**Sveučilište u Zagrebu**
**PMF – Matematički odsjek**


# Objektno programiranje (C++)

Predavanja 04 – C++11

**Vinko Petričević**

# auto

- C++11 uvodi ključnu riječ auto, pomoću koje možemo kreirati varijablu koja ima tip kao rezultat nekog izraza

```
int x = 1;    double y = 1.0;
auto z = x + y;  // prevodilac će učiniti z tipa double

template <typename T1, typename T2> void f(T1 t1, T2 t2) {
 auto t = t1 + t2; // biti će tip kojeg je povratni tip od operator+(t1, t2)
    // ...
}
```

Ako želimo samo kreirati varijablu nekog tipa možemo koristiti/ili kreirati funkciju koja vraća neki tip

```
decltype(f()) sum = 0.0; // sum ima tip koji vraća f()

template <typename T1, typename T2>
auto g(T1 const & t1, T2 const & t2) -> decltype(t1*t2)
{  return t1*t2; }
```

# range-for

- C++11 uvodi novi način prolazaka spremnika

```
char name[] = { 'a','b','c','d','e'};
for(char x : name) std::cout << x << ",";

for(auto x : name) std::cout << x << ",";
for(auto& x : name)  x += 1;

int table[][3] = { {1,2,3}, {4,5,6}, {7,8,9} };

for(auto& row : table){
   for(auto col : row) std::cout << col << ",";
       std::cout << std::endl;
```

# Neuređeni spremnici

- C++11 uvodi neuređene asocijativne spremnike unordered_set, unordered_map, unordered_multiset i unordered_multimap koji su zasnovani na hash-tablici

```
template <typename T,
    typename Hash = hash<T>, typename EqPred = equal_to<T>,
    typename Allocator = allocator<T> >
class unordered_set;

template <typename Key, typename T,
    typename Hash = hash<Key>, typename EqPred = equal_to<Key>,
    typename Allocator = allocator<pair<const Key, T> > >
class unordered_map;
```

- #include <unordered_set> odnosno <unordered_map>
- Oni koriste funkciju hash umjesto operator<, pa nemaju upper_bound ni lower_bound, ali imaju equal_range
- Pristup elementima u različitim pretincima je konstantan, ali je unutar jednog linearan
- Ako je dobro napravljen hash, operator[] je brži, ali je prolazak kroz cijeli spremnik sa ++iterator sporiji nego kod uređenih spremnika

# Neuređeni spremnici – posebne funkcije

| | |
|---|---|
| c.hash_function() | Vraća *hash* funkciju |
| c.key_eq() | Vraća predikat ekvivalencije |
| c.bucket_count() | Vraća trenutni broj pretinaca |
| c.max_bucket_count() | Vraća maksimalni mogući broj pretinaca |
| c.load_factor() | Vraća trenutno opterećenje |
| c.max_load_factor() | Vraća trenutno maksimalno opterećenje |
| c.max_load_factor(val) | Postavlja maksimalno opterećenje na val |
| c.rehash(bnum) | Vrši rehaširanje spremnika tako da broj pretinaca bude barem bnum |
| c.reserve(num) | Vrši rehaširanje spremnika tako da ima prostora za barem num elementa |

# Neuređeni spremnici – analiza

- Za analizu strukture spremnika imamo i sljedeće specijalne funkcije (pogledati primjere)

| | |
|---|---|
| c.bucket(val) | Vraća indeks pretinca u koji bi vrijednost val bila spremljena |
| c.bucket_size(buckidx) | Vraća broj elemenata u pretincu s indeksom buckidx |
| c.begin(buckidx) | Vraća jednosmjerni iterator koji pokazuje na prvi element u pretincu s indeksom buckidx |
| c.end(buckidx) | Vraća jednosmjerni iterator koji pokazuje iza zadnjeg elementa u pretincu s indeksom buckidx |
| c.cbegin(buckidx) | Vraća konstantan jednosmjerni iterator koji pokazuje na prvi element u pretincu s indeksom buckidx |
| c.cend(buckidx) | Vraća konstantan jednosmjerni iterator koji pokazuje iza zadnjeg elementa u pretincu s indeksom buckidx |

# Neuređeni spremnici

- Na većini standardnih tipova je napravljeno hashiranje
- ukoliko bismo željeli neku svoju klasu koristit kao ključ (slično kao i što smo kod uređenih spremnika trebali definirati <), ovdje trebamo napraviti operator== i strukturu/funkciju std::hash<t>)

```
namespace std {
    template<> struct hash<razlomak> {
        std::size_t operator()(const razlomak &r) const {
            return std::hash<long>{}(*(const long*)(&r));
        }
    };
}
```

- Nakon ovoga će raditi npr. unordered_set<razlomak>, kojeg smo napravili na vježbama 02

- Ili možemo pisati zasebne funkcije

```
size_t hasher(const std::string &sd){return std::hash<std::string>()(sd);}
bool eqOp(const std::string &lhs, const std::string &rhs)
{return lhs==rhs;}
...
std::unordered_map<std::string,int, decltype(hasher)*, decltype(eqOp)*> m;
```