Dynamický datový typ fronta

1 Prvek

V tomto cvičení se budeme věnovat implementaci ADT *fronta*. Prvky fronty budou, stejně jako u zásobníku, tvořit lineární spojový seznam. Struktura pro takový prvek tudíž bude vypadat naprosto stejně.

V úloze pro toto cvičení budeme pracovat s posloupností číselných hodnot, proto bude hodnota prvku typu `float`.[1]

```
typedef struct _prvek Tprvek;

struct _prvek {
  float hodnota;
  Tprvek * dalsi;
};
```

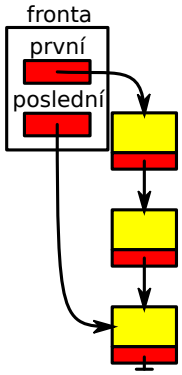
2 Fronta

ADT fronta opět vytvoříme jako strukturu. Tentokrát bude potřeba uchovávat dva ukazatele, na první a na poslední prvek fronty. Také se bude hodit uchovávat údaj o aktuální délce, případně o maximální povolené délce, pokud půjde o frontu s omezenou délkou.

```
typedef struct {
  Tprvek* prvni;
  Tprvek* posledni;
  int delka;
  int maxDelka;
} Tfronta;
```

Prvky se vkládají vždy na konec fronty, tedy za poslední prvek. Odebírání prvků se děje ze začátku fronty. Funguje to analogicky tomu, jak předpokládáte fungování fronty v obchodě před pokladnou.

Graficky znázorněná představa fronty může vypadat takto:



Charakteristická vlastnost fronty se označuje jako FIFO (first in, first out). Fronta *zachovává pořadí* prvků. V algoritmech se fronta používá často jako vyrovnávací paměť – buffer, v situacích kdy prvky přicházejí rychleji, než je stíháme zpracovat.

2.1 Modifikace fronty

Častou modifikací výchozí implementace fronty je *fronta s omezenou maximální délkou*. V tomto případě hlídáme, zda je fronta již zaplněna. Pokud ano, buďto skončí operace vkládání (enqueue) chybou, nebo dojde před vložením k zahození prvního (nejstaršího) prvku fronty.

Při implementaci v modulu bude lepší první varianta, protože druhou si pak může uživatel modulu snadno realizovat sám, pokud ji ve svém algoritmu potřebuje.

Další modifikací je takzvaná *prioritní fronta*. U tohoto řešení máme nějaký mechanismus, jak klasifikovat příchozí prvky – přidělujeme jim prioritu. Prvky s vyšší prioritou pak mohou předbíhat ve frontě prvky s nižší prioritou.

Pokud máme jen několik tříd prvků, to znamená, že více prvků má vždy stejnou prioritu, realizuje taková fronta skutečný *třídící algoritmus*. Máme-li širokou škálu priorit, kdy v podstatě každý prvek může mít jinou prioritu, bude taková fronta realizovat *řadící algoritmus*, který už znáte – insertion sort.

1 Přesně v tuto chvíli by vás mělo začít štvát, že musíme pro každou úlohu ten datový typ předělávat. Nešlo by použít nějaké obecné řešení, které by bylo méně pracné? Jazyk C nabízí pouze řešení pomocí netypového ukazatele (void*), což tedy moc pohodlné není. V objektově orientovaných jazycích jako Java, C++ aj. se pro tento účel používají tzv. *šablonové* nebo *generické typy*. Princip je takový, že se např. fronta implementuje s datovou položkou jakéhosi zástupného, nekonkrétního typu a konkrétní typ se dosadí až při definici konkrétní proměnné typu fronta. Knihovnu s frontou pak stačí implementovat pouze jednou a uživatel si do ní dosadí konkrétní typ dat, jaký zrovna potřebuje.

2.2 Inicializace fronty

Před použitím je potřeba frontu správně inicializovat. Je důležité nastavit vnitřní ukazatele na hodnotu NULL, což představuje prázdnou frontu.

Podobně jako jsme si to ukázali u zásobníku, i frontu lze provozovat jako proměnnou typu struktura `Tfronta`, nebo jako proměnnou typu ukazatel na strukturu `Tfronta*`. Nezapomeňte, že u druhé varianty se nesmí pracovat s neinicializovanou proměnnou!

Inicializační funkce pak mohou vracet buďto samotnou strukturu, nebo ukazatel na alokovanou proměnnou. Při realizaci dynamické verze je potřeba uvolnění zastřešující struktury zahrnout do funkce `frontaFree`.

```
// varianta pro statickou proměnnou
Tfronta frontaInit(int maxDelka);

// varianta pro dynamickou proměnnou
Tfronta* frontaInitD(int maxDelka);
void frontaFree(Tfronta *f);
```

Parametr *maxDelka* slouží pro nastavení fronty s omezenou délkou. Není nijak složité zařídit, aby fronta fungovala jak v obecném režimu s neomezenou délkou, tak i v režimu s omezenou délkou. Stačí stanovit, že když hodnota *maxDelka* bude nulová (nebo třeba záporná) bude fronta neomezená, a naopak, když bude kladná, nepůjde do fronty vložit více prvků.

