IAL Algoritmy
Obsah 2. přednášky

- Dynamické přidělování paměti
- Datové struktury
- ADT
  - syntaktická a sémantická specifikace,
  - diagramy signatury.
- ADT seznam
  - jednosměrný, dvousměrný, kruhový,
  - základní operace nad seznamy,
  - jednosměrný seznam - práce s ukazateli
  - seznam s hlavičkou
  - dvousměrný seznam - implementace

Statické 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

- 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.
- Funkce malloc() a free()

Dynamické proměnné

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

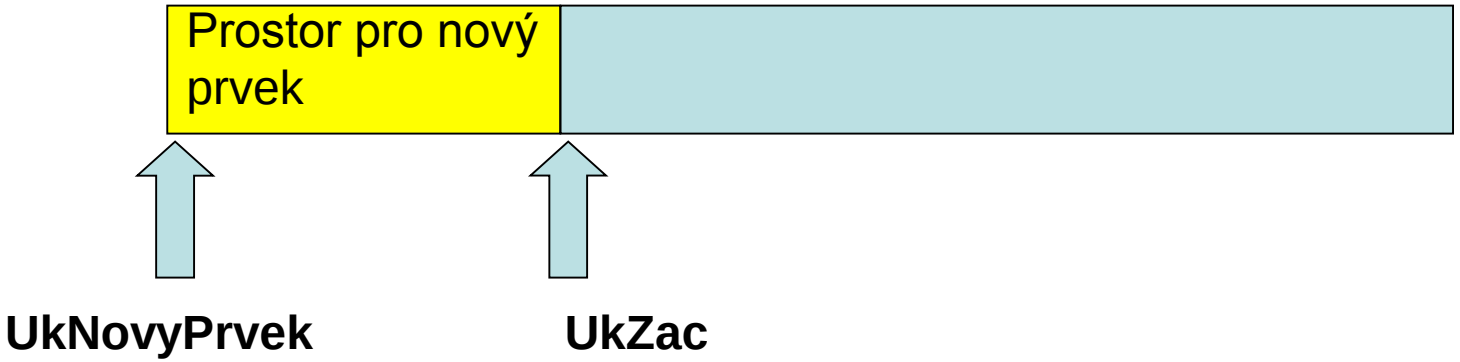
Operace malloc()

- Operace malloc() vrací hodnotu „ukazující“ na paměťový prostor. Jeho velikost je třeba určit parametrem této funkce – obvykle pomocí operátoru sizeof(). Vrácený ukazatel je vhodné přetypovat na ukazatel na datový typ s nímž je ukazatel svázán. Paměťový prostor pro operaci malloc() se bere z jisté paměťové oblasti, vyhrazené pro tento účel.
- NULL je ukazatelová konstanta s hodnotou vyjadřující, že ukazatel neukazuje na žádnou proměnnou (strukturu). Tuto konstantu lze přiřadit ukazateli libovolného typu.

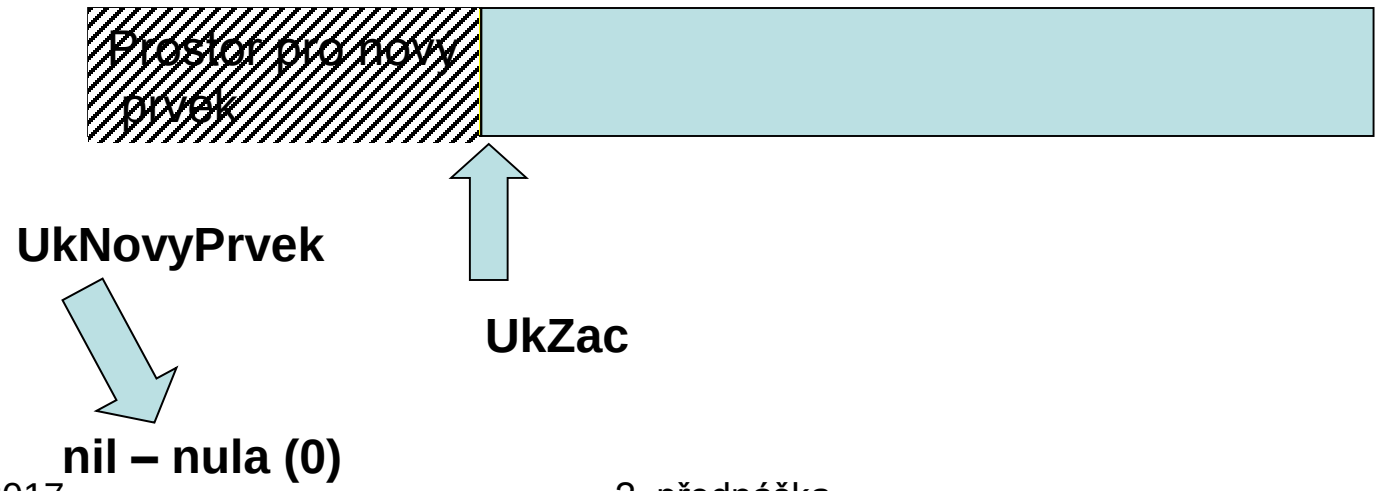
Operace free()

- Operace free(ptr) ruší použitelnost dynamické proměnné (struktury), na kterou ukazuje ukazatel ptr. 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**.

