Dynamický datový typ spojový seznam

1 Prvek

V tomto cvičení se budeme věnovat implementaci ADT *spojový seznam*. Tento typ lze považovat za nadtyp fronty i zásobníku, protože seznam obsahuje funkcionalitu obou těchto typů.

Prvek seznamu je stejný, jako u předchozích typů, ale může být i obecnější. Protože některé operace nad seznamem jsou snazší, když se v něm můžeme pohybovat oběma směry, budeme pracovat s *obousměrně zřetězeným spojovým seznamem*.

V úloze pro toto cvičení budeme pracovat s posloupností číselných hodnot, proto bude hodnota prvku typu `float. Navíc` ale přibude ukazatel na předchozí prvek.

```
typedef struct _prvek Tprvek;

struct _prvek {
  float hodnota;
  Tprvek * dalsi;
  Tprvek * pred; //chozi
};
```

2 Seznam

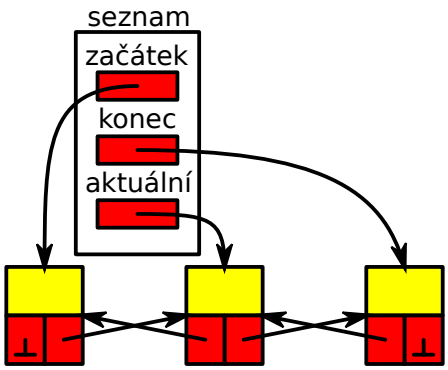
ADT seznam bude poskytovat tutéž funkcionalitu jako předchozí typy a přibudou ještě další operace. Zastřešující struktura bude uchovávat ukazatele na první a na poslední prvek a navíc budeme pracovat s ukazatelem na *aktuální prvek*. Takový ukazatel pomůže urychlit některé operace, protože nebude potřeba seznamem tak často listovat od začátku.

```
typedef struct {
  Tprvek* zacatek;
  Tprvek* konec;
  Tprvek* aktualni;
  int delka; int indexAktualniho;
} Tseznam;
```

V obecném seznamu lze prvky přidávat a odebírat jak na začátku, tak na konci seznamu. Nově to půjde dělat také uprostřed seznamu.

Práci s okrajovými prvky už máme hotovou z minulých cvičení. Operace *push*, *pop*, *enqueue* a *dequeue* prostě použijeme i v seznamu.[1] Nově zavedené operace *insert* a *remove* nyní budou obecnějšími variantami těchto operací a budou tyto starší varianty používat pro řešení okrajových situací.[2]

Graficky znázorněná představa obousměrného seznamu může vypadat takto:



Podobně jako u předchozích typů můžeme označit jednu vlastnost seznamu, která ospravedlňuje jeho používání. Je to *flexibilita*. Seznam se používá všude tam, kde je potřeba často hýbat s prvky uprostřed.

2.1 Modul seznam.h

Seznam budeme realizovat jako modul. Modul bude mimo jiné sloužit k oddělení algoritmů, které *tvoří a realizují seznam* na jedné straně a algoritmů, které *hotový seznam používají*. U těch druhých se typicky nechcete vrtat ve vnitřnostech abstraktního typu, ale používáte už hotové operace. Pokud musíte v takových

1 Bude je potřeba drobně upravit. Je nutné aktualizovat ukazatel na aktuální prvek. Dále je nutné pracovat s ukazatelem na předchozí prvek. Dejte si také pozor na přejmenované položky v zastřešující struktuře – vrchol, prvni, posledni → zacatek, konec.

2 Takto si chytře ušetříme více než polovinu práce. Prostě znovu použijeme algoritmy, které už máme hotové – přece je nebudeme vymýšlet znovu. Dobrý programátor je v podstatě líný, takže dělá věci tak, aby je mohl v budoucnu znovu použít. Příliš „poctivý“, nebo spíš neprozřetelný programátor se zbytečně nadře.

algoritmech pracovat s vnitřnostmi seznamu, asi ten seznam není implementovaný správně.

2.2 Inicializace seznamu

Zřejmé. Tento problém už jsme řešili u zásobníku i fronty. Důležité je na to nezapomenout.

```
// varianta pro dynamickou proměnnou
Tseznam* seznamInitD();
void seznamFree(Tseznam *s);
```

2.3 Operace *insert*

Tato operace vkládá novou hodnotu za aktuální prvek seznamu. Nejprve alokuje nový prvek a inicializuje jej zadanou hodnotou. Pak provede navázání nového prvku za aktuální prvek seznamu. Nakonec nastaví nový prvek jako nový aktuální prvek seznamu.

