Uniwersytet Wrocławski

C++: *kontenery*

Zbigniew Koza
Wydział Fizyki i Astronomii

CO TO SĄ KONTENERY?

Kontener C++ jest to...

- Obiekt służący do przechowywania innych obiektów
- Kontenery pojawiły się w **bibliotece standardowej** C++98
  i były główną atrakcją tamtej wersji języka
- Bez kontenerów nie można efektywnie programować w C++

Kontenery...

- Implementują różne podstawowe **struktury danych**, np.:
  - kolejki LIFO (last in first out)
  - kolejki FIFO (first in first out)
  - listy
  - drzewa czerwono-czarne
  - tablice mieszające
  - tablice dynamiczne
- Nie wymyślaj koła!

Kontenery...

- Są w C++ zaimplementowane jako szablony klas (szczegóły później), co umożliwia im przechowywanie niemal każdego rodzaju danych – **elastyczność**
- Posiadają wspólne oraz unikatowe elementy interfejsu. Istnienie elementów wspólnych umożliwia podmianę jednych kontenerów innymi, w zależności od potrzeb i możliwości

Kontenery...

- Charakteryzują się zbiorem operacji, jakie można wykonywać na przechowywanych w nich danych
- Implementacja kontenerów musi gwarantować określoną wydajność każdej operacji na kontenerach
- Różne kontenery specjalizują się w efektywnym wykonywaniu różnych operacji – mają więc zastosowania w innych sytuacjach

Kontenery...

- Są elementem biblioteki standardowej ⇒ przenośny, bezpieczny kod

Kontenery

- Z reguły dane przechowują na stercie, całkowicie automatyzując zarządzanie tą pamięcią

Dwie rodziny kontenerów

Sequence containers

- Organizują dane w **ciągi**
- Są **uporządkowane** (od pierwszego do ostatniego elementu)

Associative containers

- Organizują dane w postaci **ciągów (klucz, wartość)**
- W tym sensie reprezentują **odwzorowania** typu **klucz ↦ wartość**
- Również są **uporządkowane** wg. jakiegoś schematu

SEQUENCE CONTAINERS

Kontenery sekwencyjne

- **array** [C++11] tablica statyczna
- **vector** tablica dynamiczna
- **deque** kolejka **o dwóch końcach**
- **forward_list** [C++11] lista jednokierunkowa
- **stack** stos
- **queue** kolejka
- **priority_queue** kolejka priorytetowa

array<T, N>

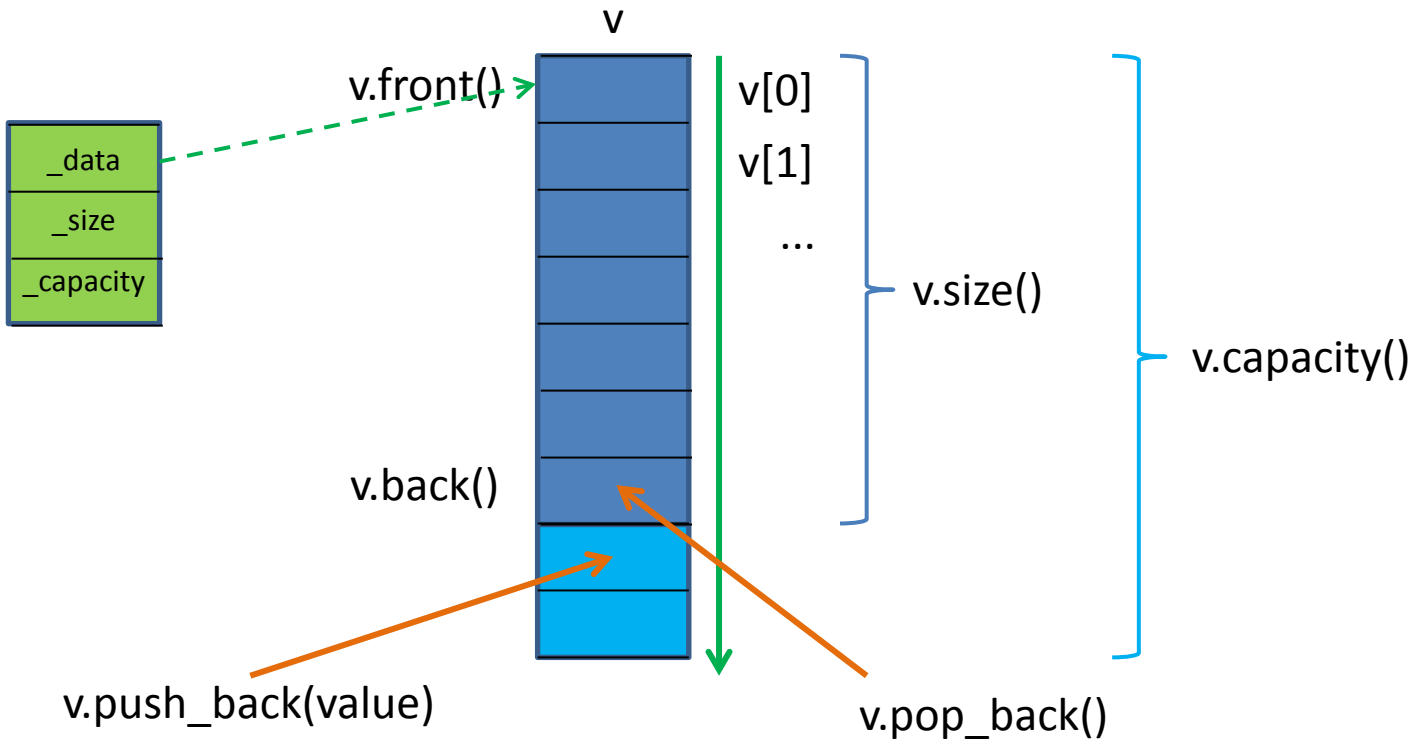
- Jest to abstrakcja zwykłej tablicy:
  `std::array<int,10> tab; ⇔ int tab[10];`
- Rozmiar jest stały
- Przykład:

```
std::array<int, 3> a = {1, 2, 3};
std::sort(a.begin(), a.end());
std::cout << a[0];
a.back() = 90;
```

