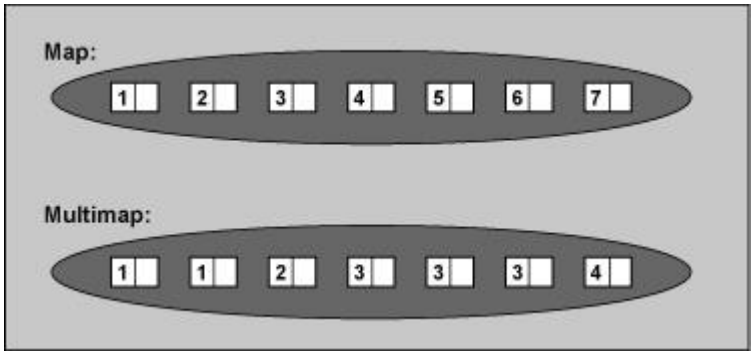
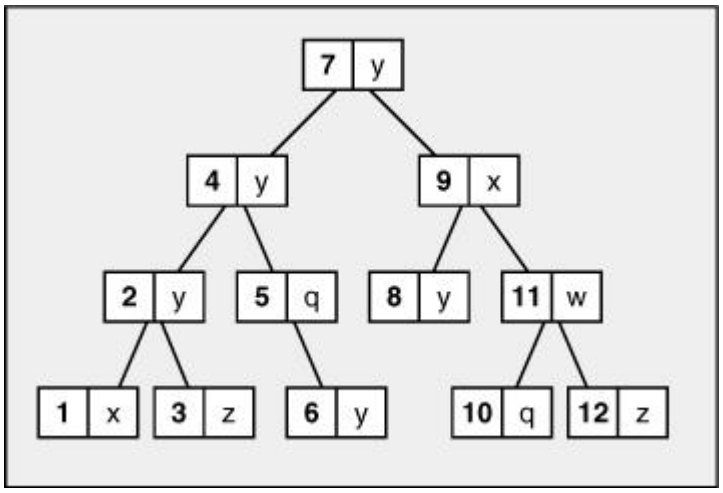
Asociativní kontejnery

map, multimap

Kontejner *map* ukládá prvky jako dvojici klíč + hodnota. Prvky jsou uloženy uspořádané podle klíče. Kontejner *multimap* může navíc obsahovat více prvků se stejným klíčem. Uspořádání klíčů v kontejnerech je dle operátoru **<** nebo podle predikátu ***mensi***, je-li predikát ***mensi*** uveden.



Prvky jsou uloženy ve vyváženém vyhledávacím stromu. Vyhledávání dle klíče je rychlé, vyhledávání dle hodnoty prvku je pomalé.



Některé konstruktory

map **m**	Vytvoří prázdnou mapu nebo multimapu ***m***.
map **m1(m2)**	Vytvoří mapu nebo multimapu ***m1***, která je kopií ***m2***. Prvky v ***m2*** musí mít stejný typ.

kde část ***map*** může mít tvar:

map<typ_klice,typ_hodnoty>	Vytvoří mapu. Uspořádání je dle operátoru **<** .
map<typ_klice,typ_hodnoty,mensi>	Vytvoří mapu. Uspořádání je dle kriteria ***mensi***.
multimap<typ_klice,typ_hodnoty>	Vytvoří multimapu. Uspořádání je dle operátoru **<** .
multimap<typ_klice,typ_hodnoty,mensi>	Vytvoří multimapu. Uspořádání je dle kriteria ***mensi***.

Operace neměnící obsah mapy nebo multimapy

`size()`	Počet uložených prvků.
`empty()`	Zda je mapa prázdná, tj. zda platí **`size()==0`** .
`max_size()`	Maximální počet prvků, který lze do mapy nebo multimapy uložit.

Operace hledání pro mapy a multimapy

`count(klic)`	Vrací počet prvků, které mají hodnotu klíče ***klic***. U mapy to může být 0 nebo 1 (mapa nemůže obsahovat více prvků se stejnou hodnotou klíče).
`find(klic)`	Vrací pozici prvního prvku, který má hodnotu klíče ***klic***. Není-li nalezen, vrací pozici **`end()`**.
`lower_bound(klic)`	Vrací první pozici, na kterou by byl prvek s klíčem ***klic*** uložen (pozice prvního prvku, jehož klíč **>= *klic***).
`upper_bound(klic)`	Vrací poslední pozici, na kterou by byl prvek s klíčem ***klic*** uložen (pozice prvního prvku, jehož klíč **> *klic***).
`equal_range(klic)`	Vrátí první a poslední z pozic, na kterých jsou uloženy prvky s klíčem ***klic*** (u mapy to může být jen jedna pozice).

Operace přiřazení

`m1 = m2`	Do ***m1*** zkopíruje obsah ***m2*** **.**

Iterátory

`begin()`	Vrací ukazatel na první prvek (pro nastavení iterátoru na začátek).
`end()`	Vrací ukazatel ukazující za poslední prvek (pro zjištění, zda iterátor už není mimo mapu).
`rbegin()`	Vrací ukazatel na poslední prvek (pro nastavení reverzního iterátor na začátek průchodu).
`rend()`	Vrací ukazatel ukazující před první prvek (pro zjištění, zda reverzní iterátor není už mimo mapu).
`cbegin()`	Vrací ukazatel na první prvek (pro nastavení konstantního iterátoru na začátek).
`cend()`	Vrací ukazatel ukazující za poslední prvek (pro zjištění, zda konstantní iterátor už není mimo mapu).
`crbegin()`	Vrací ukazatel na poslední prvek (pro nastavení reverzního konstantního iterátoru na začátek průchodu).
`crend()`	Vrací ukazatel ukazující před první prvek (pro zjištění, zda reverzní konstantní iterátor není už mimo mapu).

