Konstruktory a destruktory

Nedostatek

- atributy po vytvoření objektu nejsou automaticky inicializovány
  - hodnota atributů je náhodná
- vytvoření metody pro inicializaci, kterou musí programátor explicitně zavolat, není spolehlivé
  - *není zaručeno, že ji programátor, či spíše další uživatel knihovny, nezapomene zavolat*

Bezpečné pole z minulé hodiny

```
class BezpPole
{
 private:
  int n,akt,*pole;
 public:
  void vloz(int prvek);
  int vrat_prvek(int index);
  int vrat_akt_pocet();
  void init();
};
```

Bezpečné pole z minulé hodiny

```
void main()
{
  int i;
  BezpPole a;
  a.init(); // na init nesmíme zapomenout
  a.vloz(3); a.vloz(4); a.vloz(2);
  for(i=0;i<a.vrat_akt_pocet();i++)
    cout << a.vrat_prvek(i) << endl;
}
```

Konstruktor

- OOP poskytuje prostředek pro počáteční inicializaci (nejen) atributů tzv. *konstruktor*

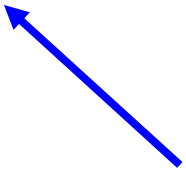
Konstruktor

- jde o speciální metodu, která je automaticky vyvolána při vzniku objektu
  - konstruktor musí přirozeně programátor naprogramovat
- slouží pro počáteční inicializaci objektu
- pokud není konstruktor definován, překladač jej vytvoří automaticky (implicitní), který ale hodnoty atributů neinicializuje (prázdný)

- konstruktor má stejný identifikátor jako je jméno (identifikátor) třídy
  - t.j. musíme deklarovat a vytvořit metodu se stejným názvem jako jméno třídy a ta je automaticky konstruktorem
- konstruktor nevrací žádná data, není deklarován ani jako **`void`**!
- parametry může mít libovolné, lze jej přetížit
  - je to dokonce i vhodné, abychom mohli inicializovat objekt různými způsoby
  - je vhodné mít vždy implicitní konstruktor (bez parametrů)

```
class Komplex
{
 public:
    float re,im; //reálná a imag. část
    float velikost();
    Komplex();
    Komplex(float real, float imag);
};
```

implicitní konstruktor



přetížený konstruktor

- **implementace**

```
Komplex::Komplex()
{
  re = im = 0;
}

Komplex::Komplex(float real, float imag)
{
  re = real; im = imag;
}
```

- použití

```
void main()
{
  Komplex k_cislo;
  Komplex c1(1,3);
  Komplex *pc1,*pc2;

  pc1 = new Komplex(5,10);
  pc2 = new Komplex();
  float vel = k_cislo.velikost();
  vel = pc1 -> velikost();
  delete pc1; delete pc2;
}
```

zde se volá implicitní konstruktor

zde se volá přetížený konstruktor

zde se volá implicitní konstruktor

- pokud deklarujeme konstruktor s parametry, musíme deklarovat a implementovat i implicitní, chceme-li jej využívat
  - pak již implicitní konstruktor není generován automaticky
- při vzniku objektu se nejprve alokuje paměť pro objekt (pro atributy) a pak se volá konstruktor
- v konstruktoru zpravidla inicializujeme atributy nebo alokujeme potřebnou dynamickou paměť, jestliže ji objekt využívá
  - ruší se pak v *destruktoru*

- konstruktor nelze dodatečně vyvolat

```
void main(void)
{
  Komplex c;

  c.Komplex(2,2);
}
```

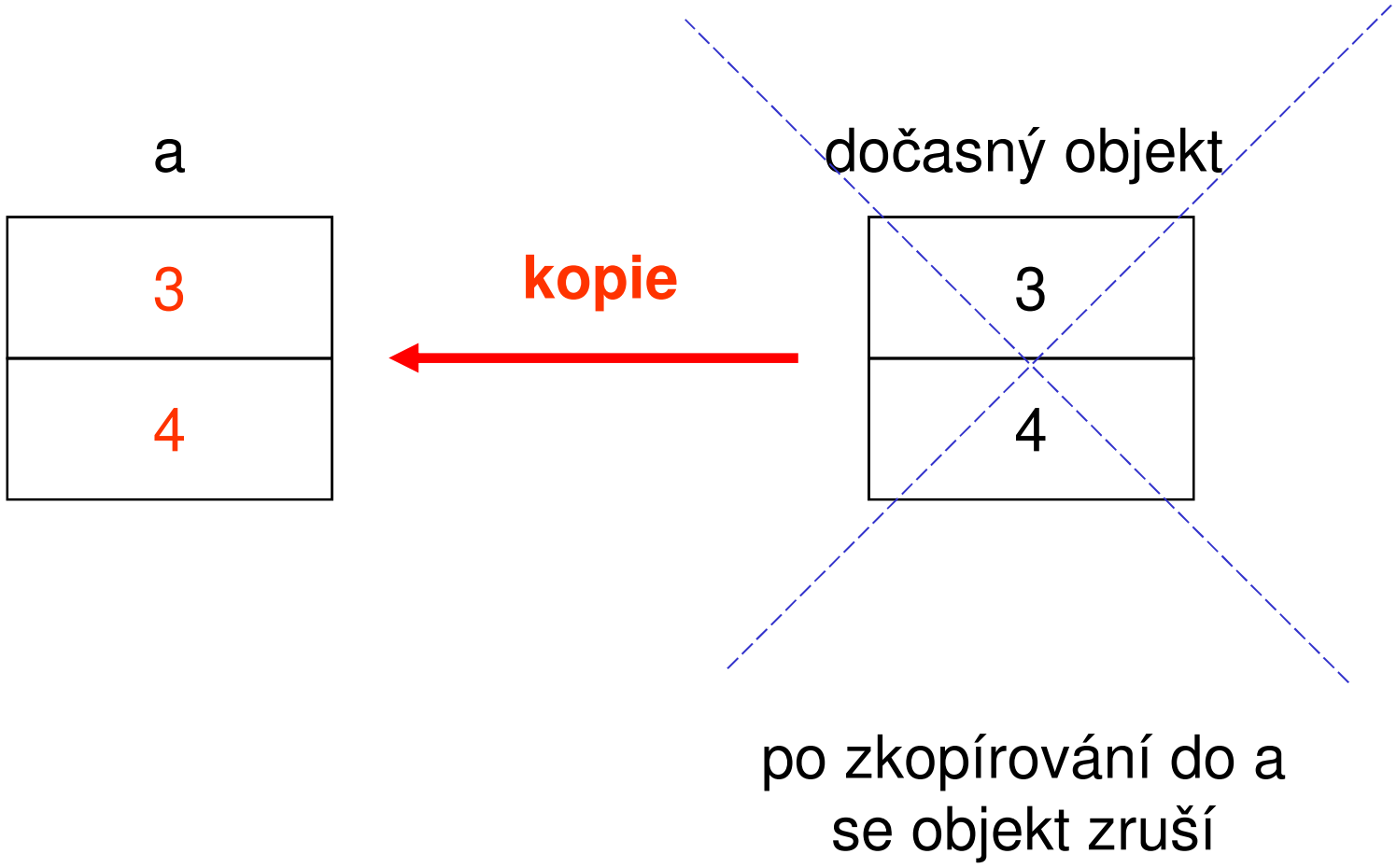
chyba

- explicitní volání konstruktoru znamená vytvoření dočasného (nepojmenovaného) objektu, který se po použití automaticky zruší

```
Komplex a;
// zavolal se impl. konstruktor

a = Komplex(3,4);
```

  – do proměnné `a` se zkopíruje hodnota dočasného objektu, ve kterém je re=3 a im=4


a
3
4
kopie
dočasný objekt
3
4
po zkopírování do a
se objekt zruší

Poznámka:

- inicializaci atributů lze v konstruktoru provést pomocí *inicializátorů* (při deklaraci třídy jako in-line konstruktor nebo při implementaci v souboru .cpp)
  - u atributů typu reference, konstantních atributů a objektových atributů, kde není ve třídě definován implicitní konstruktor, je to jediná možnost

```
class Komplex {
  float re, im;
public:
  Komplex(): re(0),im(0) {};
  Komplex(float x, float y): re(x),im(y) {};
}
```

- konstruktor s inicializátory v souboru .cpp:

```
Komplex::Komplex(): re(0),im(0)
{

}
Komplex::Komplex(float x, float y):
  re(x),im(y)
{

}
```

```
class A
{
    int x;
    bool barevne;
public:
  // máme jen konstruktor s parametry
  A(bool b);
};

A::A(bool b)
{
  x = 0; barevne = b;
}
```

```
class B
{
    int z;
    A a;
public:
  B();
  B(bool b);
};

B::B(): a(true)
{
  z = 0;
}
B::B(bool b): a(b),z(0)
{

}
```

Úkol

Doplňte třídu pro uložení bodů z minulého cvičení o konstruktor. Definujte dva přetížené konstruktory

- implicitního konstruktor, který inicializuje souřadnice bodu na počátek
- konstruktor se souřadnicemi px,py

Destruktor

- speciální metoda, která je automaticky volána při rušení objektu
- nejprve je zavolán destruktor nad objektem a pak je objekt zrušen (dealokována paměť)
- typicky:
  - pokud je v konstruktoru dynamicky alokována paměť, pak je v destruktoru provedena dealokace
  - změna statických (třídních) atributů

