Osnovi računarskih sistema

Milena Vujošević–Janičić

Čas 1

Vežbe — 24 12 2004

1.1 Šablonske funkcije

Šabloni omogućavaju prevazilaženje ograničenja strogo tipiziranih jezika. Stroga tipiziranost ima kao posledicu da se i jednostavne funkcije moraju definisati više puta da bi se mogle upotrebljavati na raznim tipovima.

Primer 1

```
#include <iostream> using namespace std;
//min1() zato sto min() postoji u iostream-u
int min1( int x, int y ) {
    return x < y ? x : y;
}

main() {
    cout << min1( 2, 7 ) << endl;
    cout << min1( 12, 7 ) << endl;

    cout << min1( 3.4, 4.2 ) << endl;//mora se pisati nova f-ja
    //Odstampace se 3 a ne 3.4, zbog
    //automatske konverzije

    return 0;
}
```

Deo standardne C++ biblioteke je realizovan pomoću šablona. Treba voditi računa o tome da u starim verzijama prevodioca nisu implementirane sve mogućnosti šablona. Suština šablona je u zavisnosti šablona od nekih parametara (konstante ili tipovi tj. klase).

Primer 2

```
#include <iostream> using namespace std;
```

```
// napisemo definiciju za konkretan tip
/* int min1( int x, int y ) {
    return x < y ? x : y;
} */

// pa je prevedemo u sablon
template <class T> T min1( T x, T y ) {
    return x < y ? x : y;
}

// za konkretne tipove za koje sablon ne radi kako valja
// mozemo napisati konkretne implementacije
char* min1( char* x, char* y ) {
    return strcmp(x,y)<0 ? x : y;
}

const char* min1( const char* x, const char* y ) {
    return strcmp(x,y)<0 ? x : y;
}

main() {
    // ovo je puna sintaksa upotrebe sablona funkcija
    cout << min1<int>( 2, 7 ) << endl;

    // ovo je skracena sintaksa
    cout << min1( 12, 7 ) << endl;

    cout << min1( 3.4, 4.2 ) << endl;

    cout << min1<int>( 3.4, 4 ) << endl;
//  cout << min1( 3.4, 4 ) << endl; //greska.

//Ne moze se vrsiti nikakva konverzija argumenata f-je.
//Moze se resiti eksplicitnim navodjenjem
// tipa koji treba koristiti.
    cout << min1<double>( 3.4, 4 ) << endl;
    cout << min1<int>( 3.4, 4 ) << endl;

    cout << min1( "niska 1", "niska 2" ) << endl;
    cout << min1( "niska 2", "niska 1" ) << endl;

    return 0;
}
```

Pravila:

- Ukoliko šablon ne odgovara nekim tipovima moguće je predefinisati istoimenu funkciju.
- Određivanje koja će se funkcija primeniti obavlja se po sledećem redosledu:
  1. definisana funkcija istog ili kompatibilnog tipa;
  2. eksplicitno deklarisana funkcija istog ili kompatibilnog tipa;
  3. funkcionalni šablon potpuno istog tipa.
- Funkcijski šabloni se ne prevode. Za svaki konkretan upotrebljeni tip prevođenje se izvodi posebno.
- Da bi funkcijski šablon mogao da se primeni na neku klasu neophodno je da sve funkcije i operatori budu definisani na odgovarajućim tipovima.

Primer 3 *Podsećanje na inline funkcije*

```
#include <iostream> using namespace std;

// kljucna rec 'inline' sugerise prevodiocu
// da se telo funkcije umece umesto poziva
template <class T> inline T min1( T x, T y ) {
    return x < y ? x : y;
}

inline char* min1( char* x, char* y ) {
    return strcmp(x,y)<0 ? x : y;
} inline const char* min1( const char* x, const char* y ) {
    return strcmp(x,y)<0 ? x : y;
}
```

Primer 4

```
#include <iostream> using namespace std;

// sablon moze imati vise parametara
// a neki od parametara ne moraju biti klase nego konstante
template <class T, int velicina> T* napraviNiz() {
    return new T[velicina];
}

main() {

    // pravimo niz znakova duzine 200
    char* niska = napraviNiz<char,200>();
    delete [] niska;
```

```
    return 0;
}
```

