Jeszcze więcej kontenerów

#set #map
#key #value

W tej części podręcznika warto omówić jeszcze dwa użyteczne kontenery: `set` oraz `map`. Są one zupełnie inaczej zaimplementowane niż na przykład `vector` czy `deque`, ponieważ przechowywane dane są w nich w pewien sposób posortowane i dodawanie lub usuwanie elementów nie zaburza tego porządku. Z tym wiąże się pewien oczywisty fakt, że te kontenery będą miały duży narzut czasowy i pamięciowy, to znaczy będą zajmować znacznie więcej miejsca, niż same dane, z których będziemy korzystać i operacje na nich będą bardziej czasochłonne. Jednak ich funkcjonalność czasem się przydaje i wtedy warto ponieść dodatkowy koszt.

Kontener `set`

Angielskie słowo **set** oznacza „zbiór” (w sensie mnogościowym) i działanie tego kontenera rzeczywiście odpowiada takiemu matematycznemu zbiorowi. Podstawową jego cechą jest fakt, że nie może w nim być powtarzających się elementów.* Jeśli próbowalibyśmy ponownie dodać istniejący element, po prostu nie zostanie on dodany. Sprawdźmy jak to działa (elementy dodaje się przy użyciu funkcji `insert()`):

```
set<int> Q;
Q.insert(1);
Q.insert(2);
Q.insert(3);
cout << Q.size() << endl;  // pojawi się 3
Q.insert(1);
cout << Q.size() << endl;  // znów pojawi się 3, element nie został dodany
```

Jeśli chcemy sprawdzić, czy jakaś wartość występuje w zbiorze, wtedy używamy funkcji `count()` – zwraca ona ilość wystąpień szukanego elementu w zbiorze, a więc liczbę 0 lub 1.† Oto przykład użycia funkcji `count()`:

```
cout << Q.count(2) << endl;  // pojawi się 1
cout << Q.count(4) << endl;  // pojawi się 0
```

*Istnieje komplementarny kontener, w którym elementy mogą się powtarzać, jest to `multiset`.

†Można też użyć funkcji `find()`, wtedy zwróci ona iterator wskazujący na ten element. Jeśli elementu nie ma, wtedy `find()` zwróci wartość `end()`.

Elementy możemy usuwać przy pomocy funkcji `erase()` podając jej wartość do usunięcia:[‡]

```
Q.erase(2);  // element 2 został usunięty
cout << Q.size() << endl;  // pojawi się 2
```

Można wypisać wszystkie elementy zbioru przy pomocy podobnej pętli do tej, którą przedstawialiśmy w podrozdziale o iteratorach:

```
set<int>::iterator p;
for(p = Q.begin(); p != Q.end(); p++)
  cout << *p << endl;
```

Elementy zbioru pojawią się w kolejności od najmniejszego do największego.

Inne przydatne funkcje to między innymi `empty()` sprawdzająca, czy zbiór jest pusty oraz `clear()` – usuwająca wszystkie elementy zbioru:

```
cout << Q.empty() << endl;  // pojawi się 0 (fałsz)
Q.clear();
cout << Q.empty() << endl;  // pojawi się 1 (prawda)
```

Kontener `map`

Bardzo przydatny kontener `map` (mapa) to po prostu zbiór par (kontenerów `pair`). Pierwsza składowa pary nazywana jest *kluczem* (ang. **key**, wymawiaj: *kii*), zaś druga nazywana jest *wartością* (ang. **value**, wymawiaj: *waliu*).

W przypadku mapy ciekawe jest to, że klucz może być dowolnego typu, zatem przypomina to trochę tablicę indeksowaną dowolnymi obiektami. Dla przykładu weźmy mapę z kluczem typu `string` oraz wartością typu `int`:

```
map<string, int> M;
```

Elementy dodajemy do mapy po prostu nadając im wartość, na przykład:

```
M["alpha"] = 1;
M["beta"] = 5;
M["gamma"] = 10;
M["delta"] = 100;
```

Wartość elementu możemy zmieniać, na przykład:

```
M["alpha"] *= 2;
M["delta"]++;
```

[‡]Albo iterator wskazujący element do usunięcia, a nawet cały zakres odkąd-dokąd.

Jeśli zastosujemy operator inkrementacji (lub dekrementacji) do elementu o nieistniejącym kluczu, wtedy zostanie utworzony nowy element (tutaj o wartości 0), po czym zostanie zastosowany ten operator, na przykład:

```
M["epsilon"]--;  // nowy element ma wartość -1
```

Wszystkie elementy mapy możemy wypisać używając iteratora, ale musimy pamiętać, że są to pary, a nie wielkości skalarne:

```
map<string, int>::iterator p;
for(p = M.begin(); p != W.end(); p++)
  cout << (*p).first << ' ' << (*p).second << endl;
```

Na ekranie pojawią się pozycje posortowane względem klucza:

```
alpha 2
beta 5
delta 101
epsilon -1
gamma 10
```

Jedna drobna uwaga: zwykle zamiast `(*p).first`, `(*p).second` pisze się `p->first`, `p->second`, co bardziej podkreśla wskaźnikowy charakter zmiennej p.

Pozostałe funkcje, o których wspomniano w sekcji o kontenerze `set`, działają analogicznie: `count()` zlicza ilość elementów o danym kluczu (czyli 0 lub 1, bo klucze nie mogą się powtarzać), `erase()` usuwa element o danym kluczu, a funkcje `empty()` oraz `clear()` działają tak samo. Tylko funkcja `insert()` jest troszkę inna, ale na razie nie będzie nam potrzebna.