C++; delete Java;

Část 4: významné STL struktury

Kennny

srpen 2017

Významné struktury

- `string` - klasický řetězec
- `vector` - dynamické pole
- `list` - spojový seznam
- `map` - key-value úložiště (red-black strom, seřazeno)
- `set` - množina (seřazená)
- `unordered_map` - key-value úložiště (hash tabulka, neřazeno)
- `unordered_set` - množina (neřazená)

Významné struktury

- `queue` - fronta
- `stack` - zásobník
- `deque` - obousměrná fronta
- `priority_queue` - prioritní fronta
- a další...

std::string

- `#include <string>`
- obaluje dynamickou práci s řetězci
- odstiňuje nutnou realokaci, přetěžuje operátor +
- implementuje pomocné metody (substr, find, ...)
- pro porovnání přetěžuje operátory == a !=

Příklad

- Prostor pro příklad 04_a_string

Vsuvka: iterátor

- jednotný obalový prvek sloužící pro průchod
- každý STL kontejner ho má
- zpravidla se získává metodou `begin()` (popř. `rbegin()`, pokud to kontejner podporuje)
- přetěžuje operátor ++ (posun na další prvek v kontejneru)
- některé přetěžují operátor + (posun o N prvků v kontejneru), pokud to daný typ podporuje
- prvek na pozici iterátoru se získá dereferencí (přetěžuje unární operátor*)

```
int hodnota = *iter;
```

std::vector

- `#include <vector>`
- šablonový typ
- obaluje dynamické pole daného typu

```
std::vector<float> fVector;
std::vector<int> iVector;
std::vector<std::string> stringVector;
```

- rezervuje prostor "napřed"
- přetěžuje operátor []
- vnitřně jde o souvislou paměť

std::vector

- vkládání na konec (O(1))
  - metodou `push_back`
  - metodou `resize` a přes index
  - metodou `reserve` a následně `push_back`
- vkládání obecné (O(n))
  - metodou `insert`
- mazání z konce (O(1))
  - metodou `pop_back`
- mazání obecné (O(n))
  - metodou `erase` pomocí iterátoru
- přístup k prvkům (O(1))
  - metodou `at()`
  - operátorem []
- průchod (O(n))
  - přes indexy od 0 až do `size()`
  - iterátory od `begin()` až do `end()`
  - range-based for

Příklad

- Prostor pro příklad 04_b_vector

std::list

- `#include <list>`
- šablonový typ
- obaluje spojový seznam daného typu

```
std::list<float> fList;
std::list<int> iList;
std::list<std::string> stringList;
```

- analogie k LinkedList z Javy

std::list

- vkládání (O(1))
  - metodou `push_back` (na konec)
  - metodou `insert` (kamkoliv)
- mazání (O(1))
  - metodou `erase` pomocí iterátoru nebo rozsahu iterátorů
- přístup k prvkům (O(n))
  - iterace od začátku ruční
  - `begin()` a `std::advance`
- průchod (O(n))
  - iterátory od `begin()` až do `end()`
  - range-based for

Příklad

- Prostor pro příklad 04_c_list

std::map

- `#include <map>`
- šablonový typ
- obaluje key-value úložiště zadaných typů

```
std::map<int, float> ifMap;
std::map<long, std::string> intStringMap;
std::map<std::string, int> stringIntMap;
```

- implementace red-black stromem
- pro uložení musí typ být porovnatelný operátorem <
  - např. int, long, std::string toto splňují (std::string přetěžuje operátor <)
  - pozor ale na `const char*` - jde o adresu, tedy číslo

std::map

- vkládání (O(log n))
  - užitím klíče a operátoru []
  - metodou `insert` (vkládat pár)
- mazání (O(log n))
  - metodou `erase` (klíč nebo iterátor)
- přístup k prvkům (O(log n))
  - užitím klíče a operátoru [] (! pokud neexistuje, vytvoří ho !)
  - získání iterátoru metodou `find()`
- průchod (O(n))
  - iterátory od `begin()` až do `end()`
  - range-based for

std::map

- co když klíč nemá operátor < přímo, nemá ho ani přetížen, nebo jednoduše vyžadujeme jiné chování?
- –> dodefinujeme si buď operátor, nebo definujeme jiný komparátor mapy
- operátor< - známe
- komparátor - funkce, lambda funkce nebo funktor (struktura s přetíženým operátorem())

```
// const char* komparacni funktor pro mapu
struct constchar_comparator
{
    bool operator()(const char *a, const char *b) const
    {
        return strcmp(a, b) < 0;
    }
};
```

- prvky jsou stejné, pokud `!(a < b || b < a)`

Příklad

- Prostor pro příklad 04_d_map

std::set

- `#include <set>`
- šablonový typ
- obaluje množinu daného typu

```
std::set<float> fSet; // muze byt nesikovne (float aritmetika)!
std::set<int> iSet;
std::set<std::string> stringSet;
```

std::set

- vkládání (O(log n))
  - metodou `insert`
- mazání (O(log n))
  - metodou `erase` pomocí klíče nebo iterátoru
- detekce přítomnosti prvku (O(log n))
  - užitím `find()`
- průchod (O(n))
  - iterátory od `begin()` až do `end()`
  - range-based for

std::set

- pro komparátor platí stejná pravidla jako u mapy
- není vhodné pro nahrazování vectoru / listu, každý container má svůj účel

Příklad

- Prostor pro příklad 04_e_set

std::unordered_map

- `#include <unordered_map>`
- rozhraní stejné jako std::map
- rozdíl v implementaci - hash tabulka
- v podstatě analogie k HashMap v Javě
- přidání, přístup k prvku a mazání průměrně O(1)
- nezaručuje pořadí prvků
- proti obyčejné std::map zabírá více paměti

std::unordered_set

- `#include <unordered_set>`
- úplně stejná analogie jako std::map a std::unordered_map

std::queue

- `#include <queue>`
- klasická implementace fronty
- důležité metody:
  - `push()` - vložit prvek
  - `front()` - přední prvek (kandidát na výběr)
  - `pop()` - zahození předního prvku
  - `empty()` - je fronta prázdná?
  - `size()` - počet prvků ve frontě

```
// vyprazdneni (zpracovani) fronty
while (!q.empty())
{
    TypPrvku& el = q.front();

    // prace s prvkem

    ique.pop();
}
```

std::stack

- `#include <stack>`
- klasická implementace zásobníku
- důležité metody:
  - `push()` - vložit prvek
  - `top()` - prvek na vrcholu (kandidát na výběr)
  - `pop()` - zahození předního prvku
  - `empty()` - je fronta prázdná?
  - `size()` - počet prvků ve frontě

```
// vyprazdneni (zpracovani) zasobniku
while (!q.empty())
{
    TypPrvku& el = q.top();

    // prace s prvkem

    ique.pop();
}
```

Příklad

- Prostor pro příklad 04_f_misc

Konec 4. části

```
exit(0);
```