- destruktor má stejný identifikátor jako je jméno třídy a od konstruktoru je odlišen úvodním znakem „~” (vlnovka)
  - t.j. musíme deklarovat a vytvořit metodu se stejným názvem jako jméno třídy uvozenou vlnovkou a ta je automaticky destruktorem
- destruktor nesmí mít žádné parametry a nevrací žádnou hodnotu
- destruktor lze vyvolat přímo

Příklad - fronta

- datová struktura typu **FIFO**
  - First In - First Out
- prvky se odebírají v pořadí, jak byly vkládány

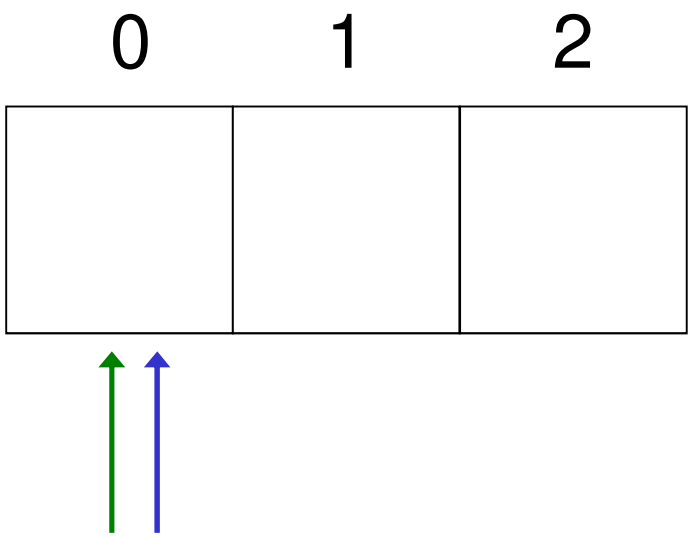
Možné implementace

- jako spojový seznam
  - „neomezená“ velikost fronty (omezená pouze velikostí dostupné paměti)
  - operace vložení prvku znamená vložení prvku na konec seznamu
  - operace výběr prvku je výběr z čela fronty
- polem pevné délky
  - fronta s omezenou velikostí
  - implementuje se jako tzv. **kruhová fronta**

Implementace pomocí pole pevné délky

- fronta je reprezentována polem a indexem čela a konce fronty
- fronta má pevnou délku a je implementována jako **kruhová**
  - indexy se zvyšují modulo délka pole
- je nutné implementovat ještě dotaz, zda je fronta plná a prázdná

Fronta délky $n = 3$



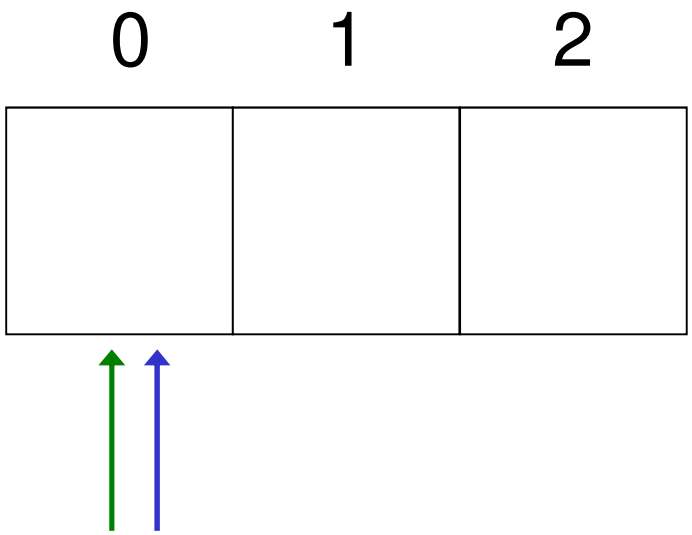
celo: 0

konec: 0

prázdná

plná

Fronta délky $n = 3$



vloz(3)

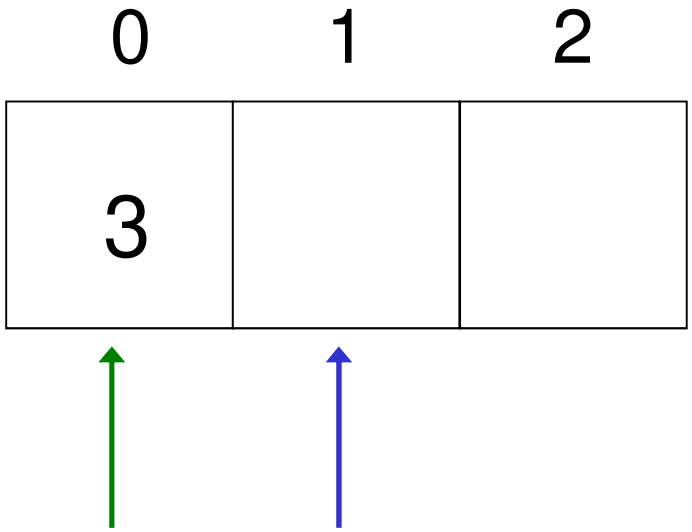
celo: 0

konec: 0

prázdná

plná

Fronta délky $n = 3$



celo: 0

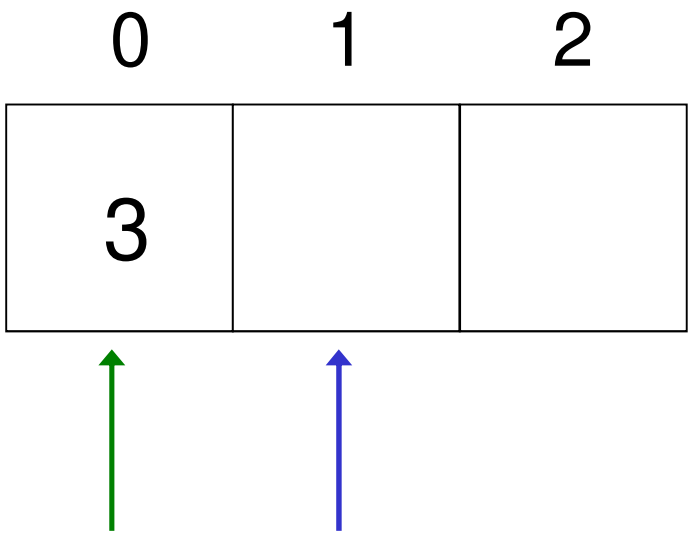
konec: 1

prázdná

plná

vloz(3)

Fronta délky $n = 3$



celo: 0

konec: 1

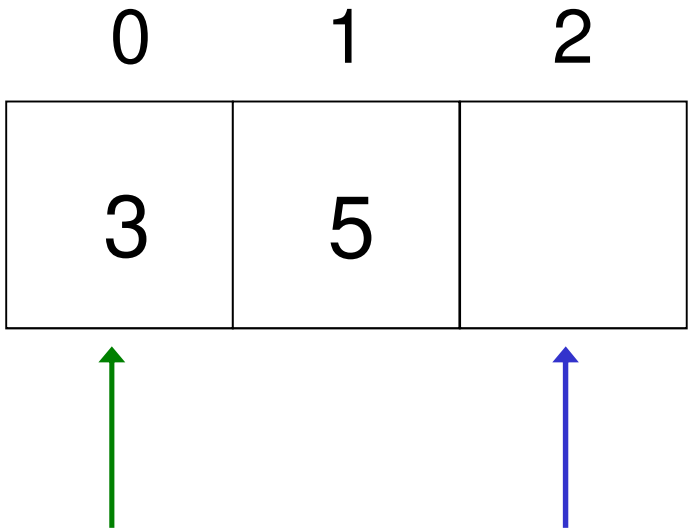
prázdná

plná

vloz(3)

vloz(5)

Fronta délky $n = 3$



celo: 0

konec: 2

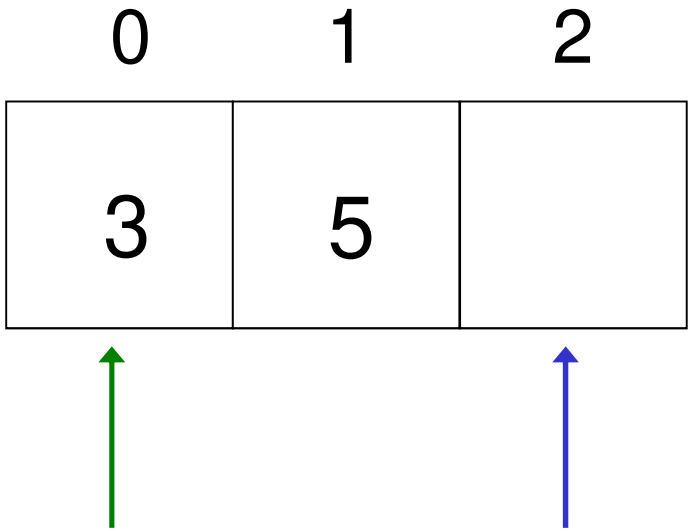
prázdná

plná

vloz(3)

vloz(5)