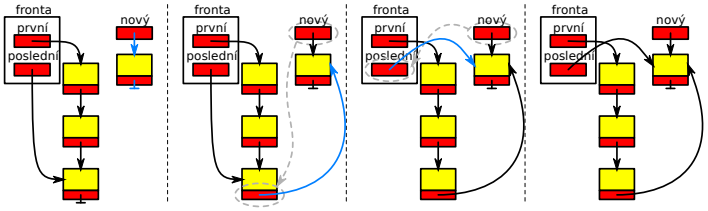
2.3 Operace *enqueue*

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

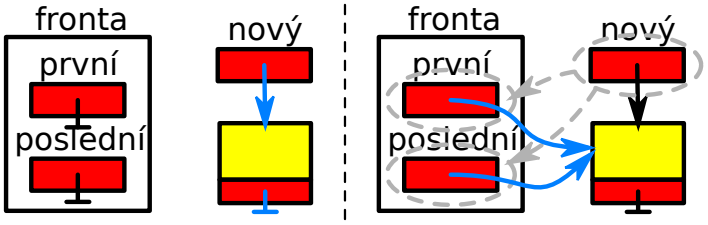
Protože operace může selhat, když se nepovede alokace, případně když bude fronta plná, bude vracet logickou hodnotu. Hodnotu složky *maxDelka* je potřeba testovat na začátku operace.

```
bool frontaEnqueue(Tfronta *f,
                   float x);
```

Jednotlivé kroky této operace je opět potřeba dělat pečlivě ve správném pořadí, jinak hrozí havárie programu. Nebojte se při programování používat papír a tužku a jednotlivé kroky si kreslete.



Při vkládání je třeba speciálně pamatovat na situaci, kdy se prvek vkládá do prázdné fronty. V takovém případě se bude měnit nejenom ukazatel na poslední prvek, ale i ukazatel na první prvek fronty. Nový prvek totiž bude tvořit jednoprvkovou frontu a bude tedy zároveň jejím prvním i posledním prvkem.



2.4 Operace *dequeue*

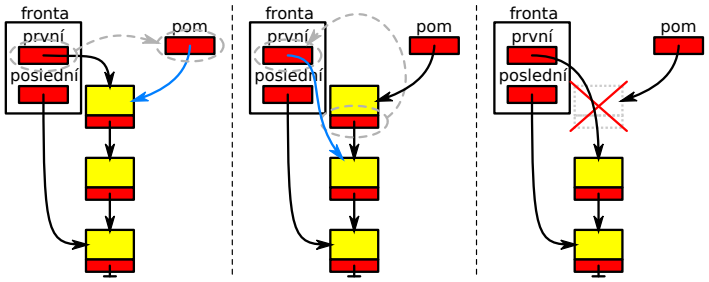
Tato operace odebírá prvek ze začátku fronty. Postup je podobný, jako u operace *pop* v zásobníku. Operace vrací hodnotu, kterou nesl odebíraný prvek.

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

```
bool frontaDequeue(Tfronta* f,
                   float* x);
```

Všimněte si, že hodnota nesená odebíraným prvkem se teď vrací přes parametr předávaný odkazem.

V této operaci je potřeba nastavit pomocnou proměnnou typu ukazatel na původní první prvek fronty. Pak se provede přemostění, kdy složka *prvni* začne ukazovat na následující prvek. Pak je teprve možné uvolnit prvek odkazovaný pomocnou proměnnou. Samozřejmě nesmíme zapomenout zkopírovat hodnotu nesenou tímto prvkem ještě před jeho uvolněním.



Analogicky předchozí operaci je potřeba pamatovat na situaci, kdy se ve frontě nachází pouze jediný prvek. Po jeho odebrání bude ukazatel na poslední prvek nastaven na hodnotu NULL. Když to nastane, je třeba nastavit také ukazatel na první prvek na NULL. Tím vznikne prázdná fronta.

Bez tohoto ošetření by složka *prvni* obsahovala neplatný ukazatel a při jakékoli další operaci s frontou by došlo k havárii.

3 Úloha – poslední prvky posloupnosti

Existuje spousta algoritmů, kde najde zásobník své využití. Stejně jako zásobník, se i fronta používá především pro svou charakteristickou vlastnost (FIFO) spíše, než pro svou schopnost ukládat libovolné množství prvků.

V následující úloze použij frontu s omezenou délkou.

Napiš program, jehož vstupem je celočíselná hodnota N a posloupnost prvků předem neznámé délky.

- Prvky posloupnosti jsou desetinná čísla.
  - V zásadě by to ale mohly být i jakékoli jiné hodnoty, například řádky textu.
- Posloupnost čti z uživatelsky zadaného souboru nebo ze standardního vstupu.
  - Při ručním zadávání ze standardního vstupu se konec vstupu v textovém terminálu Windows zadává pomocí Ctrl+Z. V Linuxovém terminálu se používá zkratka Ctrl+D.
  - Při volání funkce `fscanf` můžeš jako argument použít zabudovanou proměnnou `stdin`. Pak budeš číst ze standardního vstupu místo ze souboru.
- Program nemůže číst vstup vícekrát.
  - Nemůžeš si například dopředu spočítat prvky. Když se čte stream, tak to prostě nejde.
- Tvým úkolem je zařídit, aby program vypsal posledních N prvků vstupní posloupnosti a zachoval při tom jejich pořadí.
  - Použij svou implementaci abstraktního datového typu fronta.