3. Standardní knihovna, knihovna standardních šablon

B2B99PPC – Praktické programování v C/C++

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – STL kontejnery
  - Obecné vlastnosti
  - Datový typ std::string
  - Iterátory
  - Sekvenční kontejnery
  - Asociativní kontejnery
  - Další STL kontejnery
  - Generické funkce

Část I

STL kontejnery

I. STL kontejnery

STL kontejnery

- Kontejner (kolekce) je abstraktní datový typ určený na organizované skladování prvků konkrétního typu podle určitých pravidel.

  Ve standardní knihovně je připravena celá řada šablon užitečných kontejnerů.

- Kontejnery se od sebe významně liší způsobem přístupu k prvkům, možnostmi vkládání a rušení prvků a také časovou složitostí jednotlivých operací.
- Mezi kontejnery můžeme počítat i typ `std::string` pro zpracování řetězců znaků.
- Všechny STL kontejnery mají public kopírující konstruktor a operátor přiřazení (=).

  Více na příští přednášce.

- Základní dělení:
  - **Sekvenční kontejnery** – sekvenční (postupný) přístup k prvkům
  - **Asociativní kontejnery** – libovolný (náhodný) přístup k prvkům

I. STL kontejnery

Inicializace řetězce

Dobře

```
std::string s1("Ahoj");
std::string s2(8, 'x');
std::string mesic = "Leden";
// implicitní volání konstruktoru
// std::string mesic("Leden");
```

lec03/01-string-init.cpp

Jak spíše ne...

```
string foo = new string;
// neinicializovaný objekt
string foo = string("Ahoj");
// anonymní objekt, který se zkopíruje a pak zničí
// jako funguje to, no...
```

Obecné vlastnosti

- Není nezbytně zakončen terminačním znakem
- Na rozdíl od **C** není identifikátor ukazatelem
- Lze využít operátor `[]` pro přístup k jednotlivým znakům řetězce
- Znaky jsou indexovány od `0` do `délka - 1`

```
std::string s1("ahoj");
std::cout << s1[1];
```

- Přiřazení pomocí operátoru `=`

```
s2 = s1;  // vytvoření separátní kopie
```

- Přístup pomocí `[]` je plnohodnotný, znaky řetězce lze i měnit

```
s1[1] = s2[0];
```

Členské metody

- Délka

```
s1.length();
```

- Přiřazení – http://www.cplusplus.com/reference/string/string/assign.html

```
s2.assign(s1);  // odpovídá s2 = s1
s2.assign(s1, start, N); // zkopíruje N znaků od indexu start
```

- Přístup ke znakům – může vyvolat výjimku out_of_range

```
s2.at(0) = s3.at(2); // provádí kontrolu délky
```

lec03/02-string-elements.cpp

- Spojování

```
s3.append("pet");
s3 += "pet";
s3.append(s1, start, N);
```

Členské metody

- Porovnávání

```
// přetížené operátory ==, !=, <, >, <=, >=
s1.compare(s2)
// vrací 0 v případě stejných řetězců, porovnává znak po znaku
s1.compare(start1, length1, s2, start2, length2);
// porovná části řetězců
s1.compare(start1, length1, s2) // porovná část s1 s celým s2
```

- Části řetězců

```
s1.substr(start, N);  // vrátí část řezezce od indexu start o délce N
```

- Prohození

```
s1.swap(s2);
```

Členské metody

- Charakteristiky

```
s1.size()
s1.length()
// počet znaků v řetězci
s1.capacity()
// počet znaků, které mohou být vloženy bez realokace
s1.max_size()
// maximální možná délka řetězce
// při překročení se vyvolá výjimka length_errorexception
s1.empty()
// vrací true, pokud je řetezec prázdný
s1.resize(newlength)
// změní velikost na novou newlength
```