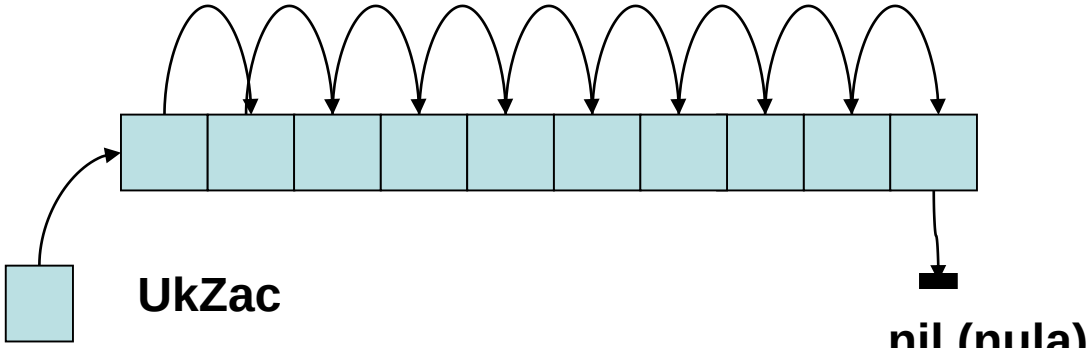
Operace free() „bez regenerace“



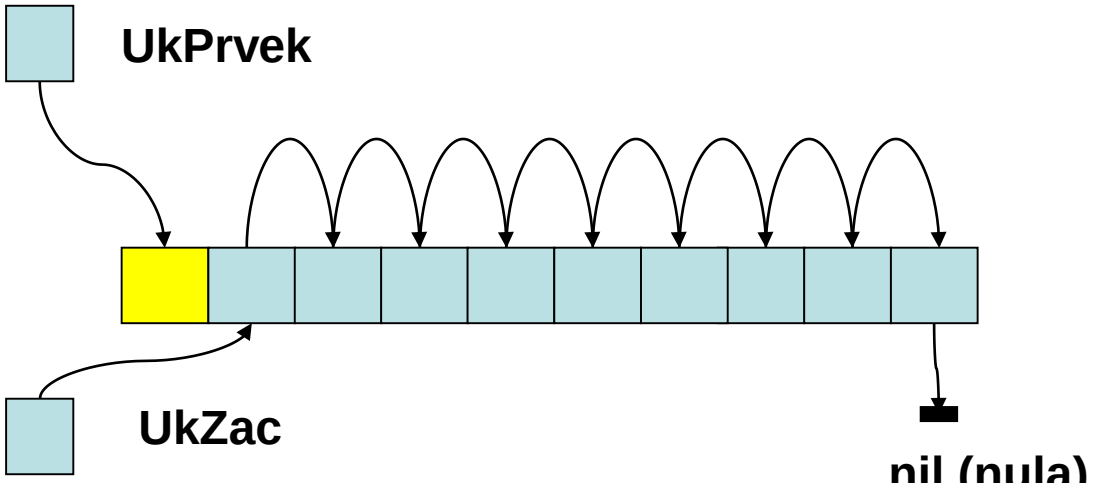
Dispose (UkNovyPrvek);



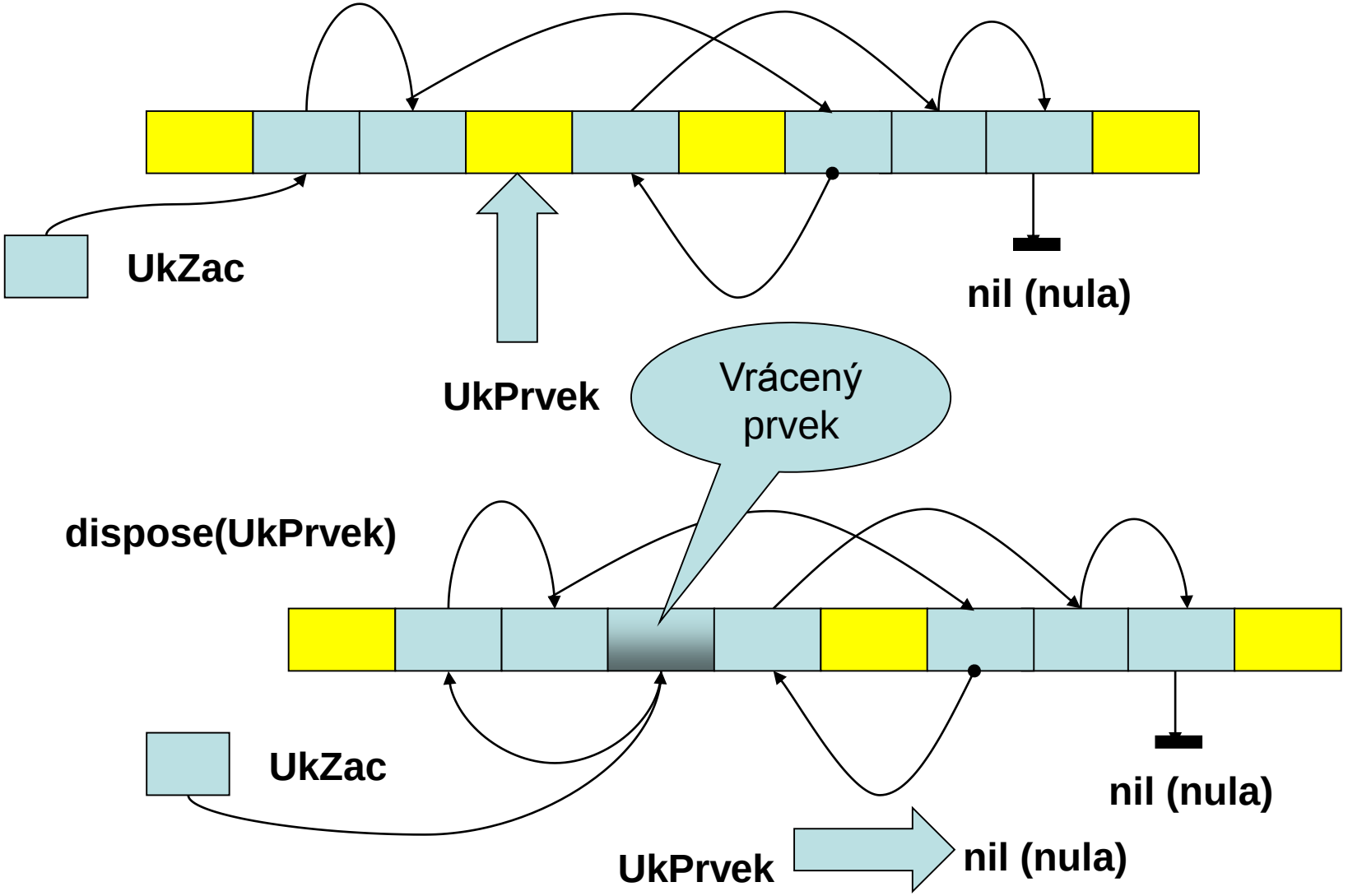
DPP s regenerací
(zjednodušený model)



new(UkPrvek)



Po čase... malloc()...free()



Datové struktury statické, dynamické, operace nad DS

- Datová struktura je **homogenní**, když všechny její prvky jsou téhož typu. Nejčastěji pak prvkům říkáme položky struktury.
- Datová struktura, jejíž prvky nejsou téhož typu ,je **heterogenní**. Jejím prvkům nejčastěji říkáme složky.
- Představitelem heterogenní (nehomogenní) datové struktury v Pascalu je záznam, v jazyce C sktruktura (struct). Ostatní pascalovské datové struktury jsou homogenní.
- Mezi základní operace nad datovou strukturou je operace přiřazení.

Konstruktor a selektor

- Konstruktor je operace, jejímiž vstupními parametry je výčet (všech) komponent struktury a výsledkem operace je datová struktura obsahující tyto komponenty. Konstruktor “vytvoří” datovou strukturu z jejich komponent. Příkladem konstruktoru v jazyce Pascal je textový řetězec v příkazu str=‘Textovy retezec’ nebo množina [1,3,5,7,9]
- Selektor je operace, která umožní přístup k jednotlivé komponentě datové struktury na základě uvedení jména struktury a zápisu přístupu (“reference”). Příklad selektoru nad textovým řetězcem z předcházejícího odstavce je zápis str[3], kde prvek řetězce má hodnotu ‘x’. Přístup k prvku datové struktury se používá za účelem změny hodnoty prvku nebo k získání jeho hodnoty.

Destruktor a iterátor

- Destruktor je operace nad dynamickými strukturami. Tato operace zruší dynamickou strukturu a vrátí prostor jí zaujímaný systému DPP.
- Iterátor je operace, která provede zadanou činnost nad všemi prvky homogenní datové struktury. Příklad: většina složitých grafických útvarů je realizována seznamem grafických elementů, z nichž je útvar sestaven. Operace, která jediným příkazem typu iterátor provede operaci "vykreslit" nad všemi elementy, zobrazí útvar jako celek.

