



Generički mehanizam

Šabloni – I deo

Kako ubrzati razvoj softverskih rešenja?

- Kreirati što efikasnije kod koji nam rešava problem
 - ☐ Ne ponavljati kod
 - ☐ **Copy-paste** - izvor mnogih grešaka
 - ☐ Potprogrami
 - ☐ Biblioteke
- Kako kreirati kod koji ne rešava samo konkretan problem već klasu budućih (sličnih) problema?

Šta nam nudi C++ ?

- Rad sa funkcijama
- Preklapanje funkcija
- Klase - Nasleđivanje
- Virtuelne funkcije
- Polimorfizam - više oblika
- Ima li još nešto?

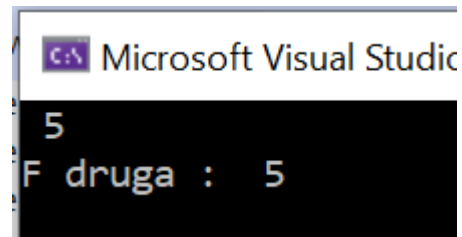
Funkcije i preklapanje funkcija

```
void F(int n) { printf("%d\n",n); }
```

```
class NumTip {  
public:  
    NumTip(int v = 0) : value(v) { }  
    int value;  
};
```

```
void F(NumTip n) { printf("F druga: %d\n",n.value); }
```

```
int main()  
{  
    NumTip n(5);  
    F(5);  
    F(n);  
    return 0;  
}
```



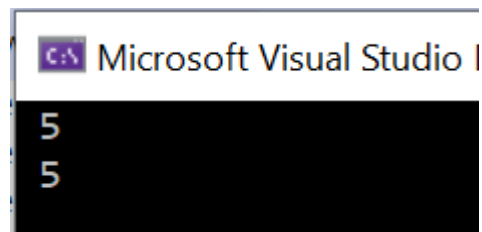
Operator konverzije i polimorfizam

```
void F(int n) { printf("%d\n",n); }
```

```
class NumTip {  
public:  
    NumTip(int v = 0) : value(v) { }  
    int value;  
    operator int()const { return value; }  
};
```

```
//void F(NumTip n) { printf("F druga: %d\n",n.value); }
```

```
int main()  
{  
    NumTip n(5);  
    F(5);  
    F(n);  
    return 0;  
}
```



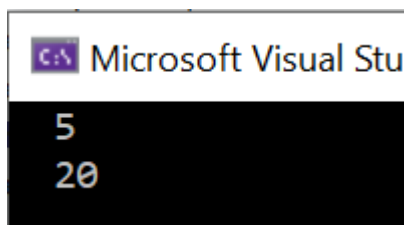
Dinamički polimorfizam

```
class Num {
    int value;
public:
    Num(int v = 0) : value(v) { }
    virtual int GetValue() const
    { return value; }
};

class NumIzvedena : public Num {
    int addValue = 10;
public:
    NumIzvedena(int v = 0) : Num(v) { }
    virtual int GetValue() const
    { return Num::GetValue() + addValue; }
};

//Samo reference na objekte izvedenih klasa iz klase Num
void PrintFun(const Num& n) { printf("%d\n", n.GetValue()); }

int main() {
    Num x(5);
    NumIzvedena y(10);
    PrintFun (x);
    PrintFun (y);
    return 0;
}
```



Generički mehanizam - šabloni

- Isti algoritam za različite podatke (sortiranje prirodnih, celih, realnih brojeva)

```
void sort(int a[], int n) { ... }  
void sort(float a[], int n) { ... }  
void sort(specKlasa a[], int n) { ... }
```

```
void push(Klasa a) { ... }
```

```
char  max(char i, char j) {return i>j?i:j;}  
int   max(int i,  int j) {return i>j?i:j;}  
float max(float i,float j){return i>j?i:j;}
```

```
int min( int a, int b ) {  
    return ( a < b ) ? a : b; }
```

```
long min( long a, long b ) {  
    return ( a < b ) ? a : b; }
```

```
char min( char a, char b ) {  
    return ( a < b ) ? a : b; }
```

Da li mogu da odredim koji je pravougonik veći?