Členské metody

- Hledání
  - v případě nalezení je vrácen index
  - v případě nenalezení `string::npos`

```
s1.find(s2)           // vrací pozici s2 v s1
s1.find(s2, pos)  // vrací pozici s2 v s1 při hledání v od pos
s1.rfind(s2)          // hledá zprava doleva
s1.find_first_of(s2)  // vrací první výskyt s2
s1.find_first_of("abc") // vrací index prvního 'a', 'b' nebo 'c'
s1.find_last_of(s2)      // poslední výskyt s2
s1.find_first_not_of(s2)  // první výskyt není v s2
s1.find_last_not_of(s2) // poslední výskyt znaku který není v s2
```

lec03/03-string-find.cpp

Členské metody

- Nahrazení

```
s1.replace(begin, N, s2)
// begin:  index s1, kde začne funkce nahrazovat
// N: počet znaků s1, které budou změněny
// s2:  nahrazující řetězec
s1.replace(begin, N, s2, index, num)
// index:  prvek s2, kde začíná náhrada
// num:  počet prvků s2, které budou použity k nahrazení
```

- Vkládání

```
s1.insert( index, s2 )
// vloží s2 na index, nepřepisuje znaky s1
s1.insert(index1, s2, index2, N);
// vloží část řetězce s2 na pozici index1 v s1
```

Členské metody

- Konverze

```
s1.copy(ptr, N, index)
// zkopíruje N znaků do pole char* ptr
// začíná na pozici index řetězce s1
// ukončuje řetězec NULL

s1.c_str()
// vrací const char*
// ukončuje řetězec NULL

s1.data()
// vrací const char*
// NEukončuje řetězec NULL
```

I. STL kontejnery

Iterátor

- Iterátor je datový typ definovaný v STL
- Odkazuje na nějaká data či pozici,
- Data mohou být v nějaké kolekci (`vector`, `set`, ...), nebo např. v souboru,
- Iterátor např. umožní čtení celých čísel z I/O proudu (`std::cin`),
- Iterátor je typicky generická třída,
- Rozhraní iterátoru umožňuje čtení (dereference), posun vpřed (`++`) a porovnávání dvojice iterátorů (`==`, `!=`),
- Některé iterátory mají ještě další rozhraní, viz dále.

Příklad – součet čísel na standardním vstupu

```cpp
#include <iostream>
using namespace std;
int main ()
{
  int sum = 0, x;
  while ( cin >> x )
    sum += x;
  cout << sum << endl;
  return 0;
}
```

```cpp
#include <iostream>
#include <numeric>
#include <iterator>
using namespace std;
int main ()
{
  cout << accumulate (
    istream_iterator<int> (cin),
    istream_iterator<int> (), 0);
  cout << endl;
}
```

lec03/04-accumulate.cpp

- `istream_iterator<int> (cin)` – při dereferenci čte data typu integer ze std. vstupu
- `istream iterator<int> ()` – znamená konec vstupu (EOF).

Příklad – součet čísel na standardním vstupu

- Generická funkce `accumulate`:
  - generická funkce definovaná v STL,
  - slouží k výpočtu součtu dat, rozsah vstupních dat je daný dvojicí iterátorů,
  - funkce pomocí iterátorů projde zadaný rozsah dat,
  - získané hodnoty přičte k existující hodnotě,
  - počáteční hodnota je daná posledním parametrem (zde 0),
  - výsledek je návratovou hodnotou funkce (zde požadovaný součet).
- Podobných generických funkcí pracujících se dvojicemi iterátorů je v STL celá řada:
  - kopírování,
  - mazání,
  - filtrování,
  - řazení,
  - ...