Fronta délky $n = 3$



celo: 0

konec: 2

prázdná

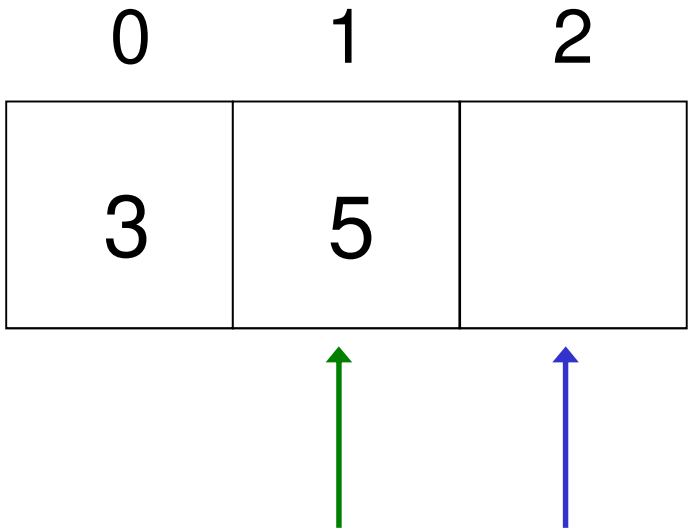
plná

vloz(3)

vloz(5)

vyjmi() – vrátí hodnotu z čela fronty - 3

Fronta délky $n = 3$



celo: 1

konec: 2

prázdná

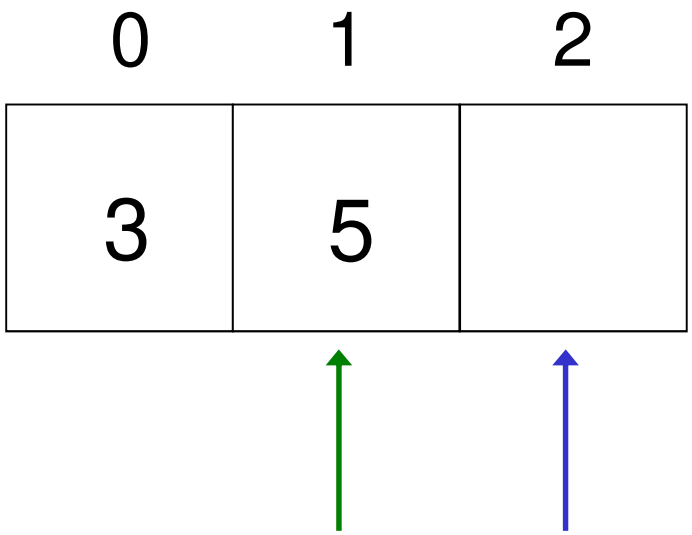
plná

vloz(3)

vloz(5)

vyjmi() – vrátí hodnotu z čela fronty - 3

Fronta délky $n = 3$



celo: 1

konec: 2

prázdná

plná

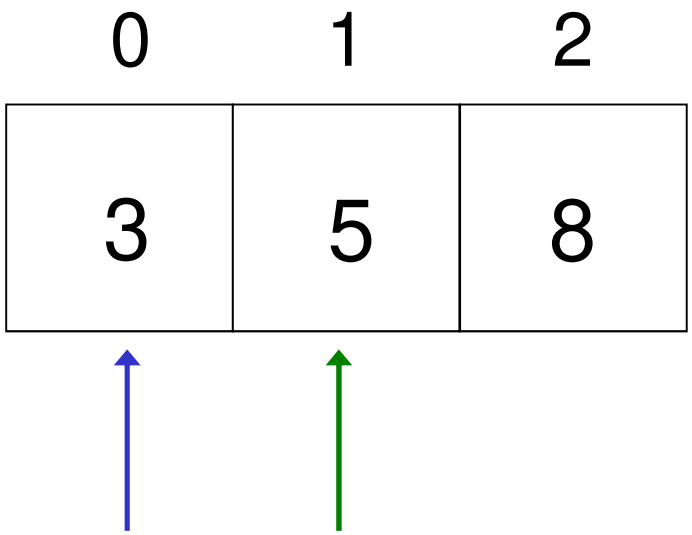
vloz(3)

vloz(5)

vyjmi() – vrátí hodnotu z čela fronty - 3

vloz(8)

Fronta délky $n = 3$



celo: 1

konec: 0

prázdná

plná

vloz(3)

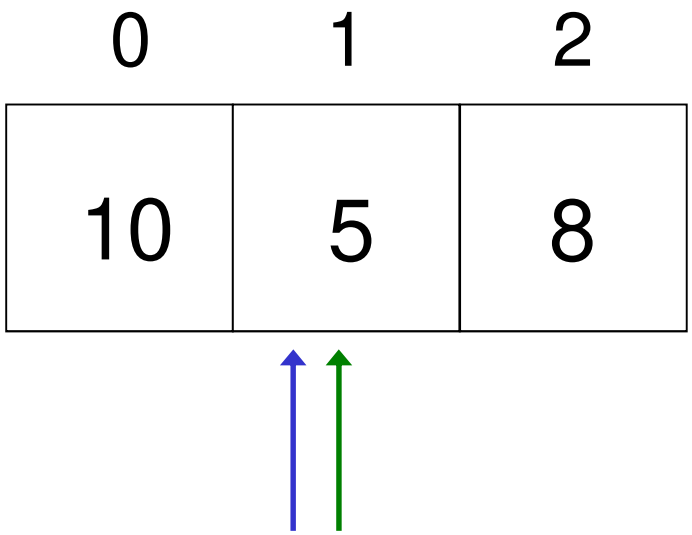
vloz(5)

vyjmi() – vrátí hodnotu z čela fronty - 3

vloz(8)

vloz(10)

Fronta délky $n = 3$



celo: 1

konec: 1

prázdná

plná

vloz(3)

vloz(5)

vyjmi() – vrátí hodnotu z čela fronty - 3

vloz(8)

vloz(10)

- index *celo* ukazuje na prvek pole na čele fronty, který bude odebrán
- index *konec* ukazuje na prvek v poli, kam se zapíše nový prvek
- pokud se indexy *celo* a *konec* rovnají, je fronta buď plná nebo prázdná
  - podle rovnosti indexů není možné rozlišit stav fronty
  - musím tedy tyto stavy uchovávat zvlášť
  - *pravidlo*: pokud se po přidání prvku indexy rovnají, je fronta plná (obdobně: prázdná)

Příklad - kruhová fronta pevné délky

```
class TFronta {
  int delka,celo,konec;
  bool plna,prazdna;
  int *fronta;
 public:
  bool je_prazdna();
  bool je_plna();
  bool vloz(int prvek);
  int vyber();
  TFronta();
  TFronta(int velikost);
  ~TFronta();
};
```

destruktor

```
TFronta::TFronta()
{
  delka = 50;
  celo = konec = 0;
  plna = false; prazdna = true;
  fronta = new int[50];
}
TFronta::TFronta(int velikost)
{
  delka = velikost;
  celo = konec = 0;
  plna = false; prazdna = true;
  fronta = new int[velikost];
}
```

```
bool TFronta::je_prazdna()
{
  return prazdna;
}

bool TFronta::je_plna()
{
  return plna;
}
```

```
bool TFronta::vloz(int prvek)
{
  prazdna = false;
  if (plna) return false;
  fronta[konec] = prvek;
  konec = (konec+1)%delka;
  if (konec == celo) plna = true;
  return true;
}
```

```
int TFronta::vyber()
{
  plna = false;
  if (prazdna) return -1;
  int prvek = fronta[celo];
  celo=(celo+1)%delka;
  if (celo==konec) prazdna = true;
  return prvek;
}

TFronta::~TFronta()
{
  delete [] fronta;
}
```

Poznámky:

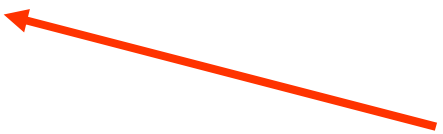
- spoléhat se na návratovou hodnotu -1 metody vyber(), která signalizuje pokus o výběr z prázdné fronty, není vhodné
  - nepoznám, zda byla vyjmuta -1 nebo bylo vybíráno z prázdné fronty
    - možné řešení: např. nastavovat chybovou proměnnou
    - poznáme i lepší možnost než chybová proměnná

Odstranění problému spočívá:

- ve správném používání knihovních funkcí
  - před každým voláním funkce vyjmi() aplikace otestuje, zda není fronta prázdná
  - např. `while (!f.je_prazdna())`
- v používání **výjimek**
  - rys objektových jazyků, který poznáme později

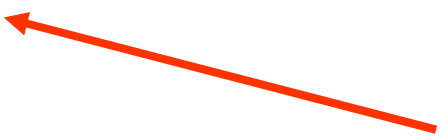
```
void main()
{
  int x;
  TFronta f1;

  f1.vloz(10);
  f1.vloz(-3);
  x = f1.vyber();
  if (f1.je_prazdna()) cout << "Prazdna";
  else cout << "Jeste tam neco je!";
};
```



zde je vyvolán destruktor

```
void main()
{
  int x;
  TFronta *f1;
  f1 = new TFronta(100);
  f1 -> vloz(10);
  f1 -> vloz(-3);
  x = f1 -> vyber();
  if (f1->je_prazdna()) cout << "Prazdna";
  else cout << "Jeste tam neco je!";
  delete f1;
};
```