Abstraktní datový typ

Abstraktní datový typ (ADT) je definován množinou hodnot, kterých smí nabýt každý prvek tohoto typu a množinou operací nad tímto typem.

Smyslem ADT je zvýšit datovou abstrakci a snížit algoritmickou složitost programu (algoritmu).

Kardinalita datového typu vyjadřuje množství různých hodnot ADT. Je výrazem paměťové náročnosti ADT.

ADT zdůrazňuje „co dělá“ a potlačuje „jak to dělá“. ADT připomíná „černou skříňku“

Příklady elementárních ADT : typ real.

Syntax a sémantika

Syntaxe vyjadřuje pravidla korektního zápisu jazykové konstrukce.

Sémantika vyjadřuje (popisuje) účinek (význam) zápisu jazykové konstrukce.

Algebraická signatura typu Kladný integer - Posint

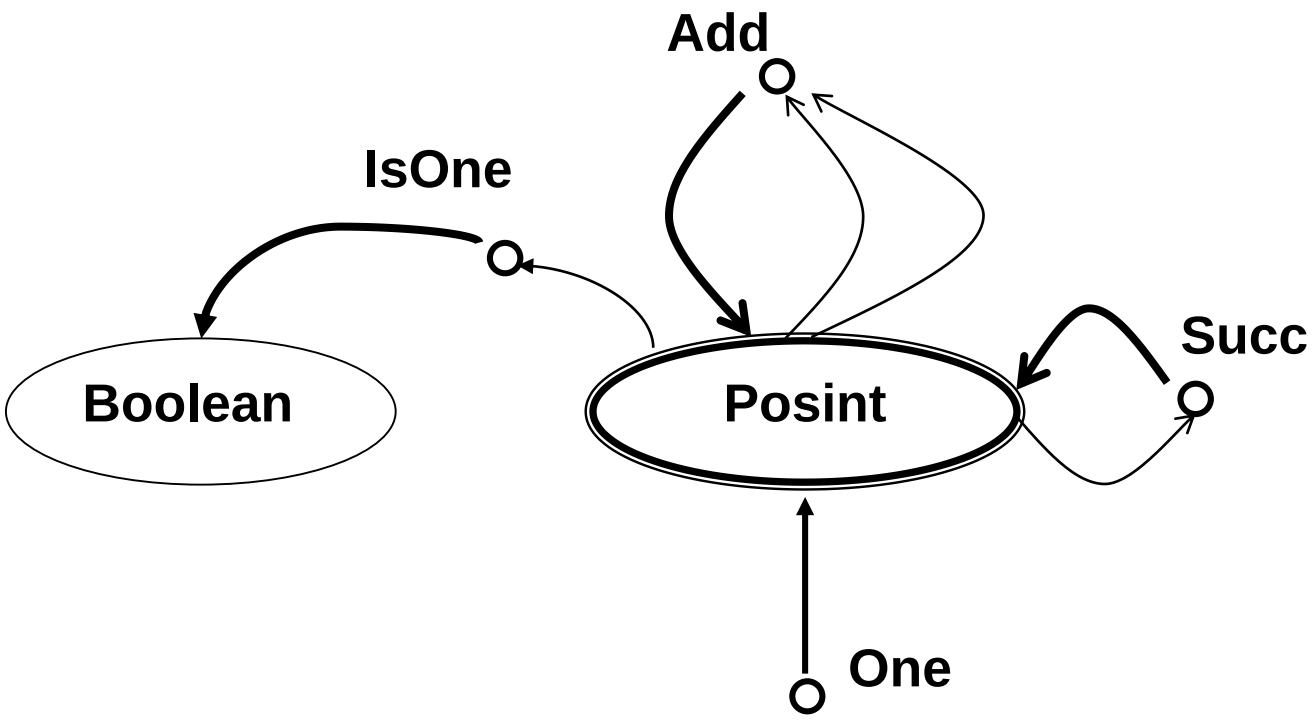
One : → Posint (Této operaci se říká generátor nebo "inicializace". Musí se použít před první operací nad typem

Add : Posint x Posint → Posint

Succ : Posint → Posint

IsOne : Posint → Boolean

Diagram signatury ADT Posint



Sémantika operace

Slovní sémantika:

- Operace One ustaví hodnotu typu Posint rovnu jedné. Tato operace je inicializace typu (generátor).
- Operace Add vytvoří aritmetický součet dvou prvků typu Posint.
- Operace Succ vytvoří hodnotu následující danou hodnotu (hodnotu o jednu větší).
- Operace (predikát) IsOne nabude hodnoty true, pokud je argument hodnota rovna jedné. V jiných případech má operace hodnotu false.

Axiomatická specifikace sémantiky

1. Add(X,Y) = Add(Y,X)
2. Add(One,X)=Succ(X)
3. Add(Succ(X),Y)=Succ(Add(X,Y))
4. IsOne(One)=true
5. IsOne(Succ(X))=false.

Jiné způsoby specifikace sémantiky

- Slovní popis
- Operační (funkční popis) – popis prostřednictvím procedur/funkcí
  - Nevýhoda -> Podkládá způsob implementace
- Specifikace úplným výčtem účinků

ADT seznam (angl. List)

Seznam je lineární, homogenní, dynamická datová struktura. Patří mezi nejobecnější datové struktury.

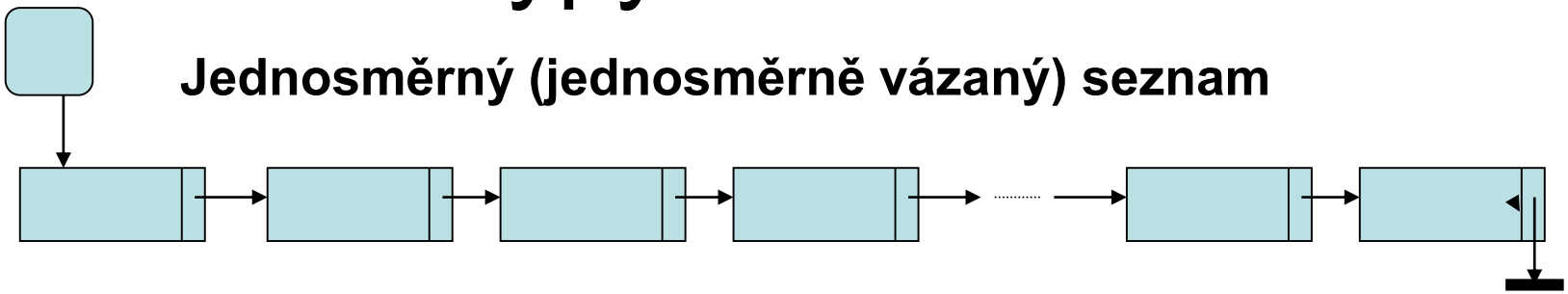
Lineárnost znamená, že každý prvek struktury má právě jednoho předchůdce (*predecessor*) a jednoho následníka (*successor*). Výjimku tvoří první a poslední prvek.