I. STL kontejnery

STL kontejnery – array

- Zapouzdřené pole fixní velikosti.
- Lze kontrolovat indexy, snadné kopírování.

```
#include <array>
#include <iterator>
// ..
array<int, 10> x;
x.fill (-1); // inicializace pole konstantou
x[3] = 100;  // nekontrolovaný přístup
for (array<int,10>::size_type i = 0; i < x.size(); i++)
  x[i] += 100;
copy (x.begin(), x.end(), ostream_iterator<int> (cout, "\n"));
array<int, 10> y(x); // kopie pole
```

lec03/05-std-array.cpp

STL kontejnery – vector

- Zapouzdřené pole proměnné velikost.
- Lze kontrolovat indexy, snadné kopírování, rozšiřování.

```
vector<int> x;
x.resize (10); // 0 0 0 ...  0
x.push_back (100);
x.insert (x.begin() + 5, 200); // vložení čísla
x[3] = 120; // změna prvku, bez kontroly
x.at (4) = 150; // změna prvku, kontrola
copy (x.begin(), x.end (), ostream_iterator<int> (cout, "\n"));
vector<int> y (x.begin () + 1, x.begin () + 9); // kopírování v rozsahu
sort (y.begin(), y.end ());
```

lec03/06-std-vector.cpp

STL kontejnery – deque

- Oboustranná fronta, dokáže nahradit stack i queue.
- Lze přidávat i odebírat z obou konců, přístup přes index.

```
deque<int> x;
x.push_back (100); x.push_front (200); x.insert (x.begin() + 1, 500);
// copy, read, display & pop
for (deque<int> y (x); !y.empty (); y.pop_front())
  cout << y.front () << endl;
x.erase (x.begin () + 1, x.begin () + 3 );
// iterate in reverse direction
deque<int>::reverse_iterator it;
for (it = x.rbegin(); it != x.rend (); ++it)
  cout << *it << endl;
```

lec03/07-std-deque.cpp

STL kontejnery – list

- Obousměrný spojový seznam.
- Vkládání/odebírání prvku z libovolné z pozice (začátek, konec, iterátorem).

```
list<int> x;
x.push_back (100); x.push_back (200); x.push_front (300);
list<int>::iterator pos = x.begin ();
pos++;
x.insert (pos, 400);
pos++;
x.erase (pos);
// iterate forward
list<int>::iterator it;
for (it = x . begin(); it != x.end (); ++it)
  cout << *it << endl;
```

lec03/08-std-list.cpp

I. STL kontejnery

STL kontejnery – set

- Množina prvků (prvek buď je nebo není obsažen).
- Vkládání/mazání/testování přítomnosti prvku.

```
set<int> x;
x.insert (20);
x.insert (100);
x.insert (1000);
cout << (x.count (20) == 1 ? "present" : "not present" ) << endl;
set<int>::iterator pos = x.find (1000);
if (pos != x.end ()) x.erase (pos);
set<int>::iterator it;
for (it = x . begin(); it != x.end (); ++it)
  cout << *it << endl;
```

lec03/09-std-set.cpp

STL kontejnery – map

- Tabulka (klíč – hodnota).
- Vkládání/mazání/čtení prvku.

```
map<string,int> x;
x.insert (make_pair ("test", 10));
x["key"] = 20;
x["testkey"] = x["test"] + x["key"];
map<string,int>::const_iterator it;
for (it = x . begin(); it != x . end (); ++it )
  cout << it -> first << "->" << it -> second << endl;
map<string,int>::iterator pos = x.find ("test");
cout << (pos != x.end () ? "present" : "not present") << endl;
x.erase (pos);
```

lec03/10-std-map.cpp

I. STL kontejnery

Další STL kontejnery

- `forward_list`, C++ 11
  - jednosměrný spojový seznam,
  - obdoba `list` s nižší paměťovou režií, ale s omezením operací,

    Např. nelze snadno vkládat před pozici iterátoru.
  - sekvenční kontejner, ForwardIterator.
- `stack`
  - zásobník,
  - implementován jako wrapper na deque (příp. list nebo vector),
  - omezení rozhraní na LIFO,
  - sekvenční kontejner, nemá iterátor.
- `queue`
  - fronta, implementována jako wrapper na deque příp. list,
  - omezení rozhraní na FIFO,
  - sekvenční kontejner, nemá iterátor.