- Szczegóły: http://www.cplusplus.com/reference/array/array/

vector<T>

- Dynamiczna tablica (zmienny rozmiar)
- Zajmuje ciągły obszar pamięci (na stercie)



vector<T>

- Najważniejsze funkcje składowe:
  - `operator[]`
  - `size`
  - `push_back(value)`
  - `pop_back()`
  - `empty`
  - `resize(n)`
  - `reserve(n)`
  - `clear`
  - `swap`
  - `shrink_to_fit` [c++11]

vector<T>

- Bardzo efektywne zarządzanie pamięcią
- Bardzo szybki dostęp do dowolnego elementu
- Możliwość dynamicznej zmiany rozmiaru
- Powolne:
  - łączenie wektorów
  - usuwanie elementów (z wyjątkiem ostatniego)
  - wstawianie elementów (z wyjątkiem na końcu)
- Zmiana rozmiaru czasami jest kosztowna

deque<T>

- Kolejka o dwóch końcach
- Szybkie operacje na początku i końcu kolejki
- Dane nie zajmują ciągłego obszaru pamięci
- Bardzo szybki dostęp do n-tego elementu
- Wstawianie i usuwanie elementów poza początkiem i końcem – kosztowne, ale nie tak, jak w std::vector<T>

deque<T>

```
#include <iostream>
#include <deque>

int main()
{
  std::deque<int> d = {17, 2, 5, 4, 6};
  d.push_front(32);
  d.push_back(21);
  for(auto n : d) // specjalna pętla for
  {
    std::cout << n << "\n";
  }
}
```

Dygresja: kontenerowa pętla for

```
Container<T> container;
...
for (auto x : container)// pętla do odczytu
{
  do_something(x);
}

for (auto & x : container)// referencja!
{
  do_something_to (x);
}
```

Dygresja: kontenerowa pętla for

C++11

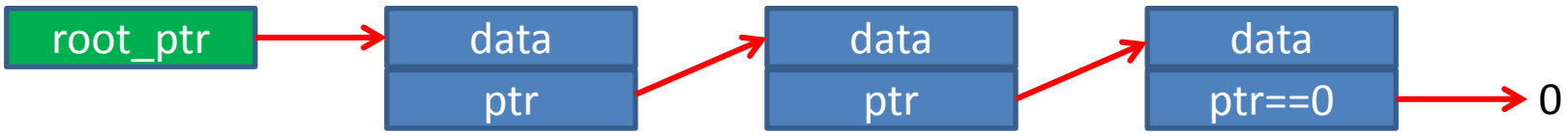
```
std::list<int> v;
...
for (auto x : v)
{
    suma += x;
}
```

C++98

```
std::list<int> v;
...
for (std::list<int>::const_iterator it =  v.begin();
     it != v.end();
     ++it)
{
    suma +=  *it;
}
```