Prvkem seznamu může být libovolný jiný datový typ – také strukturovaný. Např. seznam seznamů.

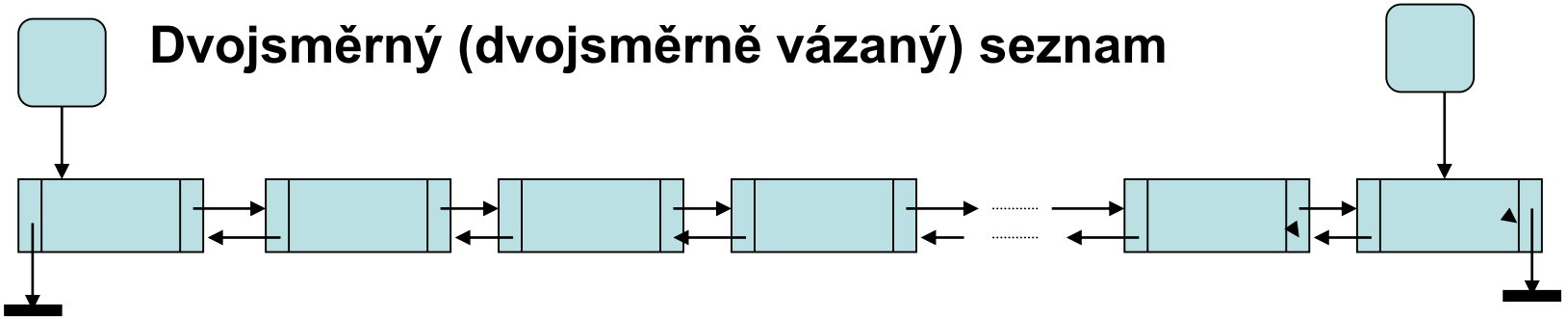
Seznam může být prázdný.

Typy seznamů

Jednosměrný (jednosměrně vázaný) seznam



Dvojsměrný (dvojsměrně vázaný) seznam



Kruhový jednosměrný seznam

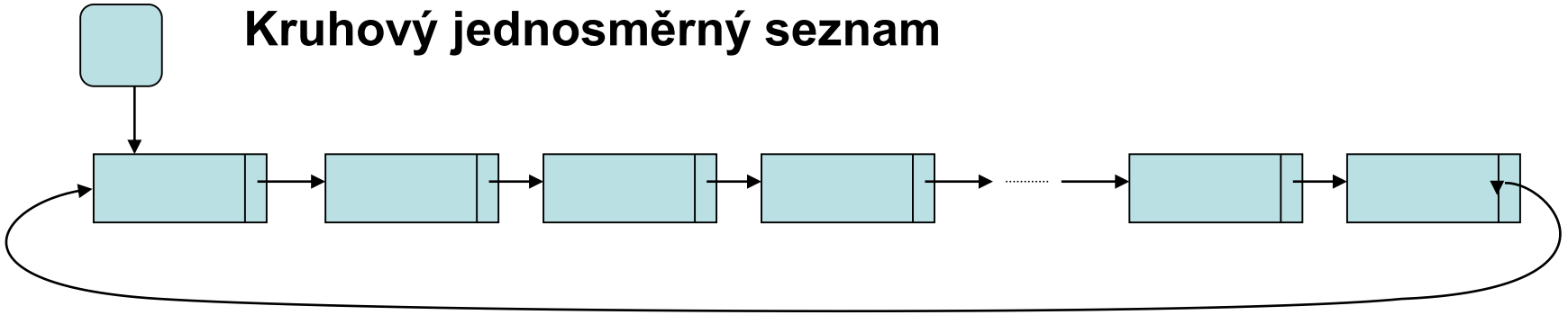
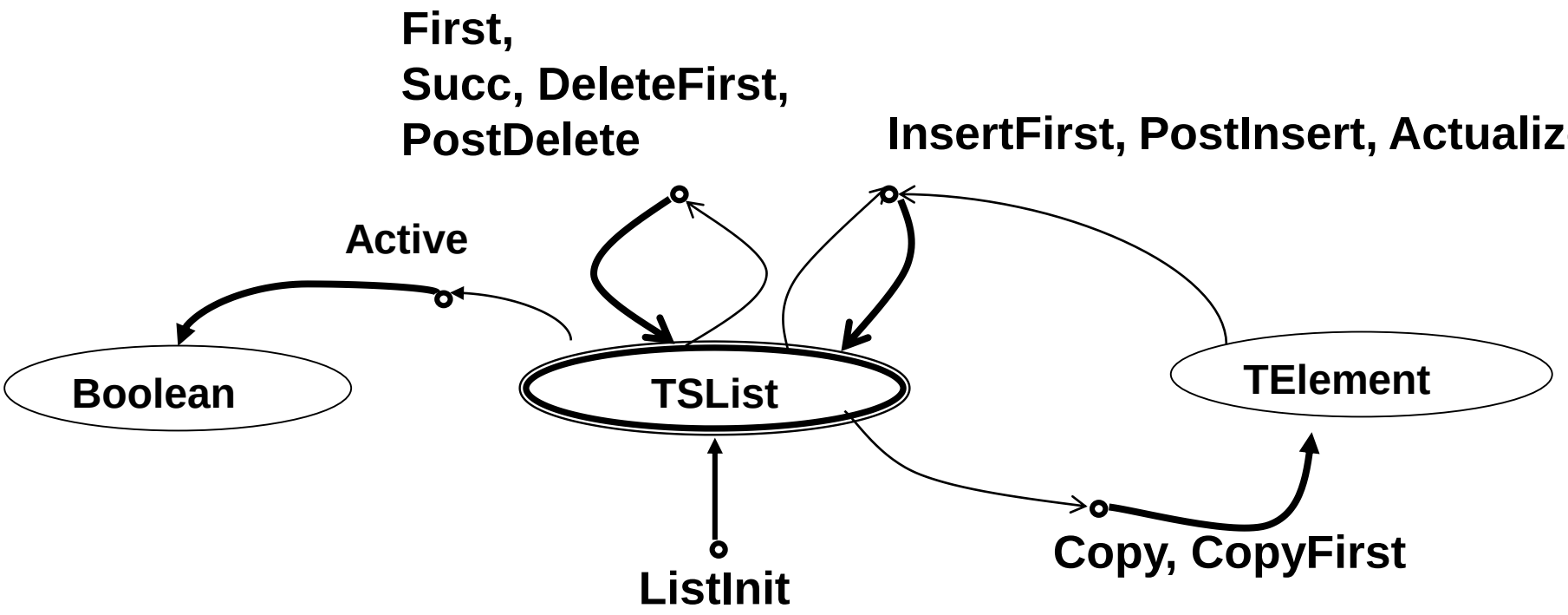


Diagram signatury ADT TList



Symetricky doplňkové operace pro dvousměrný seznam:
Last, Pred, DeleteLast, PreDelete, InsertLast, PreInsert, CopyLast

Sémantika operací nad jednosměrným seznamem

- **ListInit(L)** – vytvoří prázdný seznam prvků daného typu
- **InsertFirst(L,El)** – vloží element El do seznamu L jako první prvek (jediná operace, která může vložit prvek do prázdného seznamu)
- **CopyFirst(L,V)** – v proměnné V vrátí hodnotu prvního prvku. V případě prázdného seznamu CHYBA! (stav Error)
- **DeleteFirst(L)** – zruší první prvek seznamu

- **First(L)** – první prvek se stane aktivním (pro prázdný seznam bez účinku)
- **Succ(L)** – aktivita se přenese na následující prvek (pro prázdný seznam bez účinku, je-li aktivním poslední, aktivita se ztratí – seznam přestane být aktivním)
- **Copy(L,V)** – v proměnné V vrátí hodnotu aktivního prvku (v případě neaktivního seznamu CHYBA! – stav ERROR).
- **Actualize(L,El)** – hodnota aktivního prvku je přepsána hodnotou El. (V případě neaktivního seznamu bez účinku).

