**Seznamy - realizace**

Popis problému:

Máme dlouhatánskou, které chybí část, kterou tedy chceme přidat.

Ve větě chybí slovo „větu“

Máme dlouhatánskou větu, které chybí část, kterou tedy chceme přidat.

Jaký datový typ bude použit pro uložení věty?
Jak budeme realizovat přidání chybějícího textu?
Bude stačit paměť?

Ve větě se nám vyskytlo „netechnické“ slovo – potřebujeme jeho část vynechat.

Máme dlouhatánskou větu, které chybí část, kterou tedy chceme přidat.

Jaký datový typ zvolíme pro uložení tohoto typu dat? (Text, Věta, Slovo)
Jak budeme realizovat vyjmutí části textu?
Jak bude vypadat hospodaření s pamětí?

Základní datové struktury – charakterizují organizaci dat v paměti

- lineární seznam
- strom

Seznam (kontejner)

- skládá se z prvků (Node, uzel, element, ...)
- každý prvek má dvě části – datovou a část odkazující na další prvek

Ukázka realizace seznamu v C:

```
struct TLinSeznam {
double iData; // hodnota node, obecně jakýkoli počet, jakýkoli typ
struct TLinSeznam *iNext; // jedno i obousměrný
struct TLinSeznam *iPrev; // obousměrný
}

TLinSeznam a = {0,NULL,NULL},...,d={0,NULL,NULL},  *Prvni=NULL;
Prvni = &a;    // pouze v této funkci a funkcích volaných
               // jinak proměnné zanikají
a.iNext = &b;
b.iNext = &c;
c.iNext = &d;
```

Ukázka realizace stromu v C:

```
struct TStrom {
double iData; // hodnota uzlu, obecně jakýkoli počet, jakýkoli typ
struct TStrom *iLeft; // jedna větev
struct TStrom *iRight; // druhá větev
}

TStrom a = {0,NULL,NULL},..., g={0,NULL,NULL},*Vrchol=NULL;
Vrchol = &a;    // pouze v této funkci a funkcích volaných
                // jinak proměnné zanikají
a.iLeft = &b;
a.iRight = &c;
b.iLeft = &d;
b.iRight = &e;
c.iLeft = &f;
c.iRight = &g;
```

**Varianty seznamů**

- jednosměrný x obousměrný (jednocestný x dvoucestný); obecný vícenásobně vázaný seznam
- cyklický/kruhový - buffer

- U jednosměrného potřebujeme často pracovat s prvkem „před“ – výhodou jsou rekurzní algoritmy. Například pro požadavek rušení od konce je výhodnější než neustálé hledání posledního prvku.
- obousměrný má snadnější pohyb v poli
- jednosměrný umožňuje spojení několika seznamů (seznamy začnou různě, ale v určité pozici se napojí na jediný koncový seznam. Problém s odstraněním spojovacího/společného prvku.
- obousměrný má o ukazatel více

srovnání seznamu, stromu a pole – časová složitost

- seznamy a stromy jsou fragmentované
- nalezení prvku z daným indexem – O(n) ; O(log n) O(1)
- vložení, vybrání prvku na začátku – O(1) ; O(log n) ; nelze
- vložení, vybrání prvku na konec - O(1)/O(n) (ukazatel Tail ano/ne) ; O(log n) ; nelze
- vložení, vybrání uprostřed – nalezení prvku + O(1); O(1) ; O(log n) ; nelze
- paměťové nároky: + n . ukazatelů ; +2 . n . ukazatelů ; +0

Základní operace se seznamy

- přidání prvku – nastavení na (předchozí) prvek + vložení
- vybrání prvku – nastavení na (předchozí) prvek + vybrání
- předání odkazu na prvek
- zjištění zda je seznam prázdný

Operace musí obsloužit základní stavy:

- práce s prázdným seznamem
- manipulace s prvním prvkem
- manipulace s posledním prvkem
- manipulace s prvky uprostřed seznamu

Řešení krajních stavů

- *dummy nodes* (*sentinel nodes*) prázdné prvky na začátku a na konci,
- jelikož jsou tyto prvky bez dat a tudíž se s nimi „nepracuje“ –> pro manipulaci existuje jediný stav: manipulace s prvkem uprostřed seznamu

Realizace manipulace se seznamem pomocí:

- Ukazatel na první prvek (vždy) – *head*,
- ukazatel *tail* (většinou poslední, někdy zobecnění = některý prvek (koncová sekvence),)
- ukazatel *actual* – pomocný ukazatel

Kde jsou uložena data/prvky (kdo je „vlastní“, dáno předpisem seznamu)

- předání dynamické proměnné ukazatelem a vložení do seznamu – majitelem zůstává původce
- předání dynamické proměnné ukazatelem a vložení do seznamu – majitelem se stane seznam
- předání dynamické proměnné ukazatelem a vložení kopie do seznamu – majitel seznam
- předání hodnoty proměnné a vložení kopie do seznamu – majitel seznam
-
- má-li hodnotu, jedná se o interní uložení, jinak externí
- „majitel“ je zodpovědný za zrušení prvku (zvláště u předání ukazatelem)

**Práce se seznamy**

*Head*, *Tail* a *Prvek* jsou odkazy na prvky, se kterými se pracuje

*Procházení seznamu*

- nejjednodušší činnost (tisk, rušení celého seznamu, prohledávání, ...)

```
Pokud je seznam prázdný udělej ... a ukonči
Aktuální_prvek = Head
Dokud je Aktuální_prvek platný
   zpracuj data/proveď činnost
   Aktuální_prvek = Next(Aktuální_prvek)
```

*Přidání prvku na začátek*

```
Vstup – seznam, vkládaný Prvek
Pokud je seznam prázdný
   Head = Prvek
   SetNext(Prvek, NULL)// pro jistotu. Mělo by být inicializováno.
jinak
   SetNext(Prvek,Head)   // bývalý první prvek bude druhý
   Head = Prvek
```

*Odebrání prvku ze začátku*

```
Vstup – seznam
Pokud je seznam prázdný
   oznam chybu (hodně implementací neřeší, nutno odebírat po
                                    kontrole prázdnosti Empty)
jinak
   pom = Head
   Head = Next(Head)
   SetNext(pom, NULL) // kvuli bezpeci pri vrácení; některé
                      // algoritmy vyžadují pro zrušení
   zruš pom/ vrať pom – podle konkrétní definice (existují obě
                                                        verze)
```

Bude fungovat i pro poslední prvek?

*Přidání prvku na danou pozici*

```
vstup: seznam, prvek, index
pokud je seznam prázdný nebo je index mimo počet prvků seznamu
   ...
jinak
   Pom = Head
   Dokud Pom neni prvek za který chceme vkládat
      Pom = Next(Pom)
   SetNext(Prvek,Next(Pom))
   SetNext(Pom,Prvek)
```

*Odebrání prvku z dané pozice pozici*

```
vstup: seznam, index
pokud je seznam prázdný nebo je index mimo počet prvků seznamu
   ...
jinak
   Pom = Head
   Dokud Pom neni prvek za kterým je rušený prvek
      Pom = Next(Pom)
   Ruseny = Next(Pom)
   SetNext(Pom,Next(Ruseny))
   SetNext(pom, NULL) // kvuli bezpeci pri vrácení;
                      // některé algoritmy vyžadují pro zrušení
   zruš pom/vrať pom – podle konkrétní definice (mohou být obě
                                                        verze)
```

Poslední prvek v seznamu?

## Obousměrný seznam

*Procházení*

```
Pokud je seznam prázdný udělej ... a ukonči
Aktuální_prvek = Head
Dokud je Aktuální_prvek platný
   zpracuj data/proveď činnost
   Aktuální_prvek = Next(Aktuální_prvek)

// procházení opačným směrem
Aktuální_prvek = Tail
Dokud je Aktuální_prvek platný
   zpracuj data/proveď činnost
   Aktuální_prvek = Prev(Aktuální_prvek)
```

*Přidání prvku na začátek*

```
Vstup – seznam, vkládaný Prvek

Pokud je seznam prázdný
   Head = Prvek
   Tail = Prvek
   SetNext(Prvek, NULL)  - mělo by být, ale pro jistotu
   SetPrev(Prvek, NULL)  - mělo by být, ale pro jistotu
jinak
   pom = Head // pro propojeni
   SetNext(Prvek,Head)   // bývalý první prvek bude druhý
   Head = Prvek
   SetPrev(pom, Prvek)
```

