IAL Algoritmy
2. přednáška

- **Dynamické přidělování paměti (DPP) - bez regenerace, s regenerací s využitím zřetězeného zásobníku**
- **Uživatelská implementace DPP (UDPP):**
  - **– bez regenerace (v poli s operacemi init, new, neúčinné dispose, mark a release),**
  - **–s regenerací (zřetězený zásobník). Implementace dynamických struktur v nepascalovském prostředí s využitím uživatelského DPP.**

Dynamické přidělování paměti

zdroj: Opora str. 31

- Statická a dynamická alokace (umístění) údajů v paměti
- operace new a dispose
- DPP bez regenerace – „stánek na paměť s papírovými kelímky“
- DPP s regeneraci – „výčep s pamětí s umývanými sklenicemi“

Statické a dynamické proměnné

- Statické proměnné (struktury) dostávají jméno při deklaraci. Prostor jimi zaujímaný se vyhradí (alokuje) v době překladu. K obsahu těchto proměnných se dostáváme prostřednictvím jejich jména.
- Je-li struktura statická, nemění se za běhu programu ani počet ani uspořádání jejich komponent.
- Příkladem statické struktury je pole nebo záznam.

- Dynamické proměnné (struktury) vznikají i zanikají v době běhu programu. Nemohou mít jméno (identifikátor). K obsahu dynamických proměnných (struktur) se dostáváme prostřednictvím ukazatele.

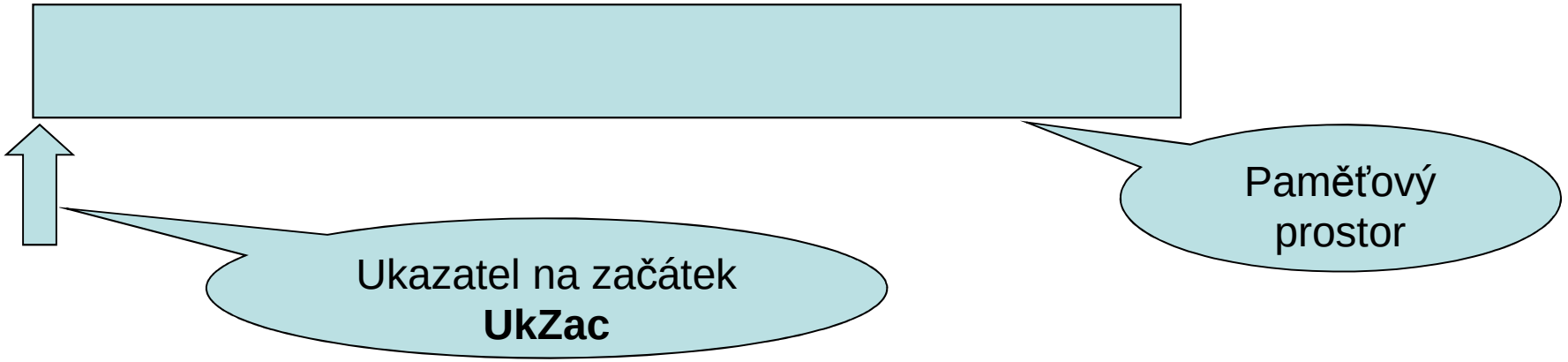
- Počet i uspořádání komponent dynamických struktur se za běhu programu mění.

- Ke tvorbě dynamické struktury je vhodné (nutné) použít datového typu „záznam“. Jeho heterogennost umožňuje, aby dynamický prvek obsahoval vedle vlastní hodnoty také
- Příkladem dynamické struktury je seznam nebo stromová struktura ukazatel(e).

Operace New

- Operace new(Uk) přiřadí ukazateli Uk hodnotu „ukazující“ na paměťový prostor, jehož velikost odpovídá datovému typu s nímž je ukazatel Uk svázán. Paměťový prostor pro operaci „new“ se bere z jisté paměťové oblasti, vyhrazené pro tento účel.
- Hodnota ukazatele lze přiřadit jen ukazateli téhož typu.
- Nil je ukazatelová konstanta s hodnotou vyjadřující, že ukazatel neukazuje na žádnou proměnnou (strukturu). Tuto konstantu lze přiřadit ukazateli libovolného typu.