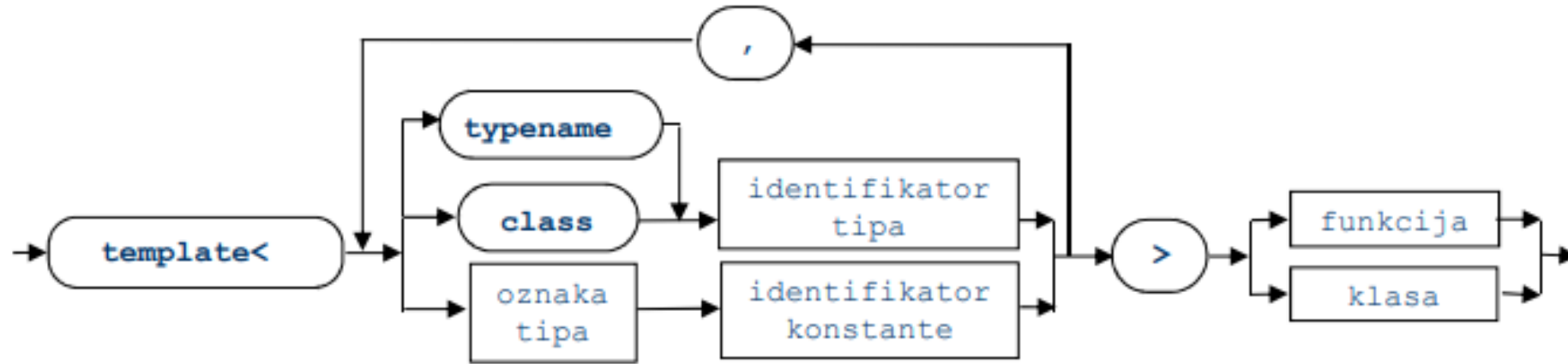
- U C++ postoji mogućnost definisanja šablona za opisivanje obrade za opšti slučaj, ne navodeći konkretne tipove podataka.
- Na osnovu šablona mogu da se automatski (prevodioc) generišu:
 1. konkretne **funkcije** za konkretne tipove podataka.
 2. konkretne **klase**.

■ Deklaracija šablona predstavlja okvir (kostur) na osnovu koga će biti kreirana klasa odnosno funkcija.

■ Konkretna klasa (odnosno funkcija) dobija se instanciranjem šablona

- Ako se koriste šabloni, onda se radi o generičkim funkcijama ili generičkim klasama na osnovu kojih se kasnije generišu konkretne funkcije ili klase.
- Slično funkcijama, ovi argumenti su formalni argumenti koji će biti zamenjeni stvarnim argumentima. Mehanizam je **statički** – instance šablona se prave tekstualnom zamenom u vreme prevođenja
- Sve funkcije tj. klase generisane na osnovu istog šablona imaju istu realizaciju a razlikuju se po tipovima sa kojima manipulišu.
- Generičke klase predstavljaju **parametrizovane tipove**.

Deklarisanje/Definisanje šablona



- Identifikatori tipa - (formalni argumenti šablona) mogu da se koriste unutar generičke funkcije ili klase na svim mestima gde se očekuju tipovi.
- Identifikatori konstante - mogu da se shvate kao simboličke konstante i da se koriste unutar generičke funkcije ili klase svuda gde se očekuju konstante.
- Oznaka tipa - za simboličke konstante mogu da budu standardni ili korisnički tipovi podataka.
- Odvojeno prevođenje šablona nema smisla – šabloni se smeštaju u *.h datoteke i uključuju tamo gde se koriste.
- **Mana šablona**: pošto su u *.h datotekama – korisnik vidi celu implementaciju algoritama, a ne samo interfejs.

//šablon funkcije

```
template <class T> T max(T a, T b){return a>b?a:b;}
```

//šablon prototipa

```
template <class T> void sort(T a[ ], int n);
```

//šablon definicije funkcije

```
template <class T> void sort(T a[ ], int n){/*...*/};
```

//šablon klase

```
template <class T, int k> class Vekt{  
    int duz;  
    T niz[k];  
public:  
    Vekt();  
    T& operator[](int i) const{return niz[i];}  
};
```

```
template <class T, int k> Vekt<T,k>::Vekt(){  
    duz=k; for(int i=0;i<k;niz[i++]=0);  
}
```