*Odebrání prvku ze začátku*

```
Vstup – seznam
Pokud je seznam prázdný
   oznam chybu (hodně implementací neřeší, nutno odebírat po
                                          kontrole prázdnosti Empty)
jinak
   pom = Head
   Head = Next(Head)
   SetPrev(Head, NULL);
   SetNext(pom, NULL) // kvuli bezpeci pri vrácení;
                      některé algoritmy vyžadují pro zrušení
   SetPrev(           )
   zruš pom/ vrať pom – podle konkrétní definice (existují obě
                                                        verze)
```

*Přidání prvku na danou pozici*

- varianta *PřidejZa* – nastavíme na daný prvek a potom pouze přidáme, parametrem může být index nebo přímo prvek/hodnota, který se má najít
- Rozdělení je možné i u ostatních algoritmů
- u obousměrného seznamu je možné i *PřidejPřed*

```
vstup: seznam, prvek, index
pokud je seznam prázdný nebo je index mimo počet prvků seznamu
   ...
jinak
   Pom = Head
   Dokud Pom neni prvek za který chceme vkládat
      Pom = Next(Pom)
   SetNext(Prvek,Next(Pom))
   SetNext(Pom,Prvek)
   SetPrev(          )
   SetPrev(          )
```

*Odebrání prvku z dané pozice pozici*

```
vstup: seznam, index
pokud je seznam prázdný nebo je index mimo počet prvků seznamu
   ...
jinak
   Pom = Head
   Dokud Pom neni prvek za kterým je rušený prvek
      Pom = Next(Pom)
   Ruseny = Next(Pom)
   SetNext(Pom,Next(Ruseny))
   SetPrev(          )
   SetPrev(          )
   SetNext(pom, NULL) // kvůli bezpečí při vracení; některé
algoritmy vyžadují pro zrušení
   SetPrev(          )
   zruš pom/ vrať pom – podle konkrétní definice (mohou být obě
                                                        verze)
```

## Iterátory

- návrhový vzor pro procházení seznamu (též Cursor)
- oddělení implementace od přímého použití (uživatel používá seznam prostřednictvím funkcí, které pracují s iterátorem)
- slouží k procházení kontejneru (např. procházení stromu není pro běžného uživatele jednoduché – jak určit pořadí?)
- vytváří jednotné rozhraní pro různé implementace (seznam i strom mohou být díky společnému interface používány stejným způsobem)
- díky pomocným proměnným může zajistit rychlejší práci (např. bude-li si pamatovat svou polohu v lineárním seznamu, nemusí při vyhledání pozice prvku vždy začínat od počátku)
-
- spojeny s příkazem foreach, kdy postupně prochází seznam a vrací jeho prvky (begin,end)
- realizuje například: nastavení na začátek, nastavení na/za konec; zjištění zda je seznam prázdný; zjištění zda existuje další prvek; posun na další prvek; možnost přístupu k prvku
- při manipulaci s kontejnerem (zvláště vypouštění prvků) je nutné o tom iterátor informovat (může mít pomocné informace, které je nutné aktualizovat – první a poslední prvek, počet prvků, aktuální prvek ...)

**Poznámky**:

- cyklická fronta/buffer/vyrovnávací buffer
- Realizace seznamu pomocí pole
  – pole struktur, struktura má ale místo ukazatelů indexy, na kterých jsou „sousední“ prvky
  - nevýhodou je situace, kdy pole nenaplníme, nebo kdy potřebujeme přidat prvky (alokace + kopie (move) + uvolnění paměti. (Pro přidání jednoho prvku potřebujeme paměť na celý seznam – při větších seznamech nemusí být paměť dostupná, zatímco jeden samostatný prvek je možné alokovat bez problémů)
- Binární strom pomocí pole
  – indexy prvků jdou spočítat
  – následující uzel je na indexech 2n+1 a 2n+2
  - využití heapsort

| | indexy v poli | | | | |
|---|---|---|---|---|---|
| úroveň 0: | | 0 | | | |
| úroveň 1: | 1 | | 2 | | následníci 0: 2x0+1 a 2x0+2 |
| úroveň 2: | 3 | 4 | 5 | 6 | následníci 1: 2x1+1 a 2x1+2 |
| | | | | | následníci 2: 2x2+1 a 2x3+2 |

Na jakých indexech začínají řádky?