Wydział Elektroniki i Technik Informacyjnych
POLITECHNIKA WARSZAWSKA

Biblioteka standardowa (wstęp)

Iteratory

Politechnika Warszawska

Biblioteka standardowa (widok z wysoka)

- Biblioteka obejmuje 79 plików nagłówkowych (bez plików specyficznych dla danego środowiska; VC: ~200)
- 26 plików (z 79) przenosi akcesoria języka C do C++ (konwencja: plik w C **<nazwa.h>** ⇒ **<cnazwa>**)
- Wszystkie nazwy z biblioteki (z wyjątkiem **makr** i operatorów **new** i **delete**) są umiejscowione w przestrzeni nazw **std**
- Semantycznie biblioteki są podzielone na kilkanaście kategorii (obsługa tekstów, strumienie, kontenery, algorytmy, ustawienia regionalne, wyrażenia regularne, diagnostyka, ...)
- Pliki nagłówkowe bibliotek C często obejmują wsparcie podstawowe dla paru kategorii usług (np. **<cstdlib>**, **<ctime>**)

Politechnika Warszawska

Biblioteka standardowa (cd1)

Biblioteka `<string>`: wsparcie dla przetwarzania tekstów
Klasa string definiuje ~100 operacji na łańcuchach znaków

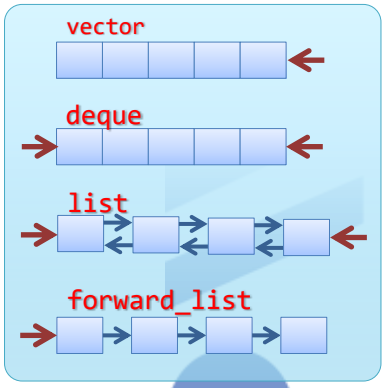
```
typedef basic_string<char> string;      // specjalizacja szablonu
typedef basic_string<wchar_t> wstring;  // inna specjalizacja
typedef basic_string<char16_t> u16string;
typedef basic_string<char32_t> u32string;
```

```
#include <string>   // Mini-przykład
#include <iostream>
using namespace std;

int main()
{ string s1 = "ABRA", s2 = "KAD";
  s2 += s1; s1 += s2;
  for(unsigned p=0; p<s1.size(); ++p)
    if(s1[p]=='A') s1.replace(p, 1, "[a]");
  cout << s1 << endl;
  return 0;
}
```

```
[a]BR[a]K[a]D[a]BR[a]
```


Kontenery i algorytmy



- Kontenery to zbiory obiektów
  - biblioteka implementuje kontenery sekwencyjne, asocjacyjne, itp.
- Dostęp do elementów w kontenerach zapewniają Iteratory
  - umożliwiają dostęp do obiektu kontenera
  - definiują obiekt „pusty” / „brak obiektu” (np. indeks -1, nullptr)
  - dostęp jednokierunkowy (forward_list)
  - dostęp dwukierunkowy (list)
  - dostęp swobodny (vector, tablica)
- Biblioteka definiuje algorytmy, do opisu których można posługiwać się pojęciem kolekcji, iteratora, zakresu.
  - Zakres można definiować jako iterator początku i końca np. search(), count(), max(),...

Biblioteka standardowa (cd2)

Kontenery - zorganizowane repozytoria obiektów	
<vector>	sekwencyjne
<array> (C++11)	
<deque>	
<list>	
<forward_list> (C++11)	
<set>	asocjacyjne uporządkowane
<map>	
<unordered_map>	asocjacyjne z haszowaniem
<unordered_set> (C++11)	
<queue>	adaptery kontenerów sekwencyjnych
<stack>	

Biblioteka standardowa (cd3)

for_each	Operacje niemodyfikujące na kolekcjach elementów
equal	
search	
count	
copy_backward	Operacje na kolekcjach elementów (z modyfikacją)
move (C++11)	
fill	
fill_n	
reverse	
rotate	
unique	
transform	funkcje na elementach kolekcji
generate	
swap	dwa elementy

<algorithm>

generyczne algorytmy podstawowe

Łącznie około 120 algorytmów

is_sorted (C++11)	Sortowanie
sort	
partial_sort	
lower_bound	Przeszukiwanie binarne (na posortowanych danych)
upper_bound	
binary_search	
equal_range	