Generisanje funkcija

Funkcije na osnovu zadatog šablona se generišu:

1. automatski - kad se naiđe na poziv generičke f-je sa stvarnim argumentima koji mogu da se uklape u šablon bez konverzije tipova argumenata.
 2. na zahtev programera - navođenjem deklaracije (prototipa) za datu generičku f-ju sa željenim argumentima.
- Kada naiđe na poziv funkcije za koju ne postoji definicija, a postoji odgovarajuća generička funkcija, prevodilac generiše odgovarajuću definiciju funkcije.
 - **"Odgovarajuća generička funkcija"** znači: stvarni argumenti se bez konverzije uklapaju u formalne argumente.
 - Generisanje na zahtev se postiže navođenjem prototipa sa tipovima stvarnih argumenata.
 - Pri pozivanju ovako generisane funkcije vrše se uobičajene konverzije tipova.
 - Generisanje funkcije iz šablona će biti sprečeno ako se prethodno pojavi definicija odgovarajuće funkcije.
 - Generisanje funkcije iz šablona se vrši samo ako odgovarajuća funkcija (generisana ili definisana) ne postoji

```
template <class T> T max(T a, T b){return a>b?a:b;}
```

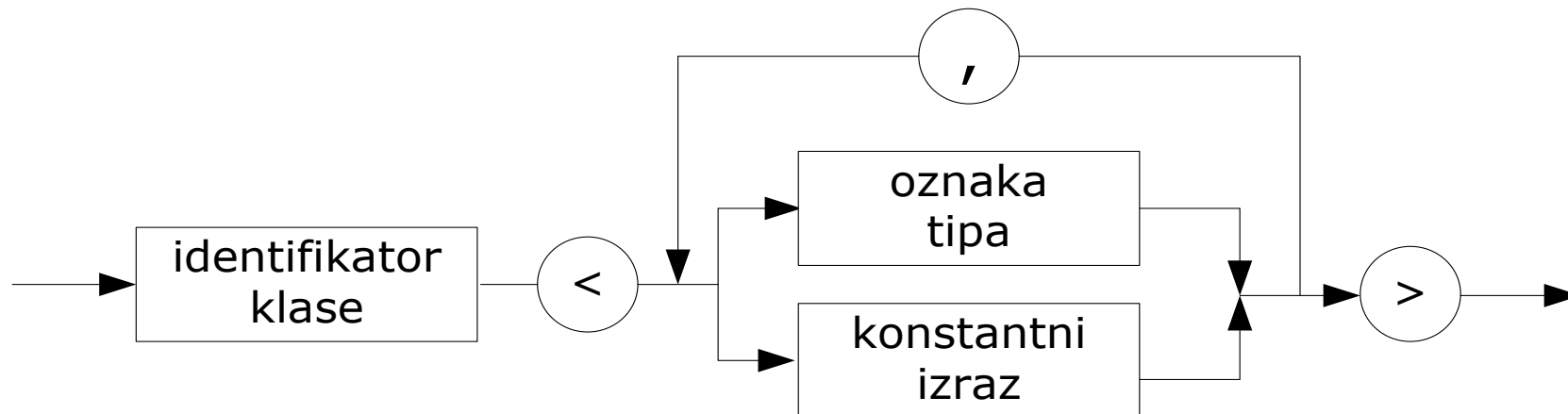
```
char *max(char *cp1,char *cp2){           // definicija obicne funkcije
    return strcmp(cp1,cp2)>=0?cp1:cp2;      // da se spreči poredjenje pokazivaca
}
void main() {
    int i1=1, i2=2; float f=0.5; char c1='a', c2='b'; char *a="alfa", *b="beta";
    int i=max(i1,i2);                      // generise se int max(int,int)
    char c=max(c1,c2);                    // generise se char max(char,char)
    int g=max(i1,c1);                      // ! GRESKA
    float max(float,float);                // zahtev za generisanje
    float g=max(i1,f);                     // poziva se float max(float,float)
    char *v=max(a,b);                     // poziva se char* max(char*,char*)
}
```

- Da bi generička funkcija mogla da se pozove za argumente i tipove, mora da se traži formiranje konkretne funkcije.
- Tako generisana funkcija koristi se kao i obična funkcija, pa će stvarni argumenti neodgovarajućeg tipa da se konvertuju u potreban tip pre poziva funkcije, ali samo ako postoji odgovarajuća automatska konverzija tipa.