Operace New „bez regenerace“

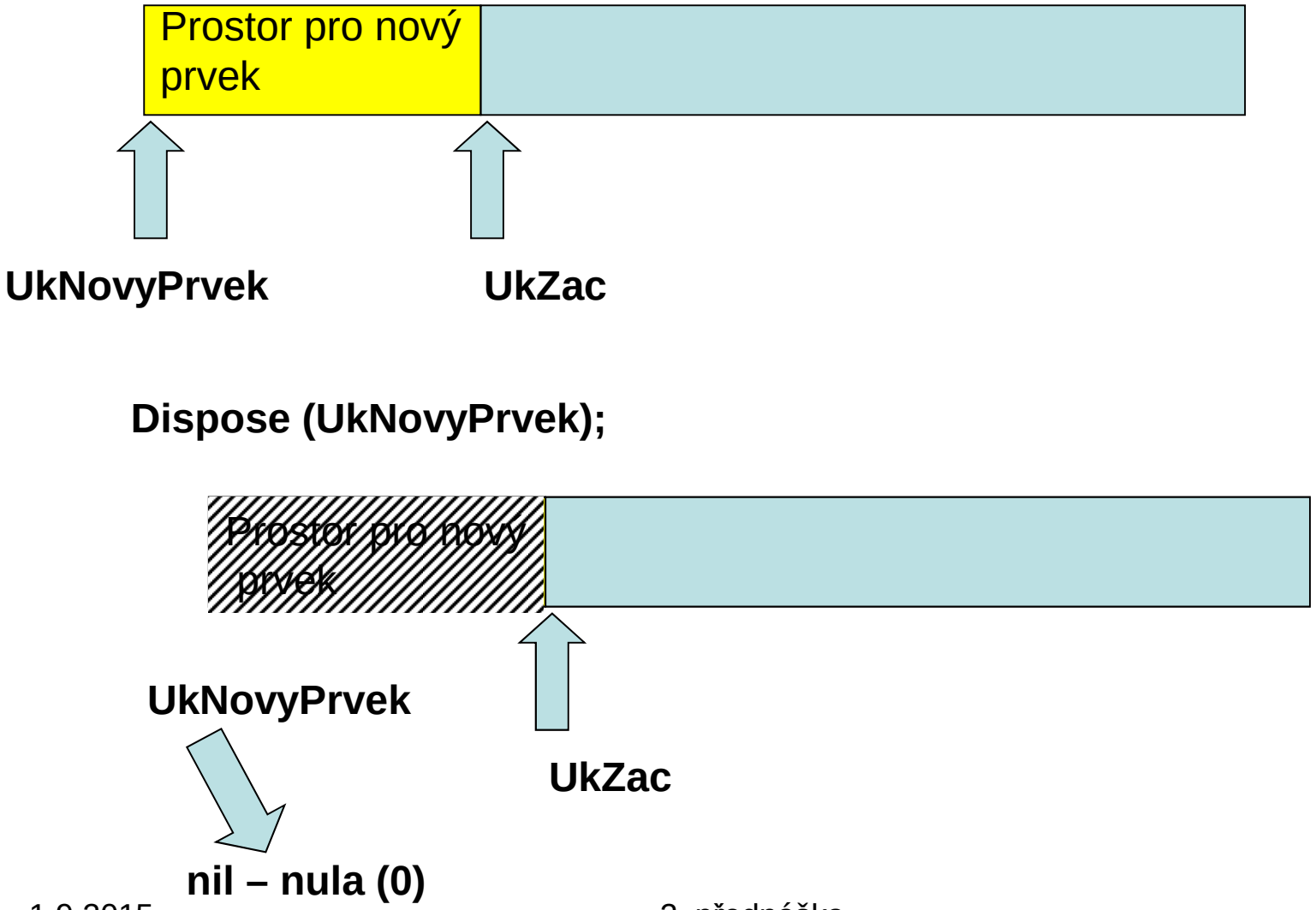


New (UkNovyPrvek);