Biblioteka standardowa (cd4)

max	min/max
max_element	
min	
min_element	
minmax (C++11)	
minmax_element (C++11)	
lexicographical_compare	porównanie leksykograficzne
is_permutation (C++11)	permutacje
next_permutation	
qsort	funkcje z <cstdlib>
bsearch	

merge	Operacje na zbiorach (posortowanych)
includes	
set_difference	
set_intersection	
is_heap (C++11)	Operacje na stertach
is_heap_until (C++11)	
make_heap	
push_heap	
pop_heap	
sort_heap	
iota (C++11)	operacje numeryczne
accumulate	
adjacent_difference	
partial_sum	

Przykład


```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{ vector<int> v;  // Pusty
  for(int i=0; i<5; ++i)
    v.push_back(rand()%1000);
  v.push_back(INT_MIN);
  v.push_back(INT_MAX);

  for(vector<int>::const_iterator i = v.begin(); i != v.end(); ++i)
    cout<<*i<<' ';
  cout<<endl;
  sort(v.begin(), v.end());
  for(auto i = v.begin(); i!=v.end(); ++i)
    cout<<*i<<' ';
  cout<<endl;
system("pause");
return 0;
}
```

kontener

„inteligentny wskaźnik”

algorytm

dedukcja typu

wyłuskanie

```
41 467 334 500 169 -2147483648 2147483647
-2147483648 41 169 334 467 500 2147483647
Press any key to continue . . .
```

for - implementacja


```
#include
#include
#include
using na

int mair
{
    vector<int> v;  // Pusty
    int tab[5] = {1,2,3,4,5};

    for (const auto& i : v) cout << i << ' ';
    cout << endl;

    for (const auto& i : tab) cout << i << ' ';
    cout << endl;
  return 0;
}
```

```
std::vector<int, std::allocator<int> >& __range1 = v;
__normal_iterator< … > __begin1 = __range1.begin();
__normal_iterator< … > __end1 = __range1.end();
for (; !operator==(__begin1, __end1); __begin1.operator++()) {
    int i = __begin1.operator*();
    std::operator<<(std::cout.operator<<(i), ' ');
}
// https://cppinsights.io/
```


Iteratory

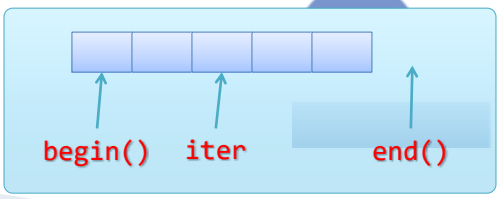
Iterator to obiekt, służący do uniwersalnego nawigowania w kontenerach

Obsługuje wybrane operatory, np:

`*` `->` `++` `--` `==` `!=` `=`

Podstawowe funkcje kontenerów związane z iteratorami, np.:

`begin( ), end( )`



Wyszukiwanie elementu o podanym kluczu


```
int find(std::vector<int> const & v, int key)
{
  for (size_t i = 0; i < v.size(); ++i) {
    if (v[i] == key)  return i;
  return -1;
  }
}
std::vector<int>::iterator findI(std::vector<int>&v, int key)
{
  for (std::vector<int>::iterator i=v.begin(); i!=v.end(); ++i) {
    if (*i == key) return i;
  return v.end();
  }
}
 auto findA(auto& v, auto key) // C++20
 {
     for (auto i = v.begin(); i != v.end(); ++i) {
         if (*i == key)  return i;
     }
     return v.end();
 }
```


Wyszukiwanie / optional c++17


```
class MyCollection {
public:
    MyCollection() : names{ "Aldona", "Andrzej", "Beata", "Jan", "Zofia" } {}
    std::optional<std::string> find_name(const std::string& name )    {
        for (auto& n : names) {
            if (n == name) return std::make_optional(n);
        }
        return std::optional<std::string>{};
    }
private:  std::vector<std::string> names;
};
int main() {
    MyCollection mc;
    const auto jan1 = mc.find_name("Jan1");
    const auto zosia = mc.find_name("Zofia");

    if (jan1 != std::nullopt) std::cout << jan1.value();
    else                      std::cout << "--pusty--";

    if (zosia.has_value())    std::cout << zosia.value();
    else                      std::cout << "--pusty--";
}
```

Szkielet kontenera z iteratorem


```
class Kontener {// zbiór/kolekcja Element'ów
  Data data;
  public:
    class uiterator {  // „uchwyt" do Elementu kolekcji
      private:
       uiterator() {...}; // inicjacja „uchwytu"
      public :
        uiterator operator++() const { return uiterator{...}; }
        bool operator!=(const uiterator i) const {...}
        Element& operator*() {...}
     };

     Kontener() {}
     uiterator begin() { return uiterator {...}; }
     uiterator end()   { return uiterator {...}; }
};
```