Generisanje klasa

Formalni argumenti šablona klase mogu biti: **tipovi** (class T) ili **prosti objekti** (biće konkretizovani konstantnim izrazima)

- Konkretna klasa se generiše kada se prvi put naide na definiciju objekta u kojoj se koristi identifikator te klase.
- Pri generisanju klase se generišu i sve funkcije članice.
- Oznaka tipa generisane klase treba da sadrži:
 1. identifikator generičke klase i
 2. listu stvarnih argumenata za generičke tipove i konstante unutar <> iza identifikatora.
- Stvarni argument može biti:
 - oznaka tipa – zamenjuje formalni argument koji je identifikator tipa; oznaka tipa može biti standardni tip, identifikator klasa ili izvedeni tip (npr. pokazivač).
 - konstantni izraz: zamenjuje formalni argument koji je identifikator konstante; ako izraz sadrži operator > ovaj se mora pisati u zagradama (>).



```
template <class T, int n>
    class Vektor {
        T  v[n];
public:
    T& operator[] (int i) {
        if ((i>0) && (i<n)) return v[i];
        else error("van opsega");
    }
};
```

// primerci ove klase sadrže niz celih brojeva od 10 komponenti

```
Vektor <int, 10> niz1;
```

// primerci ove klase sadrže niz realnih brojeva u dvostrukoj tačnosti od 20 komponenti

```
Vektor <double, 20> niz2;
```

- Deklaracija šablona može biti samo globalna
- Svaka upotreba imena šablonske klase predstavlja deklaraciju konkretne šablonske klase
- Poziv šablonske funkcije ili uzimanje njene adrese predstavlja deklaraciju konkretne šablonske funkcije
- Ako se definiše obična negenerička klasa ili funkcija sa istim imenom i sa potpuno odgovarajućim tipovima kao što je šablon klase tj. funkcije, onda ova definicija predstavlja definiciju konkretne šablonske klase tj. funkcije za date tipove


```
template<class T>                // sablon
    void sort(vector<T>& v) { ... }
```

```
void sort(vector<int>& v) { ... } // konkretna funkcija
```

- Funkcija članica šablonske klase je **implicitno šablonska funkcija**
- Prijateljske funkcije šablonske klase nisu implicitno šablonske funkcije

ZADATAK: Sastaviti genericku klasu na C++ -u za statičke stekove zadatih kapaciteta. U glavnom programu prikazati mogućnosti te klase (rad steka ne zavisi od tipa objekta koji se smeštaju na stek).

```
// stek.h
template <class Pod, int kap>
class Stek
{
    Pod stek[kap]; //Stek u obliku niza
    int vrh; //Indeks vrha steka
    int gre; //Indikator greske
public:
    Stek(){vrh=gre=0;} //Inicijalizacija
    void stavi(const Pod &pod);
    void uzmi (Pod &pod);
    int vel() const {return vrh;} //Broj podataka na steku
    void brisi() {vrh=gre=0;}
    int prazan() const {return vrh==0;}
    int pun() const {return vrh==kap;}
    int greska() const {return gre;}
};
```

```
template <class Pod, int kap>
void Stek<Pod,kap>::stavi(const Pod &pod)
{
    if (!(gre=vrh==kap))
        stek[vrh++]=pod;
}
```

```
template <class Pod, int kap>
void Stek<Pod,kap>::uzmi(const Pod &pod)
{
    if (!(gre=vrh==0))
        pod=stek[--vrh];
}
```

```
// stek.cpp
#include <stek.h>
#include <iostream.h>
const int CVEL=10; DVEL=3;
void main() {
    int i;
    double d;
    Stek <int, CVEL> clb_stek;           //Stek sa celobrojnim podacima
    Stek <double, DVEL> dbl_stek;       //Stek sa realnim podacima
    while(!clb_stek.pun()) {
        cin>>i;
        clb_stek.stavi(i);
        cout<<i; }
    while(!clb_stek.prazan()) {
        clb_stek.uzmi(i);
        cout<<i; }
    while(!dbl_stek.pun()) {
        cin>>d;
        dbl_stek.stavi(d); }
    while(!dbl_stek.prazan()) {
        dbl_stek.uzmi(d);
        cout<<d; }
}
```

Zadatak: Sastaviti generičku funkciju na C++ jeziku za nalaženje ‘fuzije’ dva uređena niza objekata u treći, na isti način uređen niz.