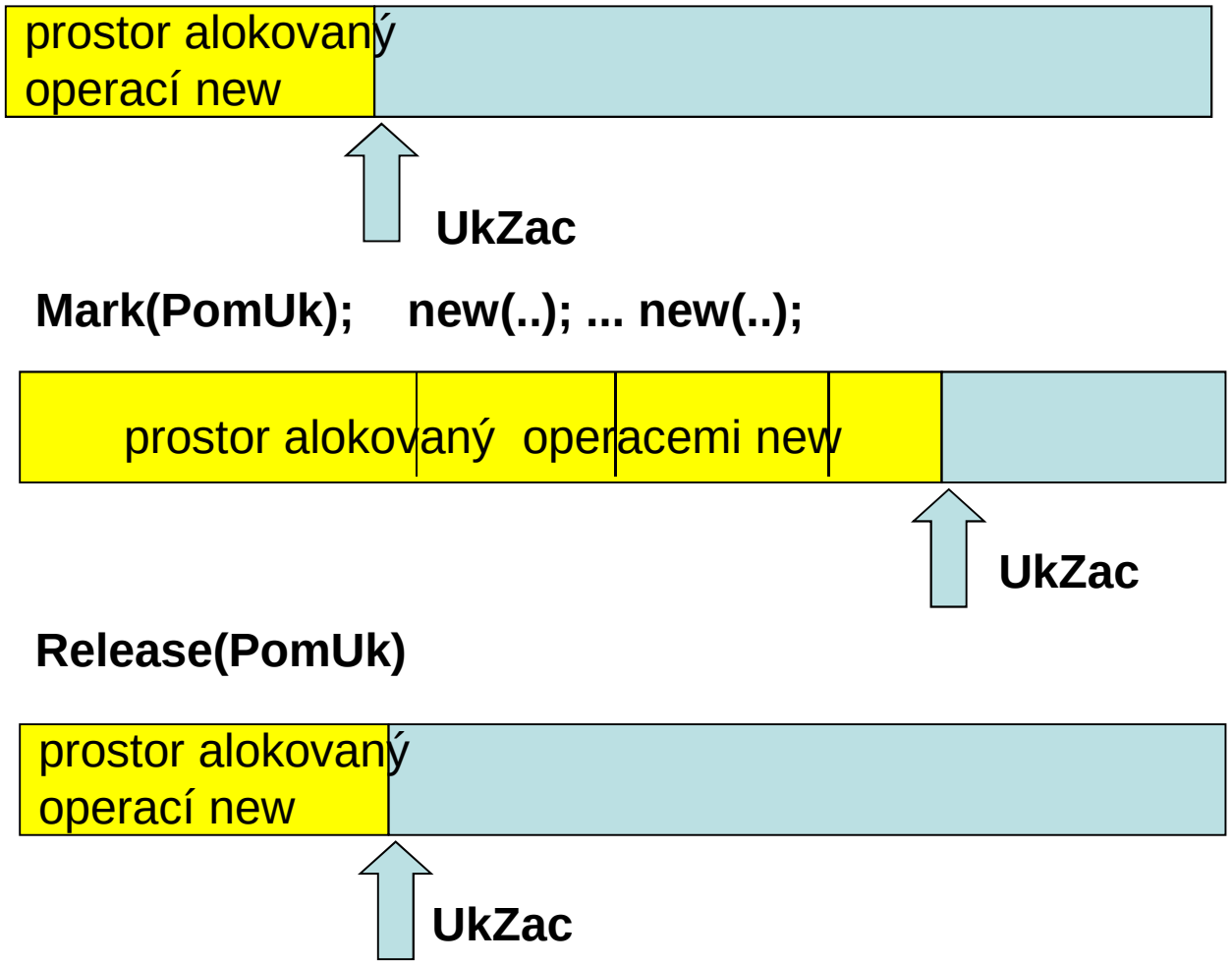
Operace Dispose

- Operace Dispose(Uk) ruší použitelnost dynamické proměnné (struktury), na kterou ukazuje ukazatel Uk. Po této operaci je hodnota ukazatele nedefinovaná.
- Pokud se paměť, kterou zaujímá zrušená proměnná (struktura) vrátí do vyhrazené paměťové oblasti, říkáme, že DPP pracuje **s regenerací**. V jiném případě jde o mechanismus **bez regenerace**.