Další STL kontejnery

- `priority_queue`
  - fronta s prioritami,
  - prioritní ukládání je dosaženo tím, že se data ukládají do datové struktury halda,
  - sekvenční kontejner, nemá iterátor.
- `multiset`, `multimap`
  - třídy odpovídají `set` a `map`,
  - stejná hodnota klíče může být uložena vícekrát,
  - `#include <map>`, `#include <set>`
  - asociativní kontejnery, ForwardIterator.
- `bitset`
  - pole `bool` hodnot zadané velikosti,
  - kompaktní uložení (po jednotlivých bitech),
  - nemá iterátor.

Iterátory

- Iterátory v STL slouží pro:
  - procházení kontejneru,
  - určení místa v kontejneru

    Např. kam má být vložen prvek, který prvek smazat.
  - určení rozsahu prvků, které má nějaká funkce zpracovávat

    Řadit, projít, sečíst, zkopírovat, ...
  - vyhledávání v kontejneru.
- Iterátory jsou zpravidla šablony tříd, jejich přetížené rozhraní se chová jako ukazatel:
  - `*` – dereference (přístup k prvku),
  - `==` a `!=` – porovnání iterátorů (stejná/různá pozice),
  - `++` – posun vpřed (prefix i postfix forma, prefix bývá efektivnější).
- Inicializace
  - "na začátek" – funkční hodnota členské funkce `begin()`,
  - "za konec" – `end()`.
- Některé kontejnery neumožňují iterování: `stack`, `queue`, `priority_queue`.

Iterátory

InputIterator – dereference pro čtení, posuv vpřed a test na (ne)rovnost. Nelze restartovat (2x dereferencovat). Příklad: `istream`.

OutputIterator – dereferenci pro zápis, posuv vpřed a test na (ne)rovnost. Nelze restartovat. Příklad: `ostream`.

ForwardIterator – jako InputIterator, navíc dereference čtení/zápis i vícekrát na stejné pozici. Příklad: `forward_list`.

BidirectionalIterator – jako ForwardIterator, navíc možnost posunu zpět operátorem `--`. Příklady: `list`, `set`, `map`, ...

RandomAccessIterator – jako BidirectionalIterator, navíc možnost posunu vpřed/zpět o zadaný počet pozic (operace `+=` a `-=`), indexace operátorem `[n]`, rozdíl iterátorů, relační operátory. Příklad: `vector`, `deque`.

Operace s interátory

advance (iterator i, int distance) – zvyšuje nebo snižuje pozici iterátoru `i` o hodnotu `distance`

distance (iterator first, iterator last) – vrací vzdálenost mezi dvěma iterátory

next (iterator i, int n) – vrací iterátor následující o `n` prvků za iterátorem `i`

prev (iterator i, int n) – vrací iterátor předcházející o `n` prvků před iterátorem `i`

begin () – vrací ukazatel na první prvek kontejneru

end () – vrací ukazatel na první prvek za posledním prvkem kontejneru

I. STL kontejnery

Generické funkce

`find(st, en, x)` – hledá první výskyt prvku `x` v rozsahu `st` až `en`,

`copy(st,en,dst)` – kopíruje prvky z rozsahu `st` až `en` do cíle `dst` (a dále), volbou iterátoru `dst` lze použít i pro vstup / výstup dat, přesunutí či přidání,

`transform(st,en,dst,fn)` – kopíruje jako `copy`, prvky při kopírování transformuje `fn`,

`sort(st,en,fn)` – seřadí prvky v rozsahu `st` až `en`, kritériem řazení je `fn`.

Generické funkce

Další běžné úlohy a podpůrné STL funkce:

- binární vyhledávání: lower bound, upper bound, binary search,
- slučování (merge) a operace založené na slučování (průnik, sjednocení seřazených polí): `merge`, `set_union`, `set_intersection`,
- práce s datovou strukturou halda (heap): `make_heap`, `push_heap`, `pop_heap`,
- minimum a maximum: `min_element`, `max_element`,
- agregace: `accumulate`,
- test vlastnosti pro všechny prvky: `all_of`, `any_of`, `none_of`.