

# Predlošci i generičko programiranje

## Objektno programiranje - 8. vježbe (2. dio)

Sebastijan Horvat

Prirodoslovno-matematički fakultet,  
Sveučilište u Zagrebu

10. svibnja 2023. godine



## Predložak (osnova za generičko programiranje)

- ▶ nacrt za stvaranje klase ili funkcije
- ▶ `vector` je primjer generičkog tipa
  - ▶ `vector<int>` ⇒ dali potrebnu informaciju kako bi se pri kompajliranju transformirao u konkretan tip

---

**Primjer.** Želimo funkciju za zbrajanje elemenata vektora:

```
vector<int> v1{1,2,3,4};  
vector<double> v2{2.1,4.5,-3.4};  
cout << zbroj(v1) << endl;  
cout << zbroj(v2) << endl;
```



## Prvo rješenje: Preopterećena funkcija zbroj

```
int zbroj(const vector<int> &v) {
    int rez = 0;
    for(auto &br : v)
        rez += br;
    return rez;
}

double zbroj(const vector<double> &v) {
    double rez = 0;
    for(auto &br : v)
        rez += br;
    return rez;
}
```

- ▶ uočite koliko koda smo morali ponoviti
- ▶ tako bi morali imati **po jednu funkciju za svaki novi tip!**
- ▶ Koja se od funkcija poziva (i gdje) na prethodnom slajdu?



## Drugo rješenje: Predložak funkcije

- ▶ formula iz koje se stvara funkcija za određeni tip

```
template <typename T> //lista parametara predloška
T zbroj(const vector<T> &v) {
    T rez = 0;
    for(auto &br : v)
        rez += br;
    return rez;
}
```

- ▶ koji stvarni tip predstavlja T određuje se pri kompajliranju na temelju toga kako koristimo tu funkciju

```
vector<int> v1{1,2,3,4};
vector<double> v2{2.1,4.5,-3.4};
cout << zbroj(v1) << endl; //T = int
cout << zbroj(v2) << endl; //T = double
```

- ↳ kompajliranjem generirane dvije funkcije (instance predloška)



## Više parametara predloška

**Primjer.** Funkciju koja vraća manji od dva podatka tipa T:

```
template <typename T>
T manji(const T &a, const T &b) {
    return (a < b) ? a : b;
}
```

ne možemo koristiti ovako:

```
cout << manji(1, 2.0) << endl;
```

- poruka kompajlera:

*error: no matching function for call to 'manji(int, double)'*



## Više parametara predloška (nastavak)

```
template <typename T, typename U>
[?] manji(const T &a, const U &b) {
    return (a < b) ? a : b;
}
```

Ispis koji dobivamo ako stavimo:	[?] = T	[?] = U
cout << manji(1.32, 5) << endl;	1.32	1
cout << manji(5, 1.32) << endl;	1	1.32

**Napomena.** Od standarda C++14 moguće je:

```
auto manji(auto a, auto b) {
    return (a < b) ? a : b;
}
```

**Napomena.** Umjesto ključne riječi `typename` nekad se piše `class`.



## Parametri predložka koji nisu za tipove

- ▶ predstavljaju **vrijednosti**, a **ne tip**
- ▶ te parametre kompajler **pri kompajliranju** zamjenjuje vrijednostima koje daje korisnik ili ih kompajler sam može odrediti ( $\Rightarrow$  vrijednost mora biti **konstantni izraz**)
- ▶ parametar mora biti **integralnog tipa**, ili **pokazivač ili (lijeva) referenca na objekt ili funkcijski tip** ([pravila](#))

### Primjer.

```
template<unsigned N, unsigned M>
int usp(const char (&p1)[N], const char (&p2)[M]) {
    return strcmp(p1, p2);
}
...
cout << usp("abc", "de") << endl;
```

(Ovdje:  $N = 4$ ,  $M = 3$ , ispis: -3)



## Još jedan primjer

- ▶ ako u prethodnom  $M$  stavimo kao `double`:

```
template<unsigned N, double M>
int usp(const char (&p1)[N], const char (&p2)[M]) {
    return strcmp(p1, p2);
}
```

dobivena poruka o grešci:

*error: 'double' is not a valid type for a template non-type parameter*



## Predložak klase

- ▶ nacrt za generiranje klase
- ▶ za razliku od predložaka funkcija, kompajler ne može odrediti tipove parametara predložka klase ⇒ moramo ih eksplicitno navesti