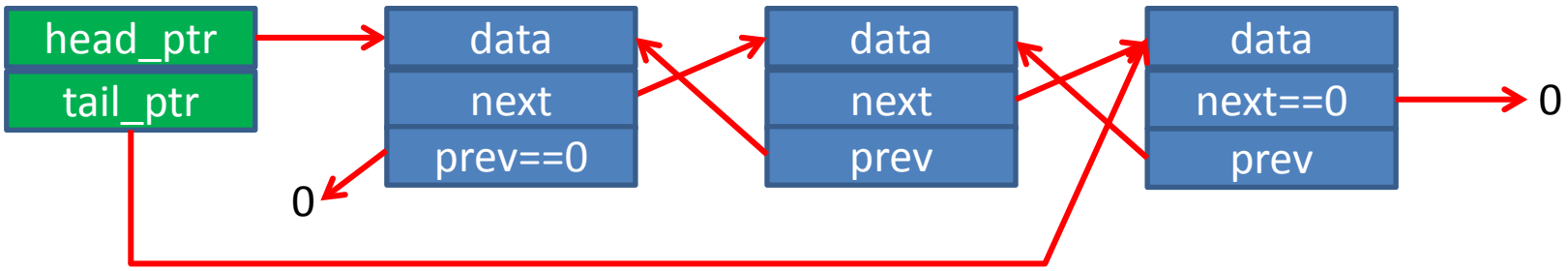
forward_list<T>

- Gwarantuje bardzo szybkie wstawianie i usuwanie elementów w dowolnym miejscu
- Gwarantuje bardzo szybkie łączenie list
- Nie ma operatora[]
- Implementacja: lista pojedynczo wiązana



list<T>

- Gwarantuje bardzo szybkie wstawianie i usuwanie elementów w dowolnym miejscu
- Gwarantuje bardzo szybkie łączenie list
- Nie ma operatora[]
- Implementacja: lista podwójnie wiązana



stack<T>

- Zapewnia funkcjonalność stosu (LIFO)
- Podstawowe funkcje:
  - push_back
  - pop_back
  - back

queue<T>

- Zapewnia funkcjonalność kolejki (FIFO)
- Podstawowe funkcje:
  - push_back
  - pop_front
  - back
  - front

priority_queue<T>

- Zapewnia funkcjonalność kopca (stogu)
- Bardzo szybki dostęp do największego elementu
- Wstawianie i usuwanie: czas logarytmiczny
- Podstawowe funkcje: jak w queue<T>
- składowa top() zwraca największy element

ASSOCIATIVE CONTAINERS

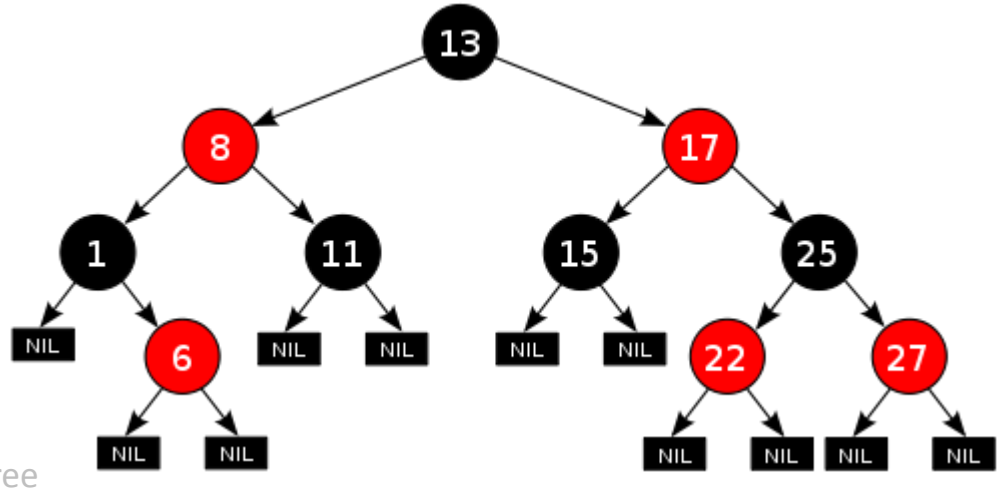
Kontenery sekwencyjne

- **set** zbiór unikatowych kluczy
- **multiset** zbiór kluczy
- **map** zbiór {unikatowy klucz->**wartość**}
- **multimap** zbiór {klucz->**wartość**}