- **PostInsert(L, El)** – vloží prvek El jako nový za aktivní prvek (v případě neaktivního seznamu bez účinku)
- **PostDelete(L)** – zruší prvek za aktivním prvkem (v případě neaktivního seznamu nebo neexistence prvku za aktivním – bez účinku)
- **Active(L)** – predikát (Booleovská funkce) – vrací hodnotu true v případě, že seznam je aktivní; v jiném případě vrací hodnotu false

Symetrické operace pro dvojsměrný seznam

- **InsertLast(L, EL)** – vloží prvek El jako poslední (do prázdného seznamu současně i první – spolu s InsertFirst je to jediná operace, která může vložit prvek do prázdného seznamu)
- **CopyLast(L, V)** – V proměnné V vrátí hodnotu posledního prvku. V případě prázdného seznamu Chyba!!
- **Pred(L)** – posune aktivitu zpět

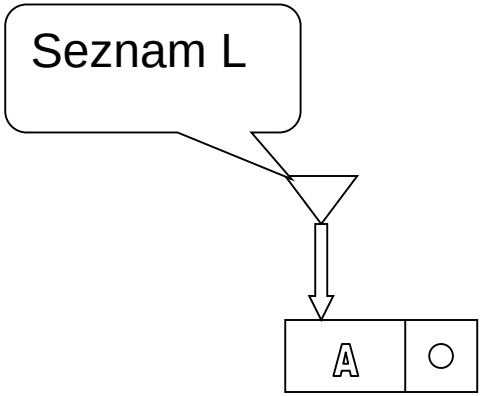
- **Last(L)** – nastaví aktivitu na poslední prvek. V případě prázdného seznamu bez účinku.
- **DeleteLast(L)** – zruší poslední prvek (V případě, že poslední prvek je současně jediný a první, vznikne prázdný seznam. Je s operací DeleteFirst jediná operace, která může odstranit jediný – poslední prvek. Pro prázdný seznam bez účinku).

- **PreInsert(L, El)** – vloží hodnotu prvku El před aktivní prvek. Je-li seznam neaktivní je operace bez účinku.

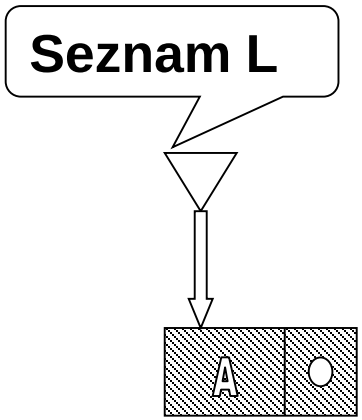
- **PreDelete(L)** – operace zruší prvek před aktivním prvkem. Je-li seznam neaktivní nebo před aktivním prvkem (nalevo od něj) není žádný prvek, je operace bez účinku.

Ukázka účinků operace s jednosměrným seznamem

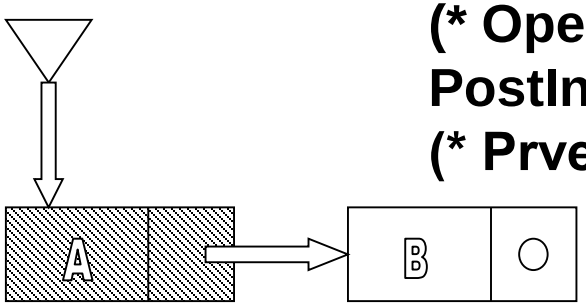
ListInit(L);
Operace vytvoří prázdný seznam.



Operace InsertFirst(L, A);

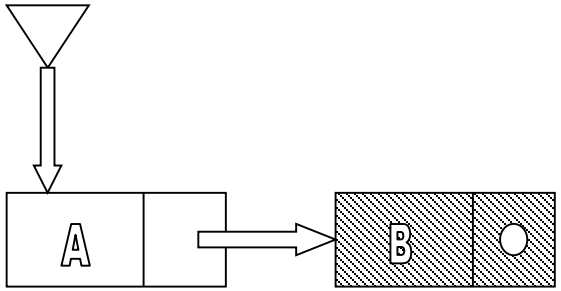


(* Operace *)
First (L);
(* První je aktivní *)

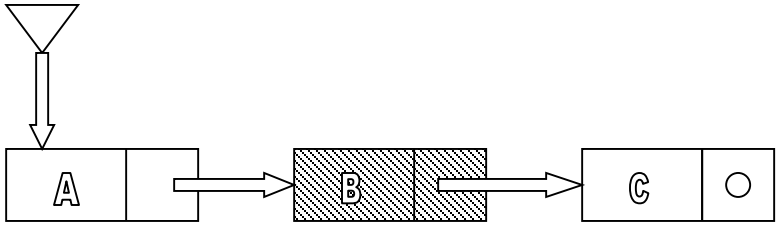


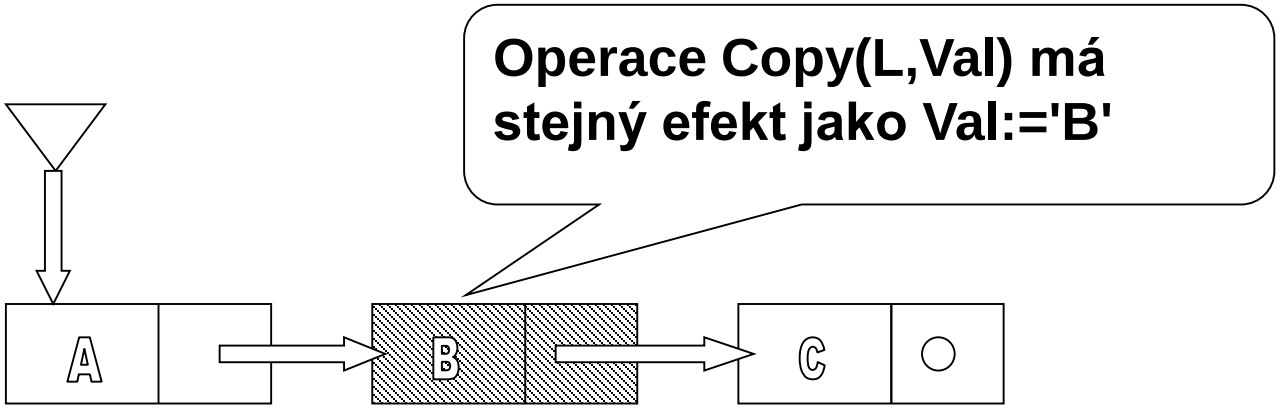
(* Operace *)
PostInsert(L,B);
(* Prvek B se vloží za aktivní *)