Sastaviti glavni program koji korišćenjem predhodne funkcije nalazi fuziju nizova:

- a) celih brojeva uređenih po vrednosti.
- b) tačaka u ravni uređenih po odstojanjima od koordinatnog početka.
- c) pravougaonika uređenih po površinama.

```
#include <iostream.h>
template <class T>
void fuzija(T a[ ], int na, T b[ ], int nb, T c[ ])
{
    for(int ia=0, ib=0, ic=0; ia<na || ib<nb; ic++)
        c[ic]= ia==na ? b[ib++]:
                ib==nb ? a[ia++]:
                a[ia]<b[ib] ? a[ia++]: b[ib++];
}
```

```
class Tacka{
    double x,y;
public:
    // citanje tacke
    friend istream& operator>>(istream& dd, Tacka& tt);
    // upis tacke
    friend ostream& operator<<(ostream& dd, const Tacka& tt);

    friend int operator<(const Tacka& t1, const Tacka &t2);
};

inline istream& operator>>(istream& dd, Tacka& tt)
{ return dd>>tt.x>>tt.y;}

inline ostream& operator<<(ostream& dd, Tacka& tt)
{ return dd<<"T("<<tt.x<<" , "<<tt.y<<" ) " ;}

inline operator<(const Tacka& t1, const Tacka& t2)
{ return t1.x*t1.x+t1.y*t1.y<t2.x*t2.x+t2.y*t2.y;}
}
```

```
class Pravougaonik{
    double a,b ;
public:
    // citanje pravougaonika
    friend istream& operator>>(istream& dd, Pravougaonik& pp) ;
    // upis pravougaonika
    friend ostream& operator<<(ostream& dd, const Pravougaonik& pp) ;
    friend int operator<(const Pravougaonik& p1,
                        const Pravougaonik& p2) ;

};

inline istream& operator>>(istream& dd, Pravougaonik& pp)
{ return dd>>pp.a>>pp.b;}

inline ostream& operator<<(ostream& dd, Pravougaonik& pp)
{ return dd<<"P["<<pp.a<<" , "<<pp.b<<" ]" ;}

inline bool operator<(const Pravougaonik& p1, const Pravougaonik& p2)
{ return p1.a*p1.b<p2.a*p2.b;}

```

```
void main()
{
    int na, nb, nc, i;
    cout<<"Fuzija celih brojeva:\n";
    cin>>na;
    int *a=new int[na];
    for(i=0;i<na;i++)
        cin>>a[i];
    cin>>nb;
    int *b=new int[nb];
    for(i=0;i<nb;i++)
        cin>>b[i];
    nc=na+nb;
    int *c=new int[nc];
    fuzija(a, na, b, nb, c);
    for(i=0;i<nc;i++)
        cout<<" "<<c[i];

    delete []a;
    delete []b;
    delete []c;
```

```
cout<<"\nFuzija nizova tacaka:\n";
cin>>na;
Tacka *a=new Tacka[na];
for(i=0;i<na;i++)
    cin>>a[i];
cin>>nb;
Tacka *b=new Tacka[nb];
for(i=0;i<nb;i++)
    cin>>b[i];
nc=na+nb;
Tacka *c=new Tacka[nc];
fuzija(a, na, b, nb, c);
for(i=0;i<nc;i++)
    cout<<" "<<c[i];
delete []a;
delete []b;
delete []c;
```

```
cout<<"\nFuzija pravougaonika:\n";
cin>>na;
Pravougaonik *a=new Pravougaonik[na];
for(i=0;i<na;i++)
    cin>>a[i];
cin>>nb;
Pravougaonik *b=new Pravougaonik[nb];
for(i=0;i<nb;i++)
    cin>>b[i];
nc=na+nb;
Pravougaonik *c=new Pravougaonik[nc];
fuzija(a, na, b, nb, c);
for(i=0;i<nc;i++)
    cout<<" "<<c[i];

delete []a;
delete []b;
delete []c;
```

```
}
```