zde je vyvolán destruktor

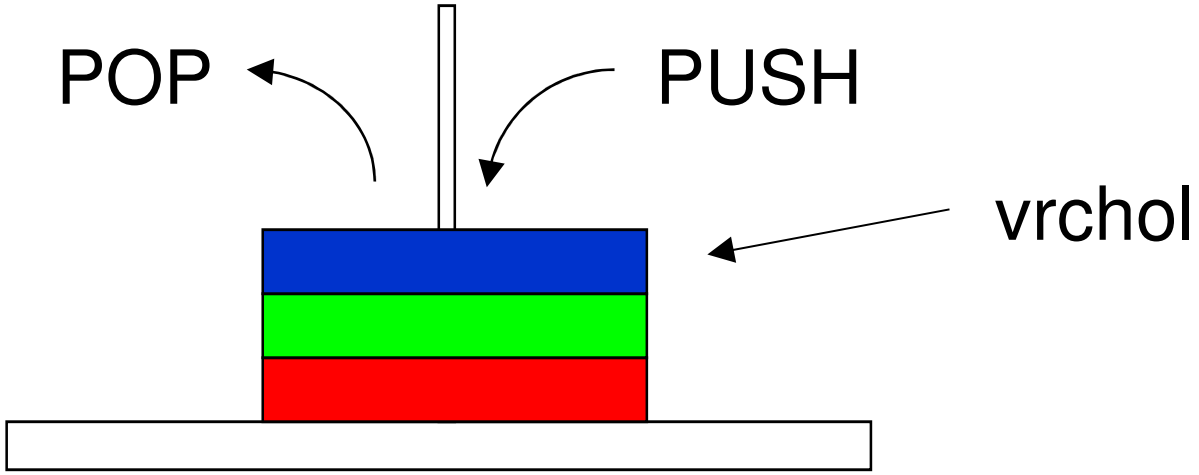
Úkol

Vytvořte třídu, implementující zásobník znaků konečné velikosti. Navrhněte vhodné metody a konstruktory. Délku specifikujte jako parametr konstruktoru (implicitní konstruktor vytvoří zásobník o velikosti 80). Zásobník otestujte v programu, který přečte znaky z klávesnice do konce řádku (znak po znaku) a vypíše je v opačném pořadí.

Zásobník (Stack)

- datová struktura typu **LIFO**
  - Last In - First Out
- nejprve se vybírá prvek, který byl vložen na *vrchol (top)* zásobníku jako poslední
- operace:
  - **PUSH** (uložení hodnoty na vrchol zásobníku)
  - **POP** (odebrání hodnoty z vrcholu zásobníku)

- někdy bývá implementována také operace **TOP**
  - zjištění hodnoty na vrcholu zásobníku bez odebrání prvku



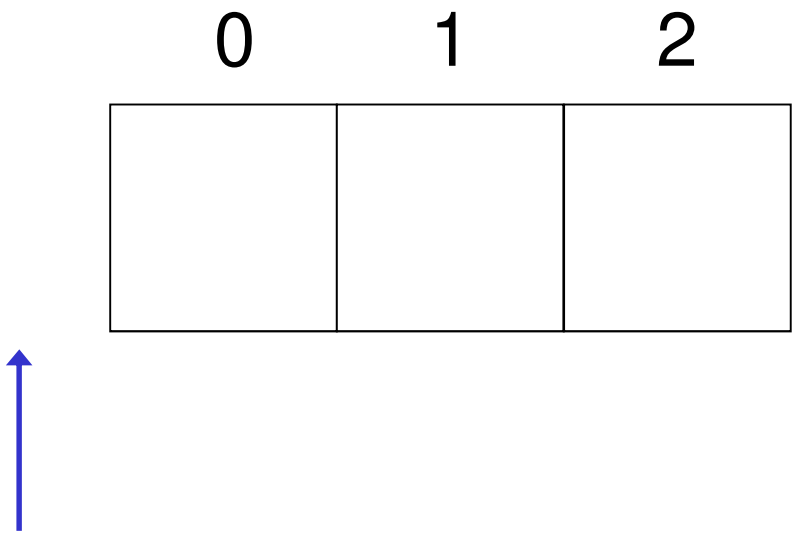
Možné implementace

- na principu spojového seznamu
  - „neomezená“ velikost zásobníku (omezená pouze velikostí dostupné paměti)
  - operace vložení prvku znamená vložení prvku na konec seznamu
  - operace výběr prvku je výběr z konce seznamu
  - zásobník je reprezentován ukazatelem na vrchol
  - prvek seznamu nese hodnotu a ukazatel na předchozí prvek v zásobníku
- polem (pevné nebo proměnné délky)
  - kruhová implementace není potřebná
  - stačí index na vrchol

Zásobník pomocí pole

- dvě varianty implementace
  - vrchol obsahuje index uloženého posledního prvku
    - při vložení nového prvku nejprve zvednu index a zapíši nový, při výběru vezmu prvek na tomto indexu a pak snížím index vrcholu; počáteční hodnota indexu vrcholu je -1
  - vrchol obsahuje index, kam mám vložit nový prvek
    - při vložení nového prvku zapíši prvek do pole na vrchol a následně jej zvednu o 1; při výběru nejprve snížím index o 1 a vrátím prvek; počáteční hodnota vrcholu je 0

Zásobník délky $n = 3$ (varianta 1)

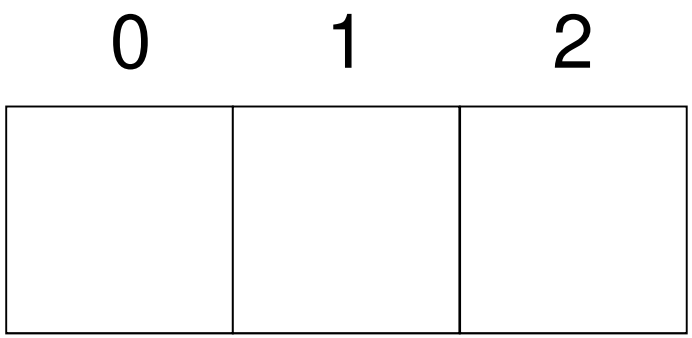


vrchol: -1

prázdný

plný

Zásobník délky $n = 3$ (varianta 1)



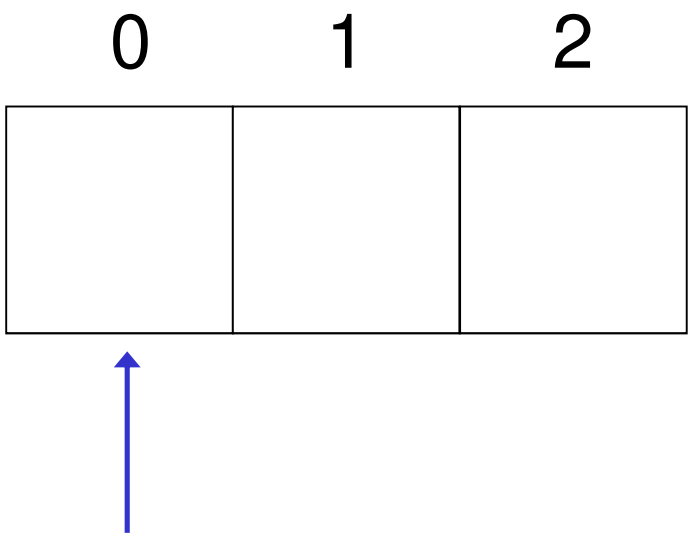
push(3)

vrchol: -1

prázdný

plný

Zásobník délky $n = 3$ (varianta 1)



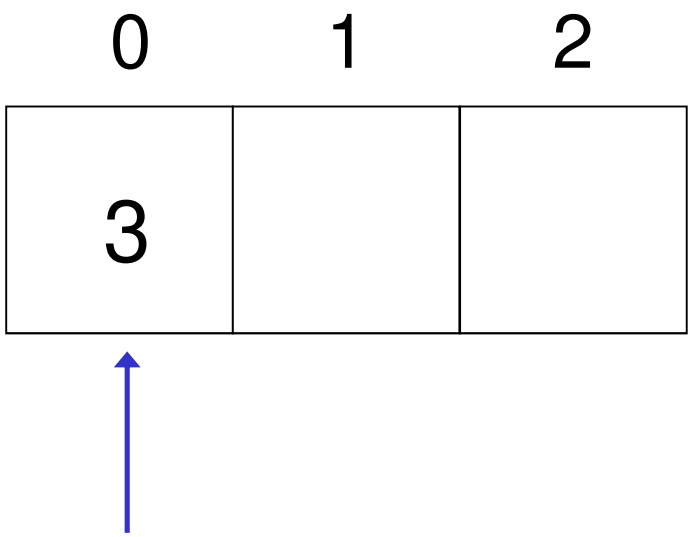
push(3)

vrchol: 0

prázdný

plný

Zásobník délky $n = 3$ (varianta 1)



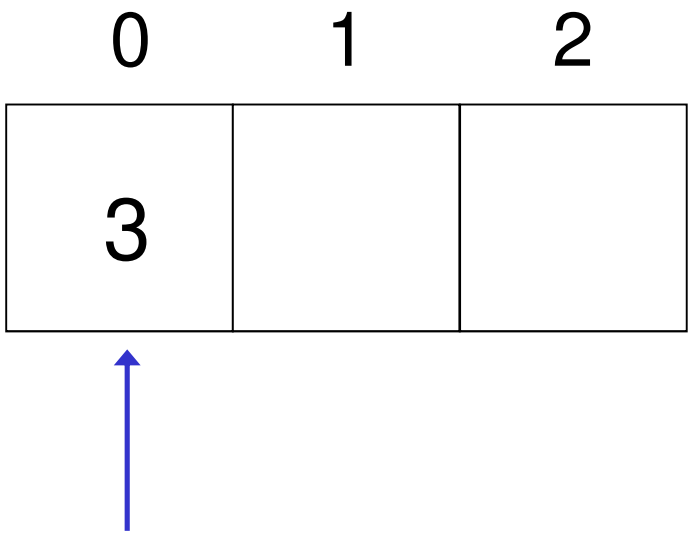
push(3)

vrchol: 0

prázdný

plný

Zásobník délky $n = 3$ (varianta 1)