(* Operace *)
Succ(L);
(* posune aktivitu na další prvek*)

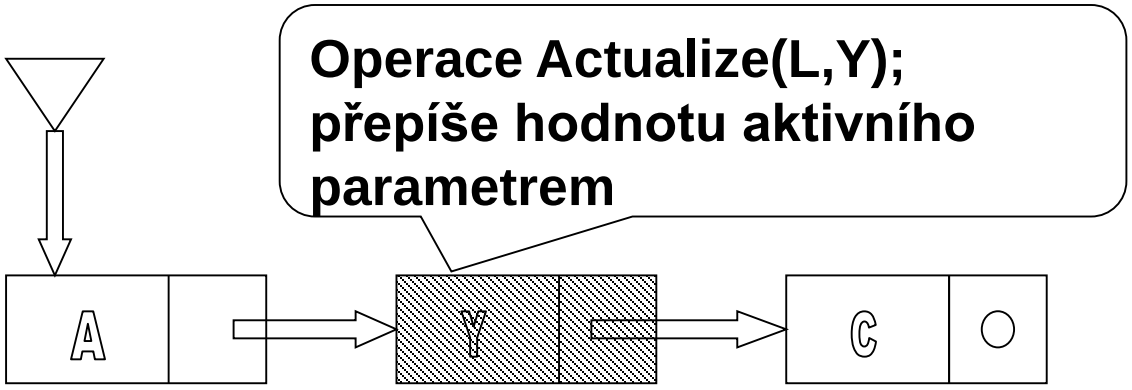


(*Operace*)
PostInsert(L,C)
(* vloží nový prvek C za aktivní *)



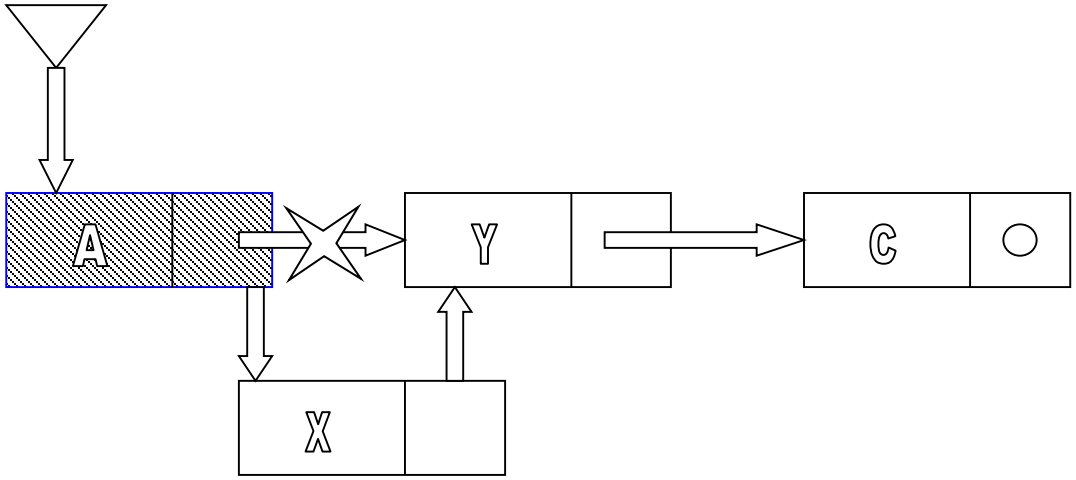


V případě neaktivního seznamu dojde k chybě !!!



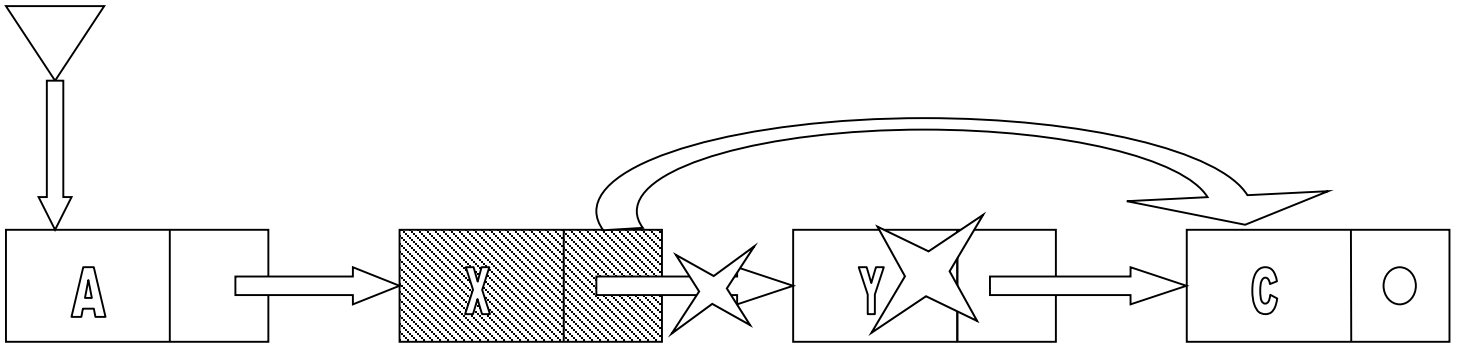
(* Dvojice operací *)

First(L); **(* Učiní první aktivním*)**
PostInsert(L,X) **(* Vloží X za aktivního*)**



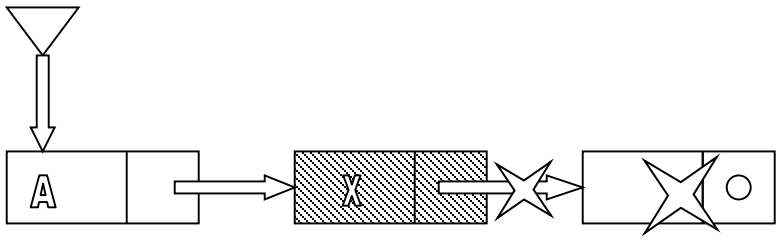
(* Dvojice operací*)

Succ(L); (* Posune aktivitu na další prvek *)
PostDelete(L); (* zruší prvek za aktivním *)

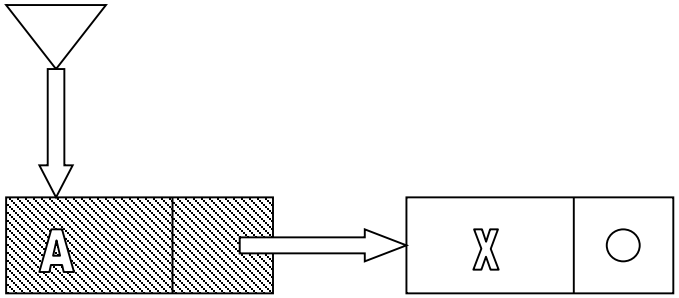


(* Operace*)

