive(DLOrig) do begin
      DCopy(DLOrig,El);
      DSucc(DLOrig);
      DInsertLast(DLDupl,El)   (* Díky  vkládání na
                               konec je kopírování jednoduché *)
   end (* while *)
end; (* procedure *)
```

Podobně snadné je vkládání nebo rušení nalezeného prvku nebo před a za nalezeným prvkem.

Implementace operací nad ADT dvousměrný seznam

Pro implementaci ADT dvojsměrný seznam zaveďme typy

```
type
  TUkPrv=^TPrv;
  TPrv= record
              Data:TData;
              LUk,PUk:TUkPrv
          end;
  TDList=record
        Frst, Lst, Act :TUkPrv;    (* ukazatele na začátek a
                konec seznamu, a na aktivní prvek seznamu *)
        (* PocPrv: integer;  Možno použít pro počitadlo
                                  prvků seznamu*)
  end;
```

```
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
      L.Frst^.LUk:=PomUk;     (* Aktuální první bude
                                 ukazovat doleva na nový prvek*)
   else    (* Vkládám do prázdného seznamu *)
       L.Lst:=Pomuk;
   L.Frst:=PomUk;          (* Korekce ukazatele začátku *)
end;
```

```
procedure DDeleteFirst (var L:TDlist);
(* Nutno sledovat zda 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 seznamu*)
          L.Frst^.LUk:= nil;  (* Není-li sezn.prázdný, první doleva uk. nil *)
       end;
       Dispose(PomUk)
   end; (* if *)
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 uk.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 *)
```

Pozn. Všimněte si stranové symetrie s PostDelete

```
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 Act *)
        Dispose(PomUk)
     end; (*if  L.Act^.Luk <> nil *)
   end; (* if  L.Act<> nil *)
end; (* procedure *)
```

Kruhový seznam

Kruhový (cyklický) seznam lze vytvořit z lineárního seznamu tak, že se ustaví pravidlo, že následníkem posledního prvku je prvek první.

V implementační doméně to znamená, že ukazatel posledního prvku ukazuje na první prvek.

Ze sémantického pohledu nemá kruhový seznam začátek ani konec. Musí mít ale přístup alespoň k jednomu prvku seznamu, který má pozici pracovního počátku.

Kruhový seznam může být jedno nebo dvojsměrně vázaný (jednosměrný nebo dvojsměrný).

Soubor operací pro ADT kruhový seznam lze odvodit ze souboru operací nad ADT seznam.

Sémantiku operace obdobné First lze vyložit jako ustavení aktivity na ”první” prvek - prvek, jehož prostřednictvím je umožněn přístup ke kruhovému seznamu.

Soubor operací je nutno doplnit o možnost testu na průchod celým seznamem. Jednou z možností je zavedení počitadla prvků, které umožní, aby se kruhovým seznamem pohybovalo s využitím počítaného cyklu. Jinou možností je zavedení predikátu který vrátí hodnotu true, když se v seznamu posuneme na poslední (znovu první) prvek.

Problém k zamyšlení:

Navrhněte vhodný soubor operací nad ADT kruhový seznam takový, kterým lze provádět všechny nejznámější operace nad kruhovým seznamem jako:

- vytvoření kruhového seznamu z lineárního seznamu
- průchod kruhovým seznamem
- vytvoření kopie kruhového seznamu
- zrušení kruhového seznamu
- ekvivalence dvou kruhových seznamů

(Pozor! Poloha pracovního „prvního“ prvku není pro ekvivalenci významná!

Vyřešte uvedené úkoly s využitím abstraktních operací nad ADT kruhový jedno a/nebo dvojsměrně vázaný seznam i s využitím pascalovských ukazatelů.

Předávání aktivity seznamu

- Aktivita se považuje za pracovní stav a nepoužívá se obvykle k předávání informace
- Lze ale zavést operace:
  - procedure MarkAct (var List:TList);
  - procedure SetActMarked (var List:TList);
  - function IsMarkUsable (List:TList):Boolean;

Některé operace nad jednosm. seznamem souboru Let_un

```
procedure Clear(var List:TList);
function lengthList(List:Tlist):TCardinal;
procedure MarkAct (var List:TList);
procedure SetActMarked (var List:TList);
function IsMarkUsable (List:TList):Boolean;
procedure FindAndRewrite(List:Tlist; Klic,Elem:TData);
procedure FindAndPostInsert(var List:Tlist; Klic,Elem:TData);
procedure FindAndDelete(var List:TList;Klic:TData);
procedure FindAndPostDelete(var List:TList; Klic:TData);
function EquList(L1,L2:TList) :Boolean;
function EquListRec(L1,L2:TList):Boolean;
function FirstListLess(L1,L2:TList):Boolean;
function FirstListLessRec(L1,L2:TList):Boolean;
```