  Jak wyżej, ale inne implementacje (i właściwości)
- **unordered_set**
- **unordered_multiset**
- **unordered_map**
- **unordered_multimap**

set<Key>

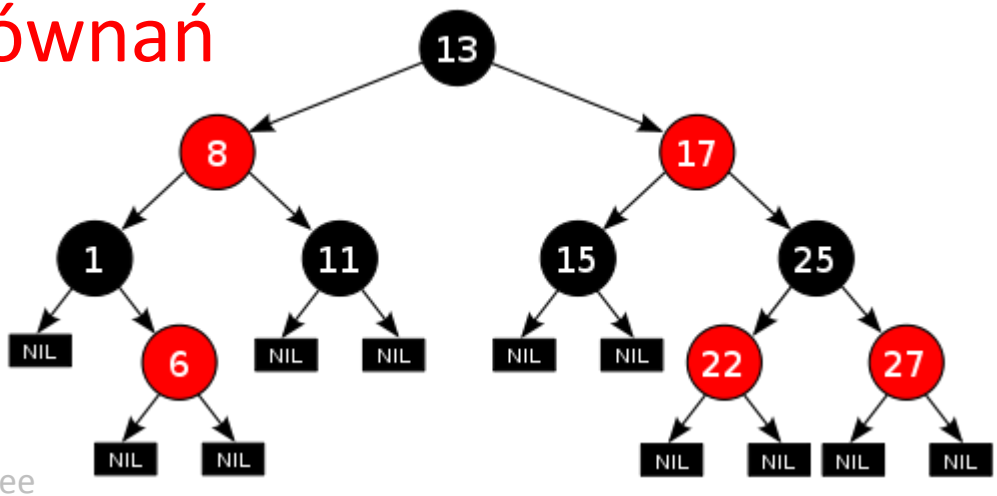
- Zbiór kluczy (= obiektów typu Key)
- Klucze są uporządkowane
- Typowa implementacja: drzewo (binarne) czerwono-czarne
- Wyszukiwanie, usuwanie, wstawianie: **czas logarytmiczny**



https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

set<Key>

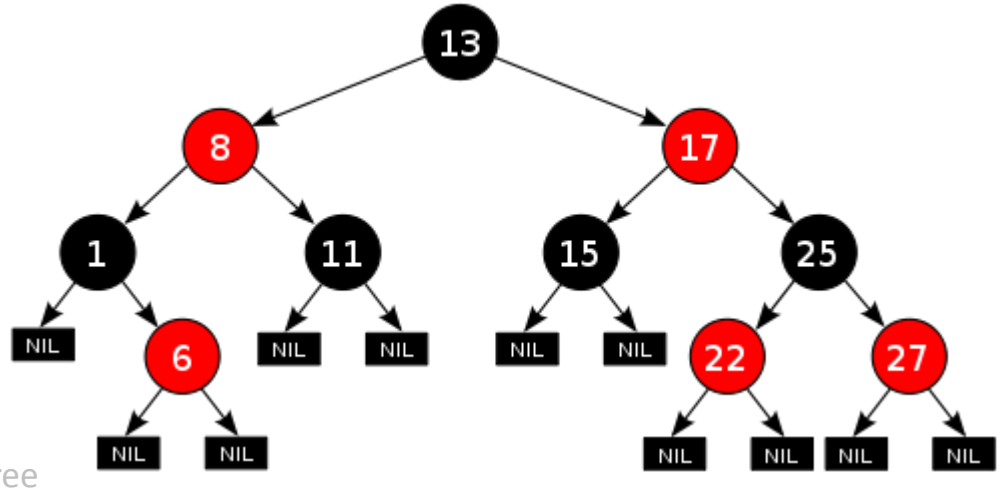
- Wyszukiwanie, usuwanie, wstawianie:
  czas logarytmiczny
- Przykład:
  Zbiór zawiera milion słów: jak szybko można sprawdzić, czy dane słowo jest w zbiorze?
- $\log_2 10^6 \approx 20$ porównań



https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

map<Key, T>

- Zbiór par {klucz, wartość}
- Pary są uporządkowane względem kluczy
- Typowa implementacja: drzewo (binarne) czerwono-czarne
- Wyszukiwanie, usuwanie, wstawianie: **czas logarytmiczny**



https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

map<Key, T>

- Podstawowe funkcje:
  - operator[]
  - find
  - insert
- Przykład:

```
std::map<std::string, int> mapa;
mapa["Ala"] = 9;
mapa["Ola"]++; // 1
std::cout << mapa["Ala"]; // 9
```

https://en.wikipedia.org/wiki/Red%E2%80%93black_tree