Primer 5 *Podrazumevane vrednosti*

```
#include <iostream> using namespace std;

template <int n, class T> void ispis( T x ) {
    for( int i=0; i<n; i++ )
        cout << x << ' ';
    cout << '\b';
}

template <class T> void ispisIntervala( T x, T y, T korak=1 ) {
    for( T i=x; i<=y; i+=korak )
        cout << i << ' ';
    cout << '\b';
}

main() {

    ispis<5,int>(2);
    ispis<3,char>('w');
    cout << endl;


    //Ispisuje interval za karaktere
    //Konverzija neophodna da bi mogao
    //da se koristi skraceni zapis
    //sablona
    ispisIntervala( 'a', 'e', (char)2 );
    cout << endl;

    //Puna sintaksa, nije potrebno vrsiti
    //konverziju dvojke u char
    ispisIntervala<char>( 'a', 'e', 2 );
    cout << endl;
    ispisIntervala( 23.4, 27.8, 0.2 );
    cout << endl;

    //Koristi se podrazumevana vrednost za korak
    ispisIntervala( 23, 27);

    return 0;
}
```

```
/* Izlaz iz programa: 2 2 2 2 2w w w a c e a c e 23.4 23.6 23.8 24
24.2 24.4 24.6 24.8 25 25.2 25.4 25.6 25.8 26 26.2 26.4 26.6 26.8
27 27.2 27.4 27.6 27.8 23 24 25 26 27 */
```

1.2 Šabloni klasa

Napravili smo klasu Lista čiji elementi čuvaju vrednosti celobrojnog tipa. Kako napraviti Listu čiji elementi čuvaju vrednosti realnog tipa? Možemo izmeniti klasu Lista i na odgovarajućim mestima gde je pisalo int staviti double. Na kojim mestima? Ne baš na svim jer neki podaci, npr dužina liste, i dalje ostaju celobrojni. Dakle, treba biti pažljiv, ali stvar je izvodiva. Šta ako želimo da napravimo Listu čiji elementi čuvaju vrednosti tipa Razlomka? Ili listu Kompleksnih brojeva? Ili Listu koja sadrži Životinje? Ili Listu koja sadrži Autobuse? Ili Listu koja sadrži Liste?

Jasno je da za svaku Listu važe ista pravila i da im je ista osnovna struktura, razlikuju se samo vrednosti koju date Liste sadrže. C++ omogućava da se ove pravilnosti iskoriste i da se definiše samo jedna Lista, odnosno da se definiše šablon klase Lista, koji će nam onda omogućiti da koristimo odgovarajuće liste u zavisnosti od potrebe. To nam omogućava i lakše održavanje koda, umesto da po potrebi menjamo svaku od prethodno pomenutih lista, dovoljno je menjati samo šablon.

Dakle, mehanizam šablona jezika C++ omogućava automatsko generisanje klasnih tipova.

1.2.1 Definicija i deklaracija šablona

Na početku definicije ili deklraracije šablona klase uvek stoji ključna reč `template`. Iza ove ljučne reči uvek stoji lista parametara šablona koji su međusobno odvojeni zarezima, a koja se navodi između simbola `<` i `>`. Ova lista naziva se **lista parametara šablona**. Ona ne može da bude prazna.

U listi parametara šablona mogu biti **tipski parametri** i **obični parametri**.

Tipski parametar sastoji se od ključne reči `class` iza koje sledi identifikator.

Primer 6

```
template <class T> class Klasa1 { ... };
```

Bilo koji ugrađeni ili korisnički definisani tipovi, kao npr int, double, Razlomak, ... , mogu da budu ispravni argumenti za T. Šablon klase može imati i više tipskih parametara.