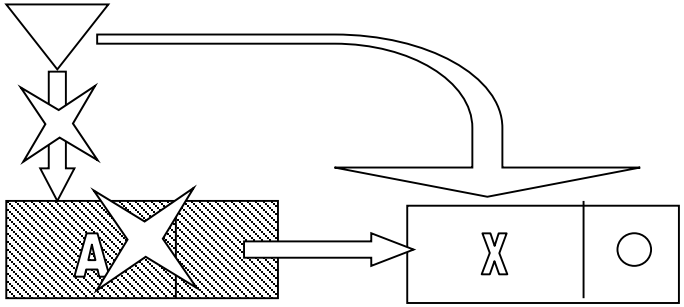
PostDelete(L); (* Zruší prvek za aktivním *)



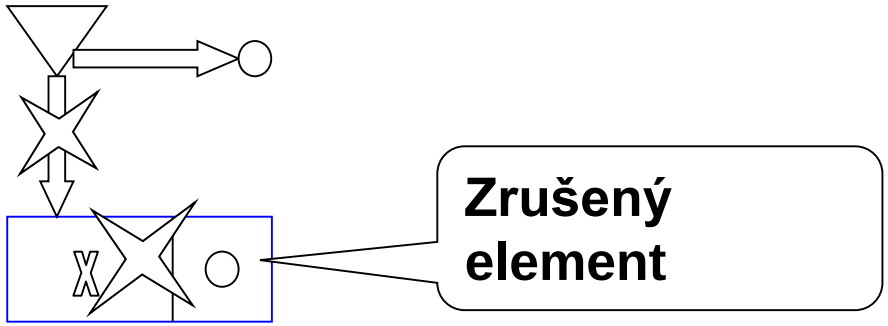
(* Operace *)
First(L); (* První je aktivní *)

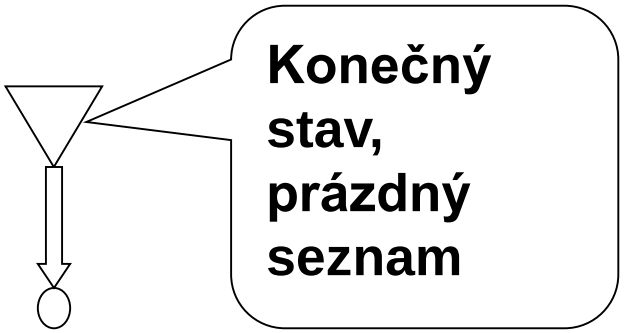


(* Operace *)
DeleteFirst(L) (* Zruší první prvek *)



(* Operace *)
DeleteFirst(L); (* Zruší první a jediný prvek.
Vytvoří prázdný seznam *)





Typické algoritmy nad ADT seznam

Typické operace nad seznamem:

1. Délka seznamu
2. Kopie (duplikát) seznamu
3. Zrušení seznamu
4. Ekvivalence dvou seznamů
5. Relace (lexikografická) dvou seznamů
6. Vkládání nových prvků do seznamu (na začátek, na pozici danou ukazatelem, pořadím, aktivitou, na konec)

7. Vkládání a rušení podseznamu
8. Vyhledávání prvku v seznamu
9. Rušení prvku seznamu (prvního, prvku na a za pozicí dané ukazatelem, pořadím, aktivitou, posledního)
10. Seřazení prvků seznamu podle velikosti (klíče)
11. Konkatenace (zřetězení) dvou a více seznamů (podseznamů) do jednoho seznamu (např. podseznamů obsahující textové „slovo“ do „věty“).
12. Dekatenace (rozčlenění) jednoho seznamu na podseznamy (např. rozčlenění textové „věty“ na „slova“)

Rekurzívní definice ekvivalence dvou seznamů

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

Algoritmy nad ADT List s použitím abstraktních operací (pouze!!!)

Délka seznamu:

```
int function Length(L:TList)
    Count ← 0
    First(L)
    while Active(L)do
      Count ← Count+1
      Succ(L)
    end while
    return (Count)
end function
```

Seznam – implementace operací (práce s ukazateli)

```
typedef struct telem                //typ prvku seznamu
{
      TData data;                   //datové složky elementu
      struct telem *nextPtr;        //ukazatel na následníka
}TElem;


typedef struct tlist      //ADT seznam
{
      TElem *first;       //ukazatel na první prvek seznamu
      TElem *act;         //ukazatel na aktivní prvek sez.
}TList;
```

```
void ListInit (TList *l)
{
  l->act = NULL;
  l->first = NULL;
} /*ListInit*/


void InsertFirst(TList *l,TData d)
{
  TElem *newElemPtr = (TElem *) malloc(sizeof(TElem));
                                   //vytvoření nového prvku
  if (newElemPtr == NULL) {
        printf("Chyba pri alokaci pameti \n");
        exit(1);
    }
  newElemPtr->data = d;          //nastavení datové složky
  newElemPtr->nextPtr = l->first;   //uk tam, kam začátek
  l->first = newElemPtr;      //Zač. ukazuje na nový prvek
} /*InsertFirst*/
```

```
void First (Tlist *l)
{
  l->act = l->first        /* První se stane aktivním;
                           v prázdném seznamu beze změny*/
}  /*First*/
```

```
bool Active (Tlist *l)
{
  return l->act != NULL
}  /*Active*/
```

```
void PostInsert (TList *l, TData d)
{
  if (l->act != NULL){
              //operace se provede jen pro aktivní seznam
    TElem *newElemPtr = (TElem *) malloc(sizeof(TElem));
    if (newElemPtr == NULL) {
        printf("Chyba pri alokaci pameti \n");
        exit(1);
    }
    newElemPtr->data = d;
    newElemPtr->nextPtr = l->act->nextPtr;
                          //nový ukazuje tam, kam aktivní
    l->act->nextPtr = newElemPtr;
                          // aktivní ukazuje na nového
  } //if aktivni
}
```

```
TData Copy (TList *l)
/* Operace předpokládá ošetření aktivity seznamu
ve tvaru: if Active(L){...Copy(&l)...}
Copy v neaktivním seznamu způsobí chybu */
{
  return (l->act->data);
}
```

```
void Actualize (TList *l, TData d)
{
  if (l->act != NULL) {
    l->act->data = d;
  }
}
```

```
void PostDelete (TList *l)
{
  TElem *elemPtr;
  if (l->act != NULL){
    if (l->act->nextPtr != NULL) {            //je co rušit
       elemPtr = l->act->nextPtr;   //ukazatel na rušeného
       l->act->nextPtr = elemPtr->nextPtr;    //překlenutí
       free(elemPtr);
    } //existuje rušený
  }  //aktivní seznam
}
```

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, LDupl)
/*Proc. s využitím ADT TList nad prvky integer a jeho operací*/
   InitList(LDupl);
   First(LOrig);
   if  Active(LOrig) then
       /*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
            Copy(LOrig,El);
            PostInsert(LDupl,El);
            Succ(LOrig);
            Succ(LDupl);
       end while
    end if
end procedure
```