### Primjeri.

- ▶ `vector<int> a;`
- ▶ `shared_ptr<vector<int>> b;`

**Zadatak.** Napraviti predložak klase `vektor`.

### Primjer.

```
vektor<int> vi({1, 2, 3, 4, 5});
vi.izbaci();
vi.dodaj(6);
VektorPok<int> p;
for(p = vi.begin(); p != vi.end(); ++p)
    cout << *p << endl;
```



## Predložak klase `vektor`

- ▶ u def. predložka klase (i članovima) koristimo parametre predložka umjesto tipova ili vrijednosti koje predstavljaju

```
template <typename T> class vektor {
    friend class VektorPok;
public:
    typedef typename std::vector<T>::size_type
        size_type;
    vektor();
    vektor(std::initializer_list<T>);
    ...
    T& dohvati(size_type);
    void dodaj(const T &s) {
        ...
private:
    std::shared_ptr<std::vector<T>> podaci;
    ...
};
```



## Još malo „dodataka” predlošku klase

- ▶ ime za tip vrijednosti i *move* verzija funkcije dodaj

```
class vektor {  
    ...  
    public:  
        typedef T tip_vrijednosti;  
        ...  
        void dodaj(T &&s) {  
            podaci->push_back(std::move(s));  
        }  
        ...  
};
```



## Funkcije članice

- ▶ svaka instanca predložka klase ima vlastitu verziju svake funkcije članice
- ⇒ funkcija članica predložka klase ima iste parametre predložka kao i sama klasa
- ▶ definicija članice izvan klase mora imati i ime klase (s argumentima predložka!) kojoj pripada

```
template <typename T>  
void vektor<T>::provjeri(size_type i,  
                        const string &poruka) const {  
    if (i >= podaci->size())  
        throw out_of_range(poruka);  
}
```



## Funkcije članice (nastavak)

```
template <typename T>
```

```
T& vektor<T>::dohvati(size_type i) {  
    provjeri(i, "nema traženog elementa");  
    return (*podaci)[i];  
}
```

```
template <typename T>
```

```
void vektor<T>::izbaci() {  
    provjeri(0, "izbacivanje iz praznog vektora");  
    podaci->pop_back();  
}
```

**Napomena.** Ako se funkcija članica predložka klase ne koristi, tada se ona niti ne instancira.



## Konstruktori

```
template <typename T>
```

```
vektor<T>::vektor() :  
    podaci(make_shared<vector<T>>()) { }
```

```
template <typename T>
```

```
vektor<T>::vektor(initializer_list<T> il) :  
    podaci(make_shared<vector<T>>(il)) { }
```



## Jednostavnija upotreba imena predložka klase

- ▶ unutar doseg samog predložka klase ne trebamo navoditi argumente uz ime predložka (uočite što vraća `operator++`)

```
template <typename T>
```

```
class VektorPok {
```

```
    ...
```

```
    VektorPok(vektor<T> &v, size_t i = 0) :  
        wptr(v.podaci), tren(i) { }
```

```
    VektorPok& operator++();
```

```
    T& operator*() const;
```

```
    T* operator->() const;
```

```
private:
```

```
    std::shared_ptr<std::vector<T>> provjeri  
        (std::size_t, const std::string&) const;
```

```
    std::weak_ptr<std::vector<T>> wptr;
```

```
    ...
```

```
};
```



## Jednostavnija upotreba imena predložka klase

- ▶ pri definiciji članica izvan tijela predložka klase, u dosegu smo klase tek kad je viđeno ime klase (zato ovdje takav povratni tip!)

```
template <typename T>
```

```
VektorPok<T>& VektorPok<T>::operator++() {  
    provjeri(tren, "inkr. preko enda");  
    ++tren;  
    return *this;  
}
```





## Ostale funkcije članice predložka klase `VektorPok`

```
template <typename T>
```

```
shared_ptr<vector<T>> VektorPok<T>::provjeri  
    (size_t i, const string &poruka) const {  
    ...  
}
```

```
template <typename T>
```

```
T& VektorPok<T>::operator*() const {  
    ...  
}
```

```
template <typename T>
```

```
T* VektorPok<T>::operator->() const {  
    ...  
}
```