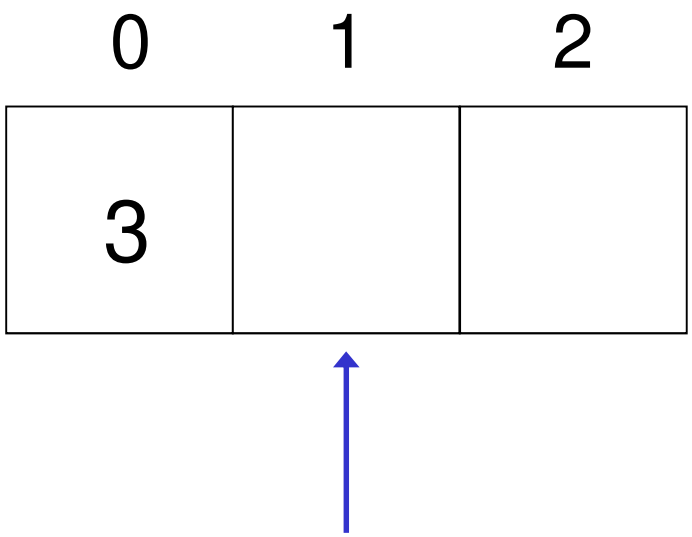
vrchol: 0

prázdný

plný

push(3)
push(8)

Zásobník délky $n = 3$ (varianta 1)



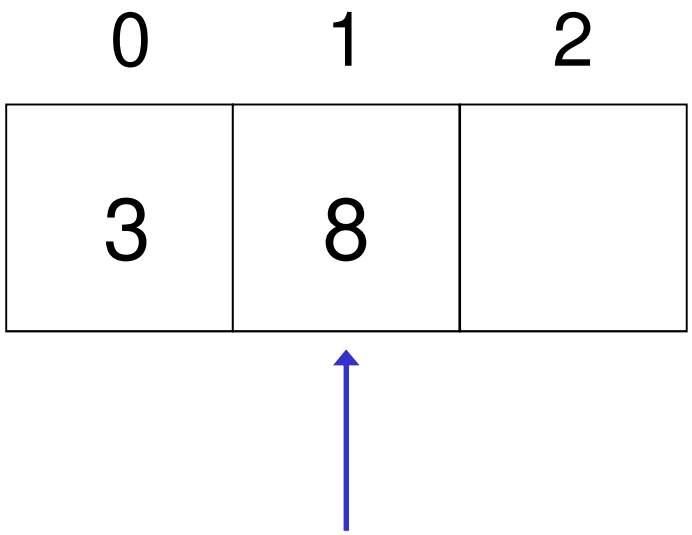
vrchol: 1

prázdný

plný

push(3)
push(8)

Zásobník délky $n = 3$ (varianta 1)



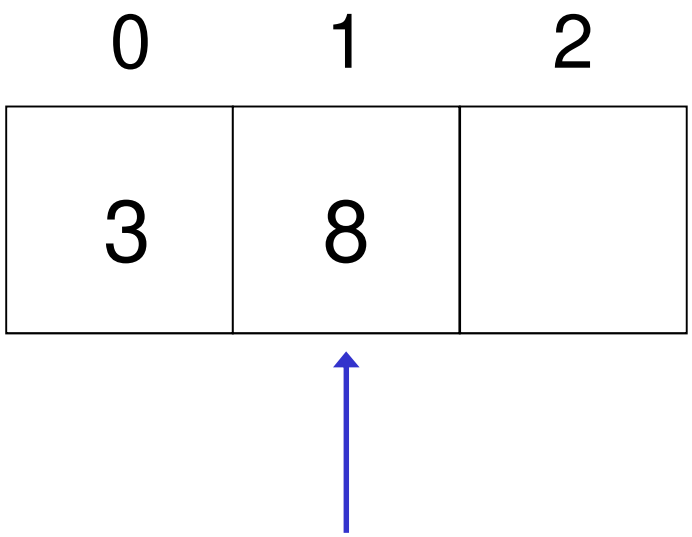
vrchol: 1

prázdný

plný

push(3)
push(8)

Zásobník délky $n = 3$ (varianta 1)



vrchol: 1

prázdný

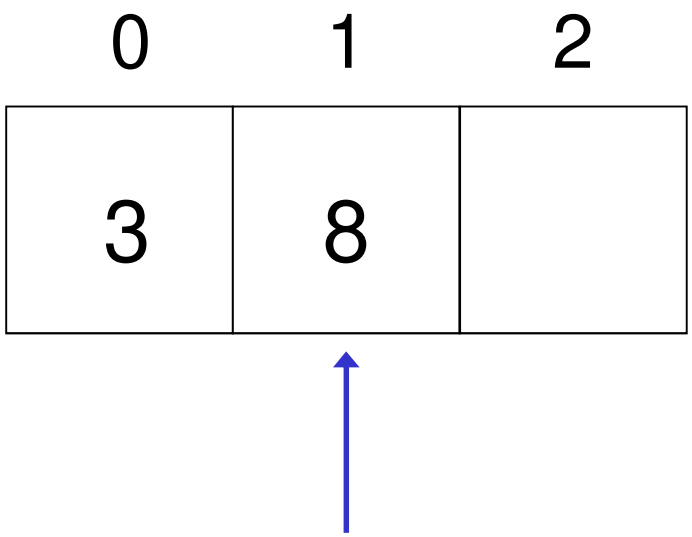
plný

push(3)

push(8)

top() vrátí hodnotu z vrcholu, tj. 8, ale neodebere ji

Zásobník délky $n = 3$ (varianta 1)



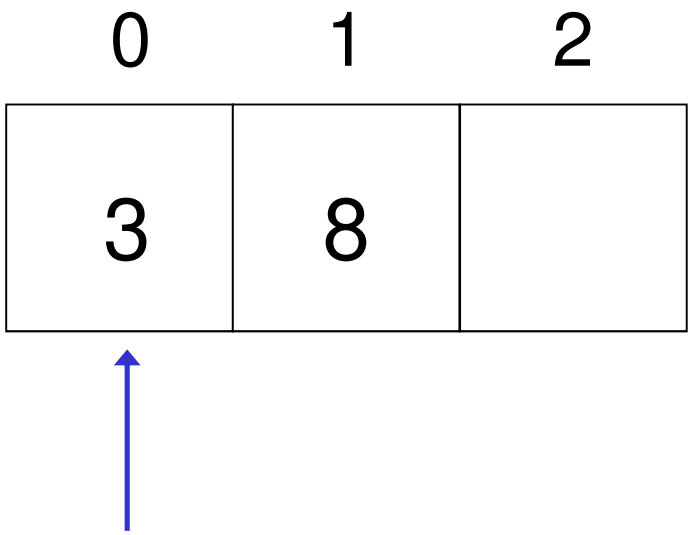
vrchol: 1

prázdný

plný

push(3)
push(8)
top()
pop() vrátí hodnotu z vrcholu, tj. 8, a odebere ji

Zásobník délky $n = 3$ (varianta 1)



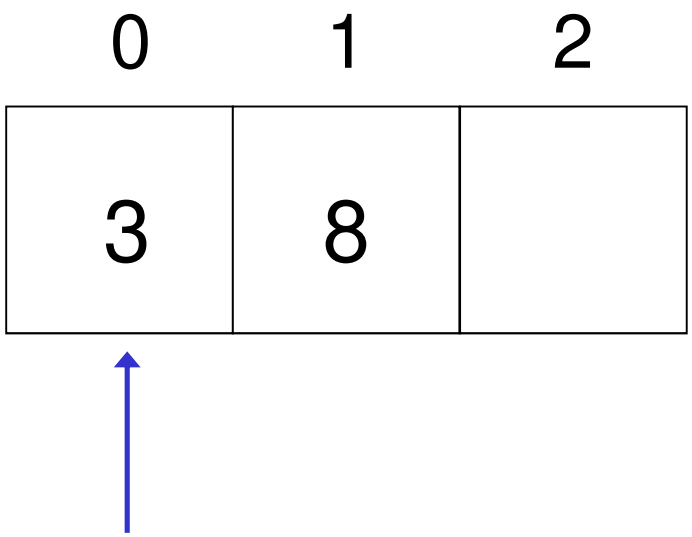
vrchol: 0

prázdný

plný

push(3)
push(8)
top()
pop()

Zásobník délky $n = 3$ (varianta 1)



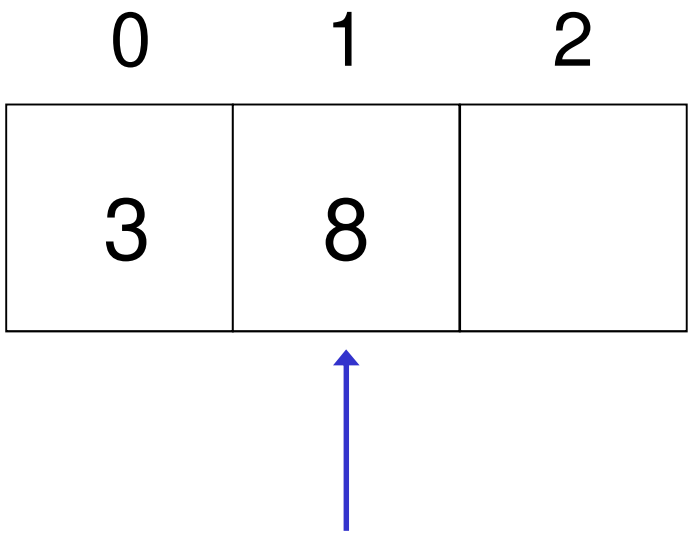
vrchol: 0

prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

Zásobník délky $n = 3$ (varianta 1)