```
procedure ListCopy(List1:TList; var List2:TList);
procedure ListCopyRec(L1:TList; var L2:TList);
procedure FindMaxSeq(L:TList; var Num,Len:TCardinal);
procedure Match(L,SubL:TList; var Pos:TCardinal);
procedure InsertSubList(var L:TList; SubList:TList;
                                        Num:TCardinal);
procedure DeleteSublist(var L:TList;  Num,Len:TCardinal);
procedure CopySublist(L:TList; var SubL:TList;
                                        Num,Len:TCardinal);
procedure Concat(var L1,L2:TList);
procedure FindAndDecat(var L1,L2:TList; Klic:TData);
procedure MergeLists(var LSource1,LSource2,LDest:TList);
procedure DecatListToSeq(SourceList:Tlist; var DestList:TUkLL);
procedure InitListOfLists(LL:TUkLL);
procedure ClearListOfLists(var LL:TUkLL);
function LengthListOfLists(LL:TUkLL):TCardinal;
procedure CopyNthList(LL:TUkLL; Num:TCardinal; var L:TList);
```

Některé operace nad dvojsm. seznamem souboru DLet_un

```
type
   TUk=^TPrvek;  (* Ukazatel na prvek jednosm. seznamu *)
   TData=integer;  (* Datova sl. prvku *)
   TDUk=^TDPrvek; (* Ukazatel na prvek
              dvojsmerneho seznamu *)
   TUkLL=^TListOfLists;
               (* Ukazatel na seznam
                  seznamu          *)
    TCardinal=0..MaxInt;
    TPrvek= record   (* Typ prvku jedno smerneho seznamu *)
        Uk:TUk;
        Data:TData;
     end; (* record *)
```

```
TDPrvek=record (* Typ prvku dvousmerneho seznamu *)
        LUk,PUk:TDUk;
        Data:TData;
 end; (* record *)
 TList=record (* ATD jednosm. seznam *)
        Zac,Kon,Act,Mark:TUk;
        MarkUsable:Boolean;
        Len:TCardinal;
  end; (* record  *)
```

(* Do Mark se zaznamena stav aktivity operaci Mark.
Promenna MarkUsable je pro test pouzitelnosti operace ActSetMarked; nastavi se na false, rusi-li se zaznamenany prvek.
Len je delka seznamu, zde nevyuzivana..*)

```
TDList=record  (* ATD dvojsm. seznam *)
     Zac,Kon,Act,Mark:TDUk;
     MarkUsable:Boolean;
     Len:TCardinal;
  end; (* record *)

  TListOfLists=record (* typ seznam dvojsměrných seznamů *)
     Next:TUkLL;
     List:TDList;
   end;
```

```
procedure DMarkAct(var List:TDList);
procedure DSetActMarked(var List:TDList);
function DIsMarkUsable(List:TDList):Boolean;
procedure DListCopy(DL1:TDList; var DL2:TDList);
procedure DFindAndRewrite(DList:TDlist;
                  Klic,Elem:TData);
procedure DFindAndPostInsert(var DList:TDlist;
                  Klic,Elem:TData);
procedure DFindAndDelete
                 (var DList:TDList;Klic:TData);
procedure DFindAndPostDelete
                 (var DList:TDList; Klic:TData);
procedure DFindAndPreInsert
                 (var DList:TDList; Klic,Elem:TData);
procedure DFindAndPreDelete
                 (var DList:TDList; Klic:TData);
procedure DClear(var List:TDList);
```

Ukázky operací nad kruhovým seznamem souboru OsLet_un

```
procedure InitListHead(var List:Tlist);
function LengthListHead(List:TList):TCardinal;
procedure InsertFirstHead(var List:TList; Elem:TData);
procedure FindAndDeleteHead(var List:TList;Klic:TData);
procedure DeleteFirstHead(List:TList);
(*.....  atd  .....*)
(* Operace pro kruhovy seznam s hlavickou *)
procedure InitListCircHead (var List:TList);
procedure InsertFirstCircHead (var List:TList;Elem:TData);
procedure FirstCircHead(var List:TList);
procedure SuccCircHead(var List:TList);
function LengthListCircHead(List:TList):TCardinal;
procedure ListCopyCirc(List1:TList; var List2:TList);
procedure Copy(List:TList;  var Elem:TData);
```

Rekurzívní definice

Délka seznamu:

Je-li seznam prázdný, má délku nula. V jiném případě je jeho délka 1 plus délka zbytku seznamu.

Ekvivalence dvou seznamů:

Dva seznamy jsou ekvivalentní, když jsou oba prázdné nebo když se rovnají jejich první prvky a současně jejich zbytky.

Relace dvou seznamů:

Výsledek operace „první seznam je menší než druhý seznam“ je „true“ když první seznam je prázdný a druhý neprázdný *nebo* když první prvek prvního seznamu je menší než první prvek druhého seznamu *nebo* když *první prvek zbytku* prvního seznamu je roven *prvnímu prvku zbytku* druhého seznamu *a současně* zbytek prvního seznamu je menší než zbytek druhého seznamu. Jinak je výsledek operace „false“.

Co musíte umět:

- Grafy syntaxe jedno i dvojsměrného seznamu
- Implementace operací nad jednosměrně i dvojsměrnými seznamy
- Implementace vyšších operací s využitím ukazatelů, s hlavičkou i bez
- Implementace vyšších operací s využitím výhradně abstraktních operací nad ADT jedno a dvojsměrný seznam, bez hlavičky i s hlavičkou

- Rekurzívní implementace vybraných operací (ekvivalence, délka, relace uspořádání).
- Návrh repertoáru abstraktních operací pro kruhový seznam.
- Ekvivalence dvou kruhových seznamů (neboli, lze pootočit jedním kruhovým seznamem položeným na druhý, stejně dlouhý seznam tak, aby se jejich hodnoty rovnaly?)