Při vkládání na konec seznamu operace insert použije operaci enqueue. V této verzi operace vkládá za aktuální prvek, proto neumí vložit hodnotu na první pozici.

Protože operace může selhat, když se nepovede alokace, bude vracet logickou hodnotu.

```
bool seznamInsert(Tseznam* s,
                  float x);
```

Jednotlivé kroky této operace je opět potřeba dělat pečlivě ve správném pořadí, jinak hrozí havárie programu.

Varianty operace insert

Naši variantu operace vložení bychom mohli nazvat také insertBehind, tedy vlož za aktuální prvek. K ní by šlo dopsat ještě podobnou operaci insertBefore, tedy vlož před aktuální prvek. Obě tyto varianty mají problém s vkládáním na jednom konci seznamu.

Další variantou je funkce insertAt, která vloží prvek na pozici danou indexem. Tato funkce si aktuální prvek nejprve posune na pozici danou indexem a pak teprve vkládá nový prvek. Když index sahá za konec seznamu, vkládá na konec.

V praxi se používají všechny tyto varianty.

2.4 Operace *remove*

Tato operace odebírá aktuální prvek seznamu. V obousměrném seznamu je její implementace jednodušší, než u jednosměrného seznamu, protože máme přímý přístup k předchozímu prvku, který je potřeba spojit s následníkem aktuálně rušeného prvku.

Rušení okrajových prvků lze teď snadno řešit pomocí operací pop a dequeue.

Operace se opět nemusí povést, proto bude vracet logickou funkční hodnotu. V tomto případě se operace nepovede, když se pokusíme odstranit prvek z prázdného seznamu.

```
bool seznamRemove(Tseznam* s,
                  float* x);
```

Otázkou zůstává, kam má ukazovat ukazatel na aktuální prvek po jeho zrušení. Ještě před rušením prvku se posune na následníka, aby nový aktuální měl stejné pořadí, jako rušený. Jedinou výjimku tvoří rušení posledního prvku, kdy se tento ukazatel přesouvá na předchozí prvek.

Snad to není potřeba, ale stejně raději zdůrazním, že operace free se bude volat nad pomocným ukazatelem, ne nad ukazatelem aktuálním.

2.5 Operace prev a next

Tyto operace mají za úkol posouvat ukazatel na aktuální prvek doleva nebo doprava. Budou vracet logickou hodnotu a při pokusu o posun za okraj seznamu budou vracet **`false`**.

2.6 Operace currentData

Tato operace má za úkol vrátit data aktuálního prvku. Operace nebude fungovat nad prázdným seznamem – pak může vracet logickou hodnotu **`false`**.

3 Úloha – řazený seznam (bez dvojčat)

Vytvoř implementaci spojového seznamu jako modul. Pak pomocí něj vytvoř program, který bude načítat soubor s číselnými hodnotami a bude je *zařazovat* do seznamu podle velikosti. Po přečtení souboru bude seznam obsahovat seřazenou posloupnost hodnot. Tuto výslednou posloupnost vypiš do výstupního souboru. Rozumí se samo sebou, že cesty k souborům bude zadávat uživatel.

Rozšíření:

- Každé dva stejné prvky se vyruší – tj. když už je v seznamu prvek s hodnotou

pět a vkládám další pětku, dojde k odebrání pětky ze seznamu. Třetí pětka se zase vloží, čtvrtá ji vyruší a tak dále. Když bude ve vstupu sudý počet prvků stejné hodnoty, ve výsledném seznamu takový prvek nebude.

Pár rad:

- Načrtni si nejprve algoritmus zařazování do seznamu. Používej v něm pouze operace nad seznamem, ne přímý přístup do něj. V tom algoritmu nepotřebuješ vidět dovnitř seznamu, nebudeš používat žádné vnitřní složky datových struktur. Ideově půjde o podobný algoritmus jako insertionSort.
- Teprve potom zvaž, jaká varianta operace insert je pro tebe nejvýhodnější.
- Při zařazování není potřeba listovat prvky od začátku. Listování od aktuálního prvku bude v průměru rychlejší – jen někdy bude potřeba listovat doleva, jindy doprava.
- Studenti, kteří stříhají ušima při myšlence na kruhový seznam, mohou realizovat tuto variantu. Pokud si ji dovedete představit, může být výsledná implementace jednodušší.