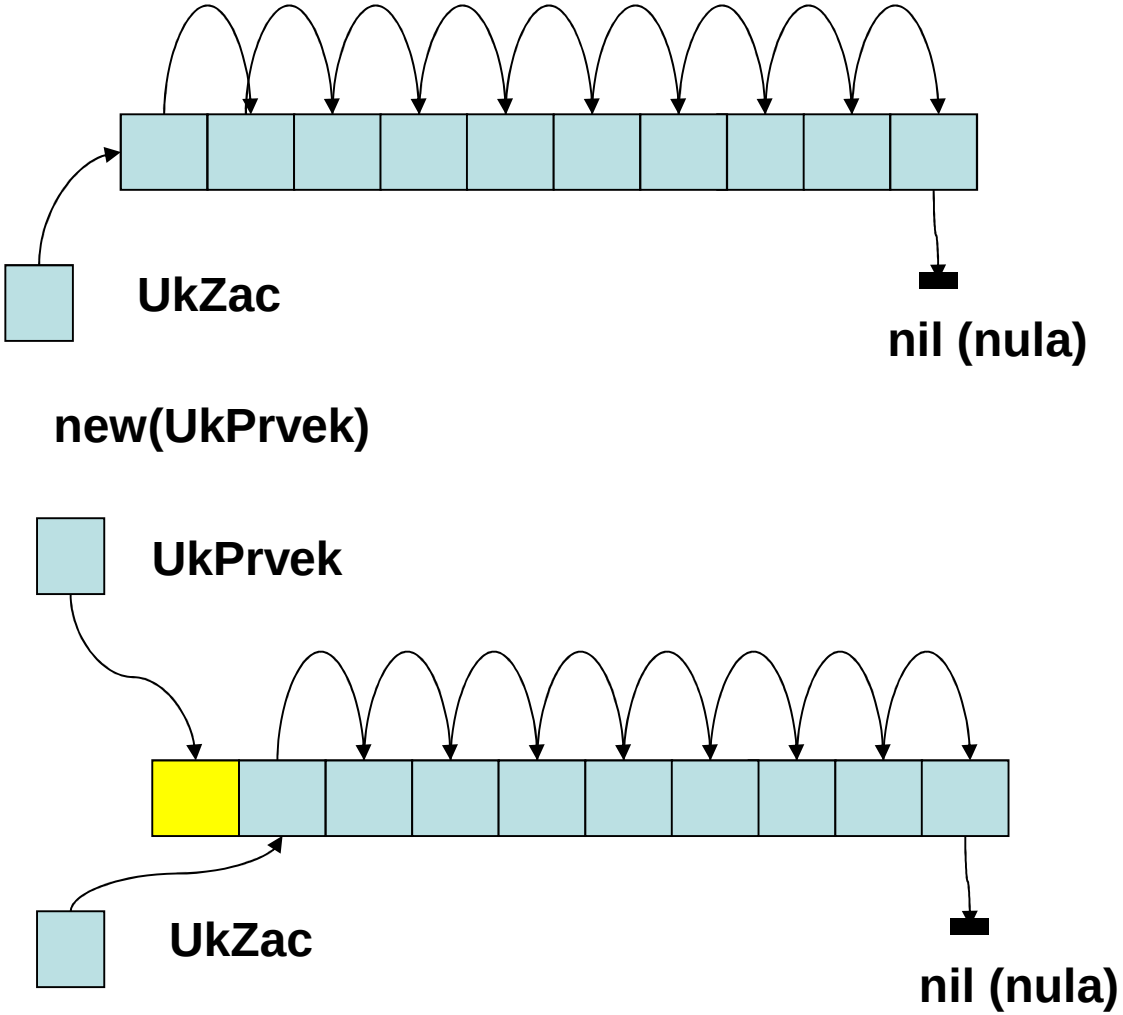
Operace Dispose „bez regenerace“