Primer 7

```
template <class T1, class T2, class T3> class Klasa2 { ... };
```

Primer 8 *Sledeća deklaracija bi izazvala grešku. Ne može ime istog parametra da se navede više puta.*

```
//Greska!!!
template <class T1, class T1> class Greska { ... };
```

Kada se jednom deklariše, tipski parametar služi kao specifikator tipa za ostatak definicije šablona klase. Unutar definicije šablona klase on može da se upotrebi na sasvim isti način kao što bi mogao neki ugrađeni ili korisnički definisan tip u definiciji nešablonske klase. Na primer, tipski parametar može da se koristi za deklarisanje podataka članova, funkcija članica, članova ugneždenih klasa itd.

Običan parametar šablona sastoji se od uobičajene deklaracije parametra. On označava da ime parametra predstavlja neku moguću vrednost. Ova vrednost predstavlja konstantu u definiciji šablona klase.

Primer 9

```
//u okviru klase Klasa3 N je konstanta
template <class T1, int N> class Klasa3 { ... };
```

Iza liste parametara šablona navodi se definicija ili deklaracija klase. Osim što sadrži parametre šablona, definicija šablona klase izgleda isto kao i definicija nešablonske klase.

Primer 10

```
template <class T> class ElementListe { ... private:
    T podatak;
    T* sledeci;
};
```

U prethodnom primeru, T se koristi da označi tip člana `podatak`. U tekstu programa , T će biti zamenjeno sa raznim korisnički definisanim ili ugrađenim tipovima. Ovaj proces zamene naziva se **konkretizacija** šablona.

Parametri šablona klase mogu da imaju **podrazumevane argumente**. Ovo važi kako za tipske parametre tako i za obične argumente. Ovo funkcioniše kao kod podrazumevanih argumenata za parametre funkcija. Podrazumevani argument za parametar šablona predstavlja tip ili vrednost koja se koristi ukoliko neki argument nije naveden prilikom konkretizacije šablona.

Primer 11

```
template <class T, int velicina = 256> class Niz { ... };
```

Primer 12 *Podrazumevane vrednosti.*

```
template <int velicina, class T = int> class Niz { ... };
```

1.2.2 Konkretizacija šablona klase

Definicija šablona klase određuje kako se konstruišu pojedine klase kada je dat skup od jednog ili više stvarnih tipova ili vrednosti.

Primer 13

```
//Definicija sablona
template <class T> class Klasa1 { ... };

//konkretizacij sablona
Klasa1<int> ki; Klasa1<double> kd; Klasa1<Razlomak> kr;
```

Primer 14

```
//Definicija sablona
template <class T, int velicina = 256> class Niz { ... };

//konkretizacij sablona
Niz<double, 200> n1; //Niz u kome je T tipa double, a velicina 200
Niz<Razlomak> n3;//T=Razlomak, a velicina uzima podrazumevanu
vrednost
```

Primer 15 *Šablon Par.*

```
#include <iostream> using namespace std;

class Par { public:
    Par( int p, int d )
        : _prvi(p), _drugi(d)
        {}

    int prvi() const
        { return _prvi; }
    int drugi() const
        { return _drugi; }

private:
    int _prvi;
    int _drugi;
};

ostream& operator << ( ostream& ostr, const Par& p ) {
    ostr << '(' << p.prvi() << ',' << p.drugi() << ')';
    return ostr;
}

main() {
    Par p(4,5);
    cout << p.prvi() << "," << p.drugi() << endl;
    cout << p << endl;
    return 0;
}
```

```
//Prevedimo ovu klasu u sablon

#include <iostream> using namespace std;

template <class T> class Par { public:
    Par( const T& p, const T& d )
        : _prvi(p), _drugi(d)
        {}

    const T& prvi() const
        { return _prvi; }
    const T& drugi() const
        { return _drugi; }

private:
    T _prvi;
    T _drugi;
};

template <class T> ostream& operator << ( ostream& ostr, const
Par<T>& p ) {
    ostr << '(' << p.prvi() << ',' << p.drugi() << ')';
    return ostr;
}

main() {
    Par<int> p(4,5);
    cout << p.prvi() << "," << p.drugi() << endl;
    cout << p << endl;

    Par<char> p1('a','b');
    cout << p1 << endl;

    //mora blanko izmedju (Par< Par...)
    Par< Par<double> > p2( Par<double>(1.1, 2.2),
                Par<double>(3.3,4.4));
    cout << p2 << endl;

    return 0;
} /*Izlaz iz programa 4,5 (4,5) (a,b) ((1.1,2.2),(3.3,4.4)) */
```