```
procedure CopyList (LOrig, LDupl)
/* Procedura s využitím ADT TList s hlavičkou
    nad prvky integer a jeho operací */

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

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, DLDupl)
    DInit(DLDupl);
    DFirst(DLOrig);
    while DActive(DLOrig) do
        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:

```
typedef struct tdelem          //typ prvku seznamu
{
      TData data;              //datové složky elementu
      struct tdelem *rPtr;     //ukazatel na následníka
      struct tdelem *lPtr;     //ukazatel na předchůdce
}TDElem;

typedef struct tdlist    //ADT seznam
{
      TDElem *first;     //ukazatel na první prvek seznamu
      TDElem *last       //ukazatel na poslední prvek
      TDElem *act;       //ukazatel na aktivní prvek sez.
  /*  int numberOfEl;    - počítadlo prvků seznamu */
}TDList;
```

```
void DListInit (TDList *l)
{
   l->act = NULL;
   l->last = NULL;
   l->first = NULL;
   /*l->numberOfEl = 0;*/
} /*DListInit*/
```

```
void DInsertFirst (TDList *l, TData d)
{
  TDElem *newElemPtr = (TDElem *) malloc(sizeof(TDElem));
                  //zkontrolovat úspěšnost operace malloc!
  newElemPtr->data = d;
  newElemPtr->rPtr = l->first; //pravý nového na prvního
  newElemPtr->lPtr = NULL;  //levý nového ukazuje na NULL
  if (l->first != NULL) {    //seznam už měl prvního
    l->first->lPtr = newElemPtr;  //první bude doleva
                                  //ukazovat na nový prvek
  }
  else{             //vložení do prázdného seznamu
    l->last = newElemPtr;
  }
  l->first = newElemPtr;   //korekce ukazatele začátku
}
```

```
void DDeleteFirst (TDList *l)
{  //nutno sledovat, zda neruší aktivní prvek resp. jediný prvek
  TDElem *elemPtr;
  if (l->first != NULL){
    elemPtr = l->first;
    if (l->act == l->first){    //první byl aktivní
       l->act = NULL;           //ruší se aktivita
    }
    if (l->first == l->last){//seznam měl jediný prvek-zruší se
       l->first = NULL;
       l->last = NULL;
    }
    else {
      l->first = l->first->rPtr; //aktualizace začátku seznamu
      l->first->lPtr = NULL;  //ukazatel prvního doleva na NULL
    }
    free(elemPtr);
  }
}
```

```
void DPostInsert (TDList *l, TData d)
{  //nutno sledovat, zda nevkládá za poslední prvek
  if (l->act != NULL){  //je kam vkládat
    TDElem *newElemPtr =(TDElem *) malloc(sizeof(TDElem));
                          //zkontrolovat úspěšnost malloc!
    newElemPtr->data = d;
    newElemPtr->rPtr = l->act->rPtr;
    newElemPtr->lPtr = l->act;
    l->act->rPtr = newElemPtr;
                         //navázání levého souseda na nový
    if (l->act == l->last){     //vkládá za posledního
      l->last = newElemPtr;     //korekce ukazatele na konec
    }
    else{              //navázání pravého souseda na vložený prvek
      newElemPtr->rPtr->lPtr = newElemPtr;
    }
  } //aktivní
}
```

```
void DPostDelete (TDList *l)
{     // Nutno sledovat, zda neruší poslední prvek
  if (l->act != NULL) {
    if (l->act->rPtr != NULL) {    // Je co rušit?
      TDElem *elemPtr;
      elemPtr = l->act->rPtr;         // ukazatel na rušený
      l->act->rPtr = elemPtr->rPtr; // překlenutí rušeného
      if (elemPtr == l->last){        //rušený poslední
        l->last = l->act;            //posledním bude aktivní
      }
      else{   //prvek za zrušeným ukazuje doleva na Act
        elemPtr->rPtr->lPtr = l->act;
      }
      free(elemPtr);
    } //je co rušit
  } //aktivní
}
```

Pozn. Všimněte si stranové symetrie s PostDelete

```
void DPreDelete (TDList *l)
{     // Nutno sledovat, zda neruší první prvek
  if (l->act != NULL) {
    if (l->act->lPtr != NULL) {    // Je co rušit?
      TDElem *elemPtr;
      elemPtr = l->act->lPtr;         // ukazatel na rušený
      l->act->lPtr = elemPtr->lPtr; // překlenutí rušeného
      if (elemPtr == l->first){      //rušený první
        l->first = l->act;           //prvním bude aktivní
      }
      else{   //prvek před zrušeným ukazuje doprava na Act
        elemPtr->lPtr->rPtr = l->act;
      }
      free(elemPtr);
    } //je co rušit
  } //aktivní
}
```

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 ukazatelů.

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 prvního seznamu je roven prvnímu prvku druhého seznamu *a současně* zbytek prvního seznamu je menší než zbytek druhého seznamu. Jinak je výsledek operace „false“.

K procvičení:

• Do seznamu celých čísel seřazeného podle velikosti vložte nový prvek tak, aby seřazení seznamu zůstalo zachováno. V případě, že seznam obsahuje prvek rovný vkládanému, vložte vkládaný za shodný (poslední ze shodných).

• Ze seznamu celých čísel vyřaďte prvek zadaný parametrem. (Příklad: Vyřazení hodnoty 5 ze seznamu "1,3,6,8" nemá žádný účinek. Zrušení hodnoty 5 ze seznamu "1,3,5,7" má za výsledek seznam "1,3,7 ".) V případě, že seznam obsahuje více shodných prvků rovných zadanému, zrušte poslední z nich.

• V seznamu celých čísel nalezněte nejdelší neklesající posloupnost. Výsledkem bude počáteční index a délka nalezené posloupnosti. (Příklad: Výsledky pro seznam: "4,3,2,1" je začátek=1, délka=1. Výsledek pro seznam: 4,1,3,5,2,4,1,8,9" je začátek=2, délka=3;)
Nápověda: Algoritmus hledá maximální délku. Algoritmus uchovává index a délku každé kandidátské posloupnosti (posloupnosti delší než nejdelší z předcházejících).

• V jednom průchodu seznamem najděte průměrnou hodnotu a rozptyl (dispersi) délek všech neklesajících posloupností daného seznamu celých čísel.
Nápověda: Rozptyl D daného souboru hodnot je průměrná hodnota kvadratických odchylek všech hodnot souboru od průměrné hodnoty.

$$D = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2$$

• Jsou dány dva seřazené seznamy celých čísel. Sloučením z nich vytvořte jeden nový seznam seřazených celých čísel. (Příklad: Je dán seznam L1="1,3',5,7,9" a L2="2,3",4,6,8" . Výsledkem bude L3="1,2,3',3",4,5,6,7,8,9".)

Poznámka: Při práci s ukazateli k vytvoření nového seznamu použijte prvky zdrojových seznamů, které tím zaniknou. V případě práce s abstraktními operacemi nad ADT List zachovejte seznamy L1 a L2 a vytvořte nový seznam L3.