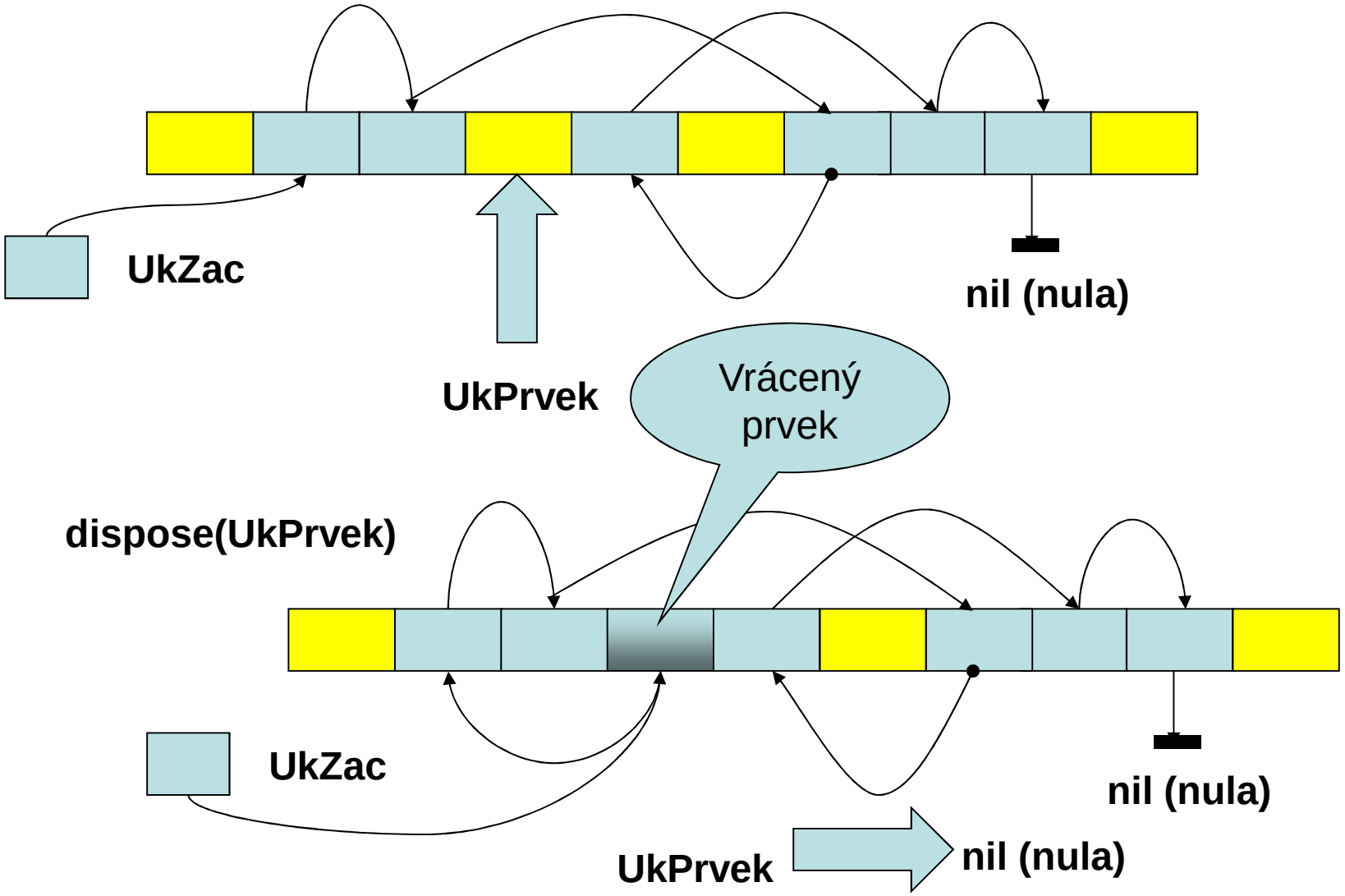
Operace Mark a Release



DPP s regenerací
(zjednodušený model)