## Predložci klasa i njihovi frendovi

- ▶ predložak klase s ne-predložak frendom daje tom frendu pristup svim instancama predložka
- ▶ ako je frend također predložak, klasa kontrolira uključuje li to prijateljstvo sve ili samo određene instance tog predložka
- ▶ najčešće prijateljstvo: jedan-jedan  
(ispod: svaka instanca predložka `vektor` daje pristup verziji `VektorPok` s istim tipom `T`; uočite potrebnu deklaraciju)

```
template <typename> class VektorPok;
```

```
template <typename T> class vektor {  
    friend class VektorPok<T>;  
    ...  
    VektorPok<T> begin();  
    VektorPok<T> end();  
    ...  
};
```



## Predložci klasa i njihovi friendovi (nastavak)

```
template <typename T>
```

```
VektorPok<T> vektor<T>::begin() {  
    return VektorPok<T>(*this);  
}
```

```
template <typename T>
```

```
VektorPok<T> vektor<T>::end() {  
    return VektorPok<T>(*this, podaci->size());  
}
```



## Funkcije friendovi

```
template <typename T>
```

```
bool operator==(const VektorPok<T>&,  
                const VektorPok<T>&);
```

```
template <typename T>
```

```
bool operator!=(const VektorPok<T>&,  
                const VektorPok<T>&);
```

```
template <typename T>
```

```
class VektorPok {  
    friend bool operator==<T>(const VektorPok<T>&,  
                             const VektorPok<T>&);  
    friend bool operator!=<T>(const VektorPok<T>&,  
                              const VektorPok<T>&);  
    ...  
};
```

- ▶ uočite potrebne deklaracije; prijateljstvo ograničeno na operatore == i != instancirane istim tipom



## Funkcije frendovi (nastavak)

```
template <typename T>
bool operator==(const VektorPok<T> &lv,
                const VektorPok<T> &dv) {
    ...
}
```

```
template <typename T>
bool operator!=(const VektorPok<T> &lv,
                const VektorPok<T> &dv) {
    ...
}
```



## Primjer main funkcije

- ▶ za instanciranje s argumentom predloška, kompajler treba pristup implementaciji metoda  $\Rightarrow$  implementacija treba biti u .h datoteci - ili možemo ovako uz kompajliranje:

```
g++ -Wall main.cpp -std=c++11 -o prog
```

```
main.cpp
```

```
...
#include "vektor.cpp"
...
vektor<string> vs({"jedan", "dva", "tri"});
vektor<int> vi({1, 2, 3, 4, 5});
{
    vektor<string> vs1 = vs;
    vektor<int> vi1 = vi;
    vs1.izbaci();
    vi1.izbaci();
}
```



## Primjer main funkcije (nastavak)

```
    vs1.dodaj("pet");
    vi1.dodaj(6);
}
VektorPok<string> q;
for(q = vs.begin(); q != vs.end(); ++q)
    cout << *q << endl;
VektorPok<int> p;
for(p = vi.begin(); p != vi.end(); ++p)
    cout << *p << endl;
...
```



## Općenito i specifično prijateljstvo predložaka

### Primjer.

```
template <typename T> class Friend;

class C {
    friend class Friend<C>;
    template <typename T> friend class Friend2;
};

template <typename T> class C2 {
    friend class Friend<T>;
    template <typename X> friend class Friend2;
    friend class Friend3; //Friend3 nije predložak
};
```

- ▶ Friend instanciran klasom C je friend klase C (specifična instancijacija zahtjeva prethodnu deklaraciju od Friend)
- ▶ sve instance klase Friend2 su friendovi klase C (nije potrebna prethodna deklaracija od Friend2)



## Alijasi predložaka tipova

**Primjer.** Promotrimo sljedeći dio koda:

```
vektor<string> vs({"jedan", "dva", "tri"});  
vektor<int> vi({1, 2, 3, 4, 5});
```

Instanca predložka klase definira tip klase pa može ovako:

```
typedef vektor<string> svektor;  
typedef vektor<int> ivektor;  
svektor vs({"jedan", "dva", "tri"});  
ivektor vi({1, 2, 3, 4, 5});
```

Za predložak možemo ovako (predložak treba def. izvan funkcije!):

```
template<typename T> using v = vektor<T>;
```

```
int main() {  
    v<string> vs({"jedan", "dva", "tri"});  
    v<int> vi({1, 2, 3, 4, 5});  
    ...  
}
```