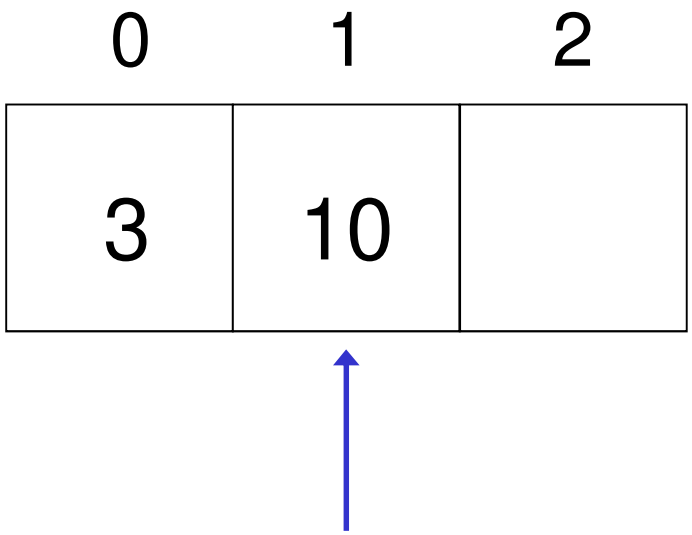
vrchol: 1

prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

Zásobník délky $n = 3$ (varianta 1)



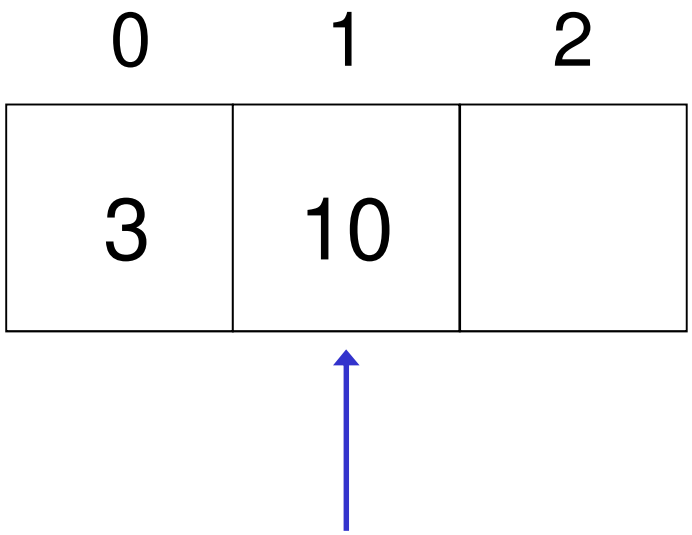
vrchol: 1

prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

Zásobník délky $n = 3$ (varianta 1)



vrchol: 1

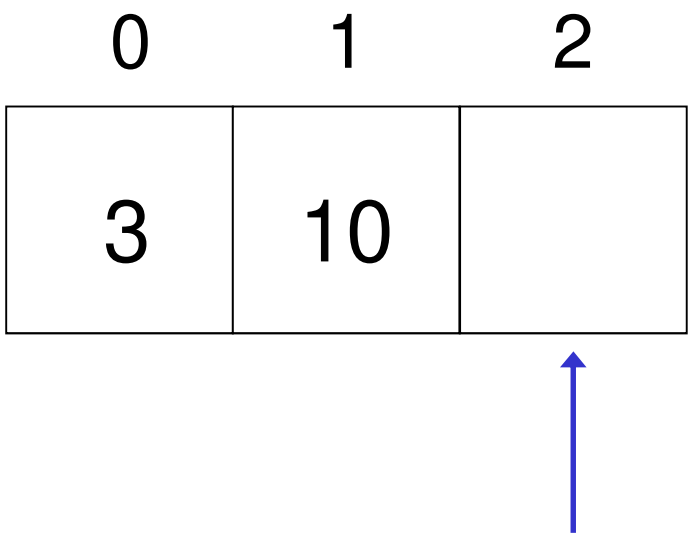
prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

push(13)

Zásobník délky $n = 3$ (varianta 1)



vrchol: 2

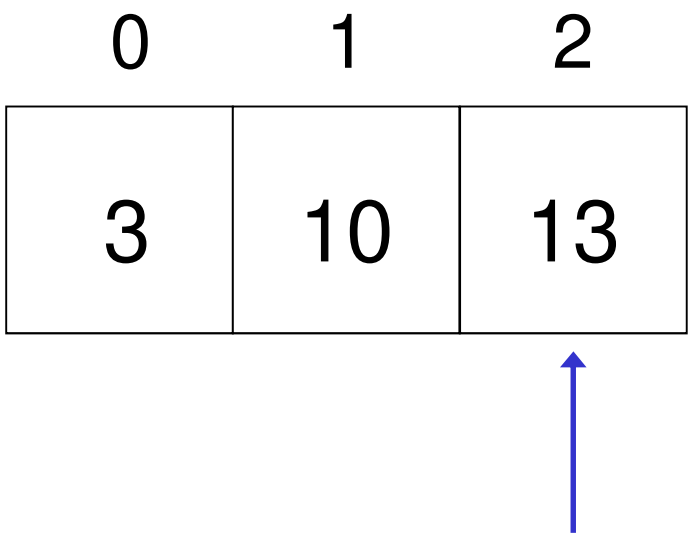
prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

push(13)

Zásobník délky $n = 3$ (varianta 1)



vrchol: 2

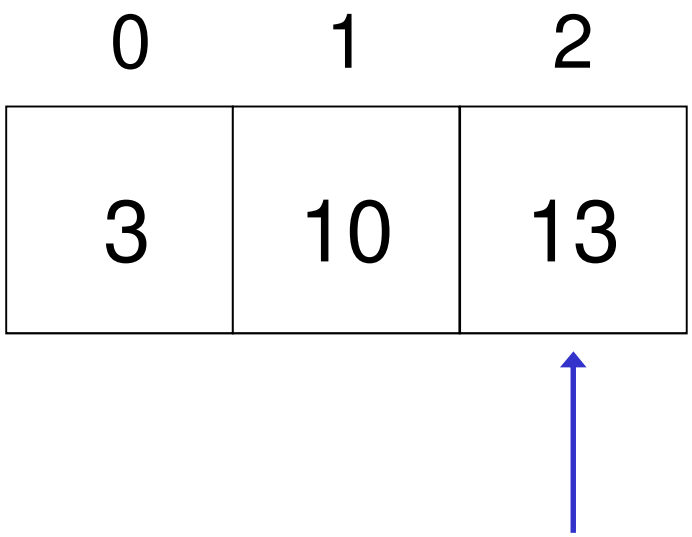
prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

push(13)

Zásobník délky $n = 3$ (varianta 1)



vrchol: 2

prázdný

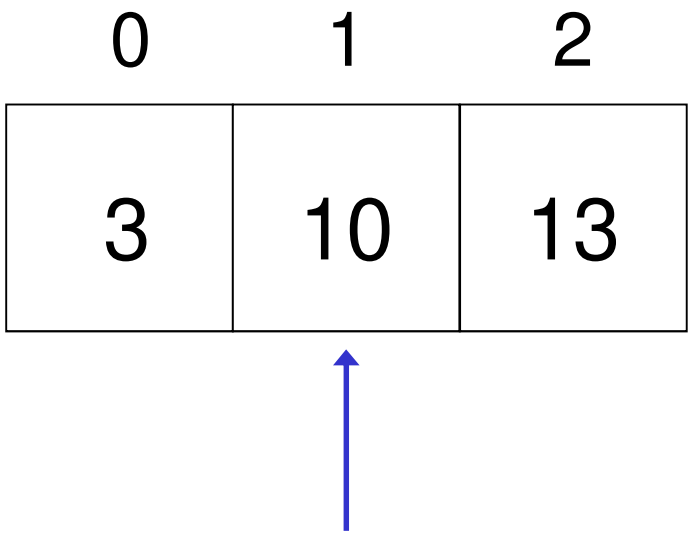
plný

push(3)
push(8)
top()
pop()
push(10)

push(13)
pop()

vrátí hodnotu 13 a odebere ji

Zásobník délky $n = 3$ (varianta 1)



vrchol: 1

prázdný

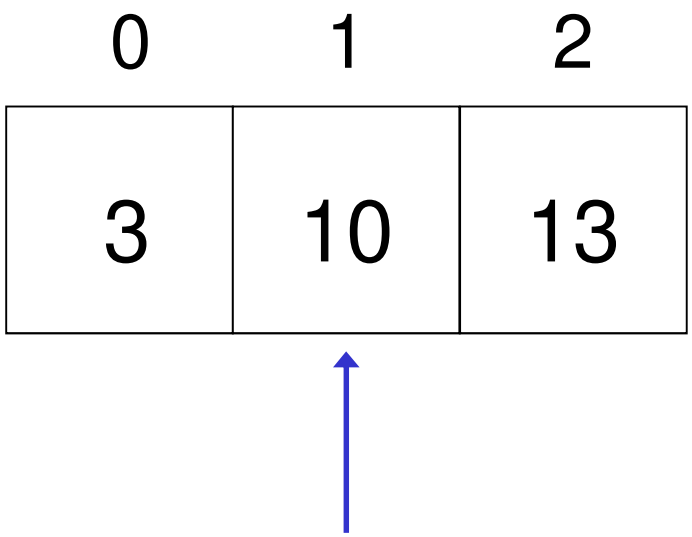
plný

push(3)
push(8)
top()
pop()
push(10)