Po čase... new...dispose



Implementace UDPP polem – bez regenerace

```
const HeapLength=100;
type
(* DPP (Dynamic Memory Allocation)
     pro max. 100 prvků *)
  TDMA = record
    FreePtr, (* Ukazatel(index)prvního volného *)
    MarkPtr: (* Ukazatel pro „mark“*)integer;
    ArrItem: (* dynamický prostor*)
             array [1..HeapLength] of TItem;
  end; (* record *)
var
  DMA:TDMA; (* globalní proměnná *)
```

```
procedure InitDMA;
(* Procedura nastaví počáteční hodnoty indexových
ukazatelů. Musí se vyvolat před použitím
ostatních operací nad UDPP *)
begin
  with DMA do begin
    FreePtr:=1;
  end
end;
```

```
procedure NewDMA(var Ptr:integer);
(* Procedura vrací indexový ukazatel prvního
volného prvku pole, který bude přidělen jako
dynamický prostor *)

begin
  with DMA do begin
    Ptr:=FreePtr;   (* Vracený indexový
                       ukazatel *)
    FreePtr:=FreePtr+1;  (* Zvýšení hodnoty
     ukazatele na začátek o „velikost prvku“ *)
  end;
end;
```

Procedura dispose – zde prázdná operace

```
procedure DisposeDMA(Ptr:integer);
(* Tato procedura je prázdná, protože nepoužíváme
indikátor zrušeného prvku *)
begin
end;
```

Operace Mark a Release

```
procedure MarkDMA(var MarkPtr:integer);
(* Uschování hodnoty FreePtr do MarkPtr *)
begin
  with DMA do begin
    MarkPtr:=FreePtr;
  end;
end;
procedure ReleaseDMA(MarkPtr:integer);
(* Znovuustavení uschované hodnoty do FreePtr *)
begin
  with DMA do begin
    FreePtr:=MarkPtr;
  end
end;
```

Implementace UDPP s regenerací s použitím zřetězeného pole na způsob zásobníku

```
type
  const
     LenghthOfHeap=100;
     NilDMA=0;  (* Nula reprezentuje nil*)
  TDMA = record
    StartPtr : integer;
    ArrItem: array [1..LengthOfHeap] of TItem;
    ArrOfLinks:array[1..LengthOfHeap] of integer;
                  (* pole spojovacích ukazatelů *)
  end; (* record *)

var
  DMA:TDMA;
```

```
procedure InitDMA;
(* Procedura spojuje prvky do seznamu.Ukazatel
StartPtr ukazuje na první prvek. Poslední
ukazuje „nikam“ – nil reprezentovaný nulou. *)
var
  i:integer;
begin
  with DMA do begin
    for i:=1 to LengthOfHeap-1 do
      ArrOfLinks[i]:=i+1;
        (* průběžné spojení prvků *)
    ArrOfLinks[LengthOfHeap]:=NilDMA;
        (* nastavení posledního na nil *)
    StartPtr:=1;
         (* nastavení ukazatele na začátek *)
  end; (* with *)
end;
```

```
procedure NewDMA(var Ptr:integer);
begin
  with DMA do begin
    Ptr:=StartPtr;   (* vrací ukazatel prvního *)
    StartPtr:=ArrOfLinks[StartPtr]  (* Ukazatel
                  prvního se posune na další *)
  end
end;

procedure DisposeDMA(Ptr:integer);
begin
  with DMA do begin
    ArrOfLinks[Ptr]:=StartPtr;  (* Ukazatel
            disposovaného prvku je nastaven na
                              aktuálního prvního *)
    StartPtr:=Ptr;    (* actualizace prvního *)
  end
end;
```

Implementace dynamických struktur

Skutečné DPP		**Simulované či „uživatelem definované DPP"**
Zápis s ukazatelem		**Zápis s indexem**
Ptr je ukazatel		**PtrI je index**
nil	**=>**	**NilDMA (* const NilDMA=0 *)**
Ptr	**=>**	**PtrI**
Ptr^	**=>**	**ArrItem[PtrI]**
Ptr^.RPtr	**=>**	**ArrItem[PtrI].RPtr**
Ptr^.RPtr^.LPtr	**=>**	**ArrItem[ArrItem[PtrI].RPtr].LPtr**