Vložení a odebrání prvku

`insert(prvek)`	Vloží kopii ***prvku*** a vrátí jeho pozici. Vrací dvojici hodnot.
`insert(poz,prvek)`	Vloží kopii ***prvku*** a vrátí jeho pozici. Pokud pozice ***poz*** pozicí prvku, který bude za vloženým prvkem, vložení se tím zrychlí.
`insert(zac,kon)`	Vloží kopii prvků od pozice ***zac*** po pozici ***kon***. Nevrací žádnou hodnotu.
`emplace(údaje)`	Sestaví prvek z údajů klíče a hodnoty a umístí ho. Vrací dvojici (*iterátor* + *bool*).
`emplace_hint(poz,údaje)`	Sestaví prvek z klíče a hodnoty, umístí ho a vrátí jeho pozici (v případě, kdy vložení bylo úspěšné). Hledání pozice vložení začíná do zadané pozice ***poz***. Je-li tato blízko cílové pozice, zrychlí se tím vložení prvku. Vrací skutečnou pozici vložení.
`erase(klic)`	Odstraní všechny prvky s hodnotou klíče ***klic***. Vrací počet odstraněných prvků.
`erase(poz)`	Odstraní prvek na pozici ***poz***. Nevrací žádnou hodnotu.
`erase(zac,kon)`	Odstraní prvky od pozice ***zac*** po pozici ***kon***. Nevrací žádnou hodnotu.
`clear()`	Odstraní všechny prvky. Mapa nebo multimapa bude nyní prázdná. Nevrací žádnou hodnotu.

Prvek je zde strukturovaný typ **`pair`** se dvěma členy – klíčem a hodnotou. K vytvoření páru lze použít:

- funkci **`make_pair`**
- šablonu **`pair<typ_klíče,typ_hodnoty>`**

Je-li návratová hodnota dvojice hodnot, je to strukturovaný typ **`pair`** se dvěma členy **`first`** a **`second`**.

- Člen **`second`** udává, zda operace byla úspěšná (zda došlo ke vložení).
- Člen **`first`** vrací pozici, na kterou prvek byl vložen, pokud došlo ke vložení, nebo pozici, na které je již prvek se stejnou hodnotou (u mapy).

Je-li návratová hodnota operace pozice vložení, je to pozice, kam byl prvek vložen, pokud došlo ke vložení, nebo kde je již existující prvek se stejnou hodnotou (u mapy).

Příklad.

```
#include <map>
struct Zlomek { unsigned cit,jmen;
  Zlomek(unsigned c, unsigned j) { cit=c,jmen=j; }
```

```
};
multimap<double,Zlomek> m;
m.insert(make_pair(5./3,Zlomek(5,3)));
m.insert(pair<double,Zlomek>(1./2,Zlomek(1,2)));
m.emplace(2./7,Zlomek(2,7));
m.emplace(3./4,Zlomek(3,4));
for (auto z:m) cout << z.first << "  "
               << z.second.cit << '/' << z.second.jmen << endl;
```

```
0.285714  2/ 7
0.5  1/2
0.75  3/4
1.66667  5/3
```

Přístup k prvkům mapy

`m[klic]`	Vrací referenci na hodnotu s klíčem ***klic***. Pokud hodnota s tímto klíčem neexistuje, vloží prvek s tímto klíčem.
`at(klic)`	Vrací referenci na hodnotu s klíčem ***klic***. Na rozdíl od předchozího operátoru [] hodnota s tímto klíčem musí existovat. Pokud neexistuje, generuje výjimku `out_of_range`.

Operátor [] lze efektivně použít jen v případech, kdy máme zajištěno, že nepoužijeme neexistující hodnotu klíče, pokud tímto způsobem nechceme vložit nový prvek. Navíc zde musí být zajištěno, že ukládané hodnoty mají implicitní konstruktor (základní číselné datové typy tento konstruktor mají, je jim přiřazena hodnota 0).

Příklad.

```
struct Zlomek { unsigned cit,jmen;
                Zlomek(unsigned c,unsigned j) { cit=c,jmen=j; }
                Zlomek() { cit=0,jmen=1; }
};
map<double,Zlomek> m;
m[0.75] = Zlomek(3,4);
auto p=m.lower_bound(0.75);
cout << p->first << "  "
     << p->second.cit << '/' << p->second.jmen << endl;
```

```
0.75  3/4
```

Příklad.

```
  map<string,float> m;
  m["jablka"]=19.9;
  m["ananas"]=29.9;
```

```
m["mandarinky"]=21.9;
cout << "ananas: " << m["ananas"];
```

nebo: `cout << "ananas: " << m.at("ananas");`

ananas 29.9

```
try { cout << "broskve: " << m.at("broskve"); }
catch (out_of_range) { cout << "cena neni uvedena"; }
```

cena neni uvedena

Příklad.

```
multimap<string,string> m;
m.emplace("peach","broskev");
m.emplace("toy","zabava");
m.emplace("key","klic");
m.emplace("bicycle","kolo");
m.emplace("toy","hracka");
m.emplace("key","klavesa");
m.emplace("toy","rozmar");
for (auto p=m.equal_range("toy"); p.first!=p.second; ++p.first)
      cout << p.first->first << "  " << p.first->second << endl;
```

toy zabava
toy hracka
toy rozmar

nebo explicitní deklarace datového typu dvojice iterátorů:

```
typedef multimap<string,string> M;
for (pair<M::iterator,M::iterator> p=m.equal_range("toy");
                                   p.first!=p.second;
                                   ++p.first) ...
```