push(13)
pop()

vrátí hodnotu 13 a odebere ji

Zásobník délky $n = 3$ (varianta 1)



vrchol: 1

prázdný

plný

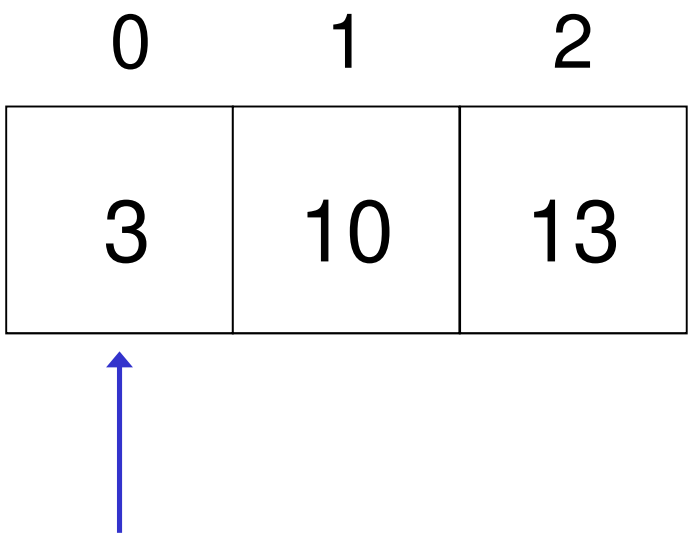
push(3)
push(8)
top()
pop()
push(10)

push(13)
pop()
pop()

vrátí hodnotu 13 a odebere ji

vrátí hodnotu 10 a odebere ji

Zásobník délky $n = 3$ (varianta 1)



vrchol: 0

prázdný

plný

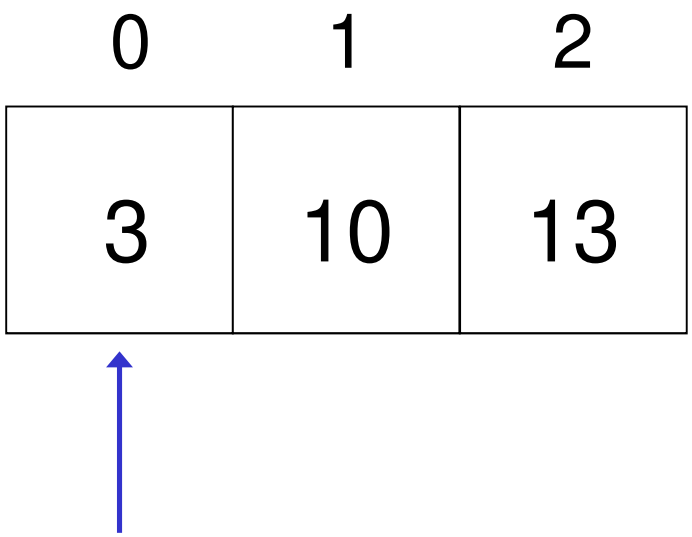
push(3)
push(8)
top()
pop()
push(10)

push(13)
pop()
pop()

vrátí hodnotu 13 a odebere ji

vrátí hodnotu 10 a odebere ji

Zásobník délky $n = 3$ (varianta 1)



vrchol: 0

prázdný

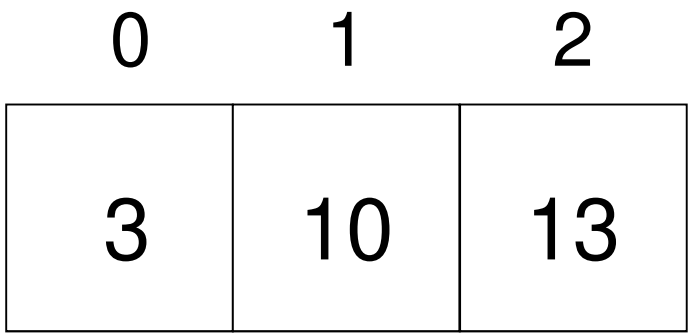
plný

push(3)
push(8)
top()
pop()
push(10)

push(13)
pop()
pop()
pop()

vrátí hodnotu 13 a odebere ji
vrátí hodnotu 10 a odebere ji
vrátí hodnotu 3 a odebere ji

Zásobník délky $n = 3$ (varianta 1)



vrchol: -1

prázdný

plný

push(3)
push(8)
top()
pop()
push(10)

push(13)
pop()
pop()
pop()

vrátí hodnotu 13 a odebere ji
vrátí hodnotu 10 a odebere ji
vrátí hodnotu 3 a odebere ji

Domácí úkol č. 1

Implementujte frontu s „neomezenou“ délkou. Implementaci proveďte na principu spojového seznamu.

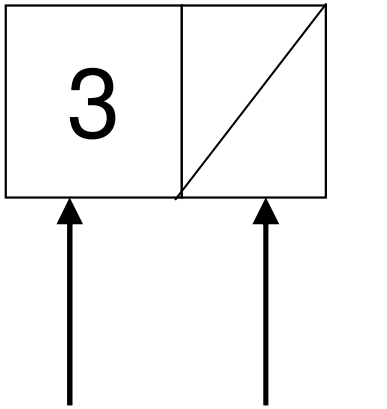
celo konec

```
NULL
```

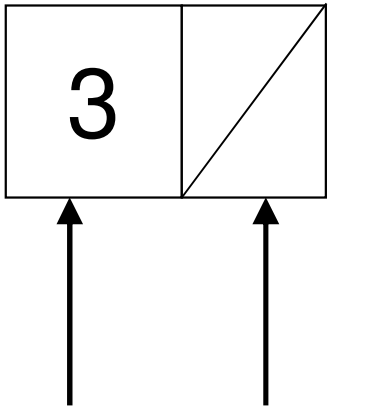
celo konec

```
NULL
```

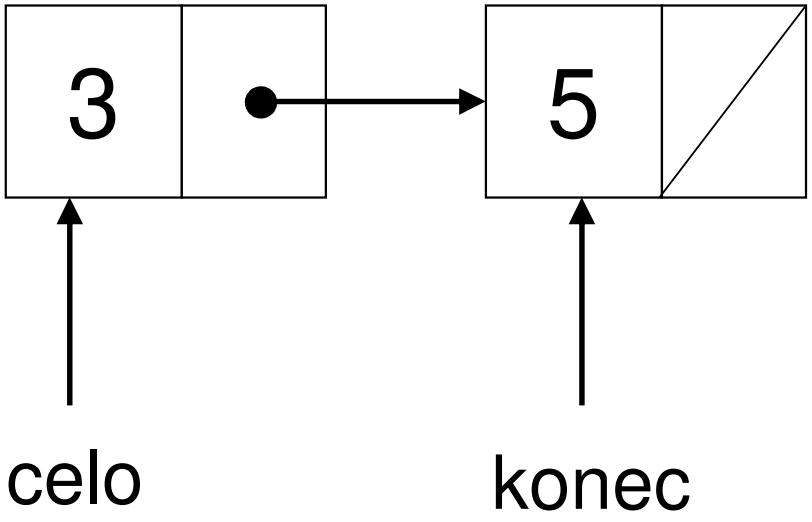
vloz(3)



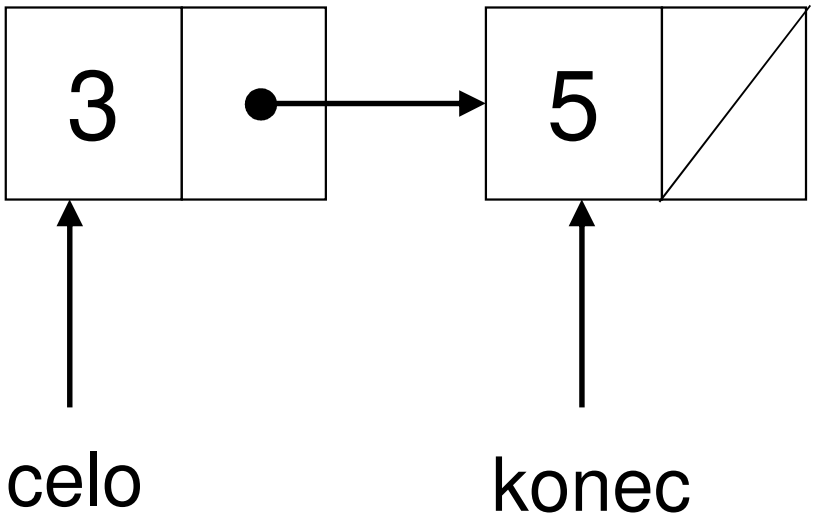
vloz(3)