Ukázka implementace operace rušení prvku seznamu s pascalovským ukazatelem

```
type
  TListPtr=^TITem; (* typ ukazatele na prvek seznamu *)
  TItem=record              (* typ prvek seznamu *)
    data:char;
    LPtr,RPtr:TListPtr   (* levý a pravý ukazatel prvku  *)
  end;
  TList=record               (* type seznam - List *)
    Frst,Lst:TListPtr      (* Ukazatele na první a poslední
                           prvek seznamu *)
  end;
```

```
procedure deleteDMA(var L:TList; Ptr:TListPtr);
(* Ptr ukazuje na rušený prvek. Procedura používá pascalovské ukazatele  *)
begin
  if Ptr<>nil then begin  (* je ukazatel rušeného nenilový? *)
     if (Ptr=L.Frst) and (Ptr=L.Lst)
     then begin    (* rušený je jediným prvkem *)
         L.Frst:=nil;  L.Lst:=nil; (* rušení jediného *)
     end else begin (* rušený není jediným prvkem*)
       if (Ptr=L.Frst)
       then begin (* rušený je prvním prvkem *)
         L.Frst:=Ptr^.RPtr;
          Ptr^.RPtr^.LPtr:=Ptr^LPtr; (* :=nil *)
       end else begin
         if (Ptr=L.Lst)
         then begin (* rušený je posledním prvkem *)
           L.Lst:=Ptr^.LPtr;
           Ptr^.LPtr^.RPtr:=Ptr^.RPtr (* :=nil*)
         end else begin (* rušený má oba sousedy *)
           Ptr^.LPtr^.RPtr:=Ptr^.RPtr;
           Ptr^.RPtr^.LPtr:=Ptr^.LPtr
         end (* Ptr=L.Lst *)
       end (* Ptr=L.Frst *)
     end; (* (Ptr=L.Frst) and (Ptr=L.Lst) *)
     dispose(Ptr);
  end (* Ptr<>nil *)
end;
```

Ukázka implementace operace rušení prvku seznamu s indexovým ukazatelem v poli

```
const
  NilUDMA=0;  (* Uživatelský nil, representovaný nulou *)
type
  TListPtr=integer;  (* typ indexový ukazatel *)
  TItem=recod   (* typ položky  dvojsměrného seznamu *)
    data:char;
    LPtr,RPtr:TListPtr
  end;
  TList=record   (* typ dvojsměrný seznam definovaný ukazateli
                    na začátek a konec *)
    Frst,Lst:TListPtr
  end;
```

ArrItem – je jméno pole, ve němž je realizován dynamický prostor

```
procedure deleteDMA(var L:TList; Ptr:TListPtr);
(* Ptr ukazuje na rušený prvek.Procedura s indexovými ukazateli a UDPP *)
begin
  if Ptr<>NilUDMA then begin  (* je ukazatel rušeného nenilový? *)
    if (Ptr=L.Frst) and (Ptr=L.Lst)
    then begin   (* rušený je jediným prvkem *)
      L.Frst:=NilUDMA;  L.Lst:=NilUDMA;  (* rušení jediného *)
    end else begin (* rušený není jediným prvkem*)
      if (Ptr=L.Frst)
      then begin (* rušený je prvním prvkem *)
        L.Frst:=ArrItem[Ptr].RPtr;
         ArrItem[ArrItem[Ptr].RPtr].LPtr:=NilUDMA
      end else begin
        if (Ptr=L.Lst)
        then begin (* rušený je posledním prvkem *)
          L.Lst:=ArrItem[Ptr].LPtr;
          ArrItem[ArrItem[Ptr].LPtr].RPtr:=NilUDMA
        end else begin (* rušený má oba sousedy *)
          ArrItem[ArrItem[Ptr].LPtr].RPtr:=ArrItem[Ptr].RPtr;
          ArrItem[ArrItem[Ptr].RPtr].LPtr:=ArrItem[Ptr].LPtr
        end  (* Ptr=L.Lst *)
      end  (* Ptr=L.Frst *)
    end;  (* (Ptr=L.Frst) and (Ptr=L.Lst) *)
    dispose(Ptr);
  end  (* Ptr<>nil *)
end;
```