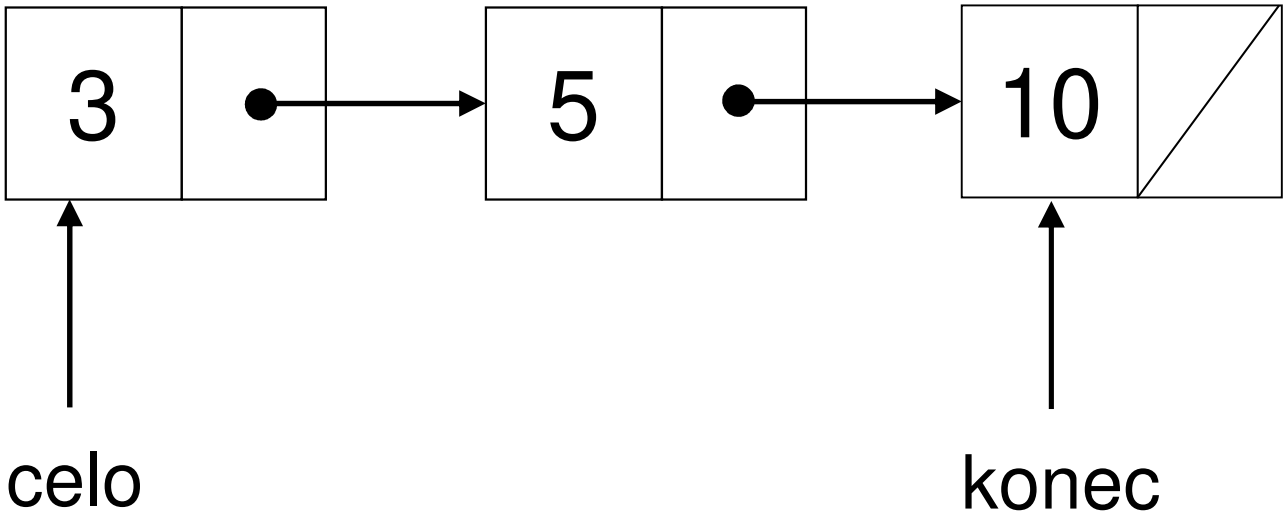
vloz(3)
vloz(5)



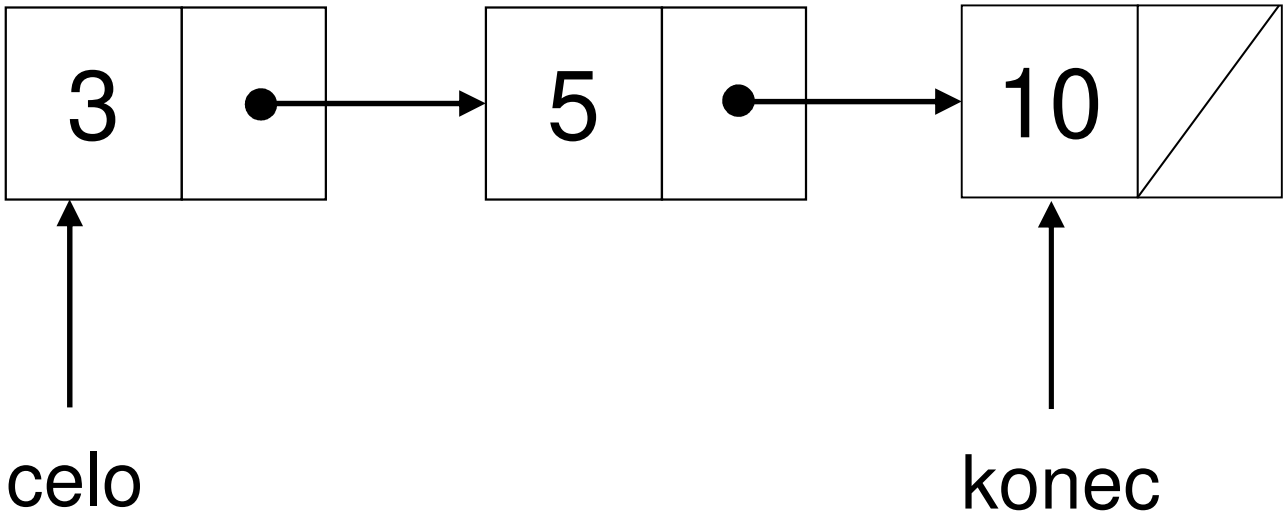
vloz(3)
vloz(5)



vloz(3)
vloz(5)
vloz(10)



vloz(3)
vloz(5)
vloz(10)

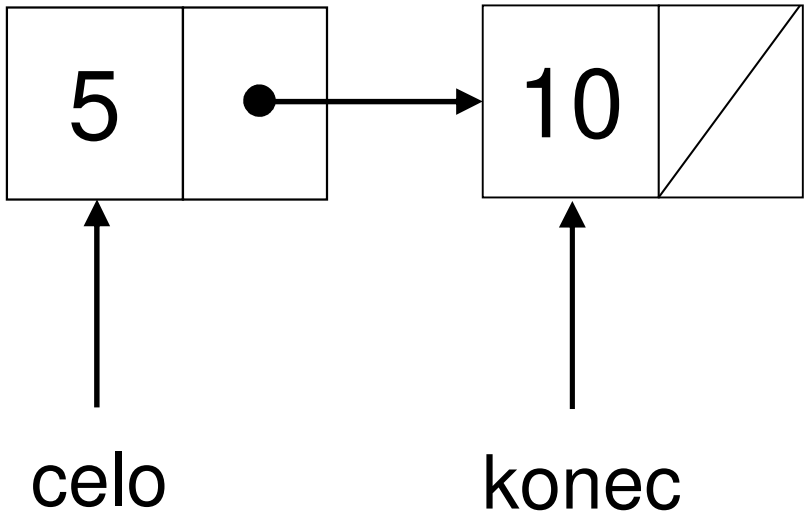


vloz(3)

vloz(5)

vloz(10)

vyjmi() – vrátí hodnotu z čela fronty - 3

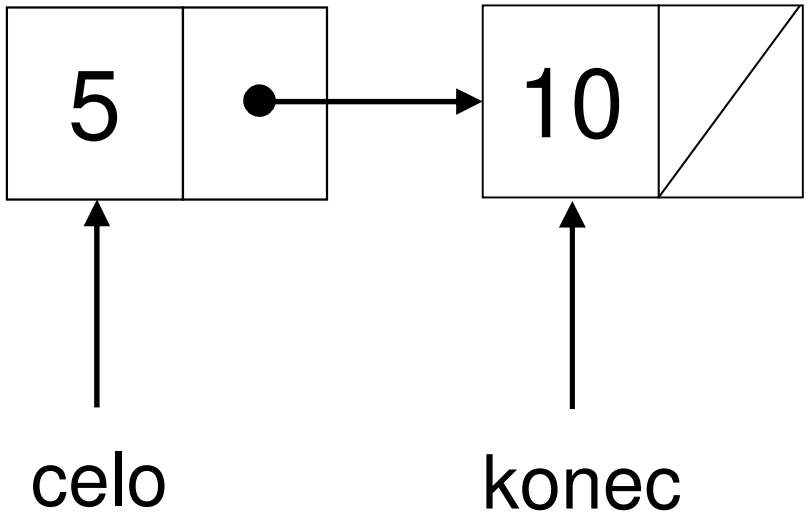


vloz(3)

vloz(5)

vloz(10)

vyjmi() – vrátí hodnotu z čela fronty - 3



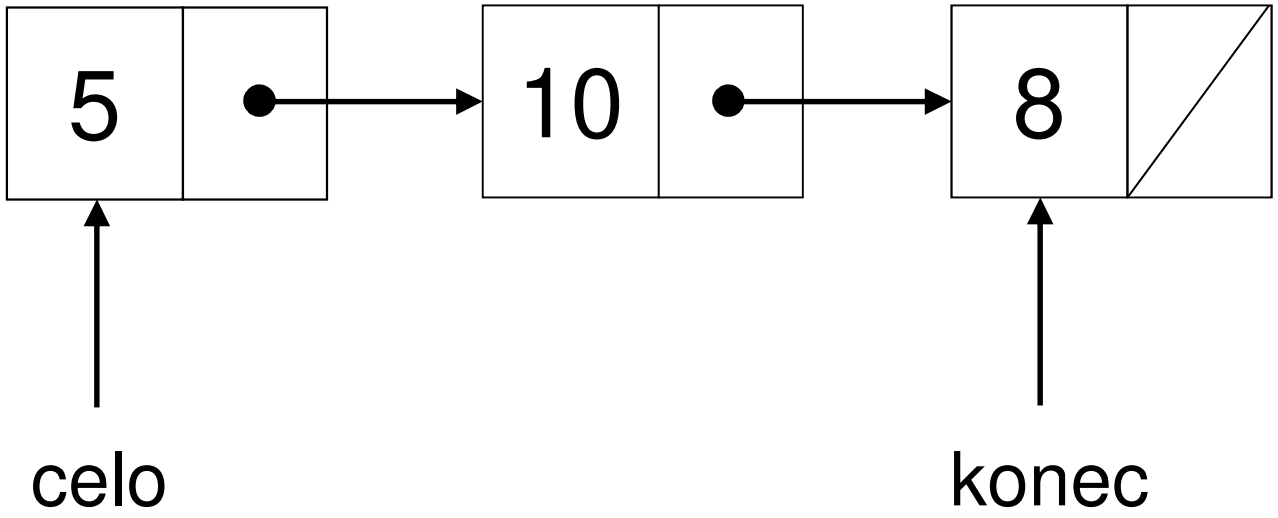
vloz(3)

vloz(5)

vloz(10)

vyjmi() – vrátí hodnotu z čela fronty - 3

vloz(8)



vloz(3)

vloz(5)

vloz(10)

vyjmi() – vrátí hodnotu z čela fronty - 3

vloz(8)

Návod:

```
class TFronta
{
  class TPolozka {
   int prvek; TPolozka *dalsi;
   TPolozka(int novy_prvek);
  }
  TPolozka *celo, *konec;
 public:
  int je_prazdna();
  atd.
}
```

Poznámka:

- konstruktor vložené třídy TPolozka implementujeme:

```
TFronta::TPolozka::TPolozka(int novy_prvek)
{
  prvek = novy_prvek;
  dalsi = NULL;
}
```