Primer 16 *Šablon vector iz standardne biblioteke šablona.*

```
#include <iostream> #include <vector>

using namespace std;

main() {
    vector<int> niz(20);
    for( int i=0; i<20; i+=2 )
        niz[i] = i*i;

    for( int i=0; i<20; i+=2 )
        cout << i << "^2 = " << niz[i] << endl;

    return 0;
}
```

Primer 17 *Matrica*

```
#include <iostream> #include <vector>

using namespace std;

//Matrica sa celobrojnim elementima
class Matrica { public:
    Matrica( unsigned v, unsigned s )
        : _Elementi( vector< vector<int> >( v ) )
        {
        for( int i=0; i<v; i++ ){
            _Elementi[i] = vector<int>(s);
            //  isto kao
            // vector<int> red(s);
            // _Elementi[i] = red;
            }
        }

    vector<int>& operator[] ( int n )
        { return _Elementi[n]; }

    const vector<int>& operator[] ( unsigned n ) const
        { return _Elementi[n]; }

private:
    vector< vector<int> > _Elementi;
};
```

```
main() {
    Matrica m( 3, 5 );
    for( int i=0; i<3; i++ )
        for( int j=0; j<5; j++ )
            m[i][j] = i*10 + j;

    for( int i=0; i<3; i++ ){
        for( int j=0; j<5; j++ )
            cout << m[i][j] << ' ';
        cout << endl;
        }

    return 0;
} /* Izlaz iz programa 0 1 2 3 4 10 11 12 13 14 20 21 22 23 24 */

//prevedimo u sablon

#include <iostream> #include <vector>

using namespace std;

template <class T> class Matrica { public:
    Matrica( unsigned v, unsigned s )
        : _Elementi( vector< vector<T> >( v ) )
        {
        for( int i=0; i<v; i++ ){
            _Elementi[i] = vector<T>(s);
            //  isto kao
            // vector<T> red(s);
            // _Elementi[i] = red;
            }
        }

    vector<T>& operator[] ( unsigned n )
        { return _Elementi[n]; }

    const vector<T>& operator[] ( unsigned n ) const
        { return _Elementi[n]; }

private:
    vector< vector<T> > _Elementi;
};
```

```
main() {
    Matrica<double> m( 3, 5 );
    for( int i=0; i<3; i++ )
        for( int j=0; j<5; j++ )
            m[i][j] = i + j/10.0;

    for( int i=0; i<3; i++ ){
        for( int j=0; j<5; j++ )
            cout << m[i][j] << ' ';
        cout << endl;
        }

    return 0;
}

/* Izlaz iz programa 0 0.1 0.2 0.3 0.4 1 1.1 1.2 1.3 1.4 2 2.1 2.2
2.3 2.4 */
```

Primer 18 *Šablon klase Lista.*

```
#include <iostream> using namespace std;

template <class T> class Lista { public:
    template <class T>
    class Element
    {
    public:
        T Vrednost() const
            { return _Vrednost;}

        const Element* Sledeci() const
            { return _Sledeci; }

    private:
        Element( const T& v  )
            : _Vrednost(v),
              _Sledeci(0)
            {}

        Element( const T& v, Element* s )
            : _Vrednost(v),
              _Sledeci(s)
            {}

        T           _Vrednost;
```

```
    Element*    _Sledeci;

    friend class Lista;
};

typedef Element<T> tipElementa;

Lista()
    : _Pocetak(0),
      _Kraj(0),
      _Velicina(0)
    {}

~Lista()
    { deinit(); }

Lista( const Lista& l )
    { init( l ); }

Lista& operator = ( const Lista& l )
    {
    if( &l != this ){
        deinit();
        init( l );
        }
    return *this;
    }

void DodajNaPocetak( const T& n )
    {
    _Pocetak = new tipElementa( n, _Pocetak );
    if( !_Kraj )
        _Kraj = _Pocetak;
    _Velicina++;
    }

void DodajNaKraj( const T& n )
    {
    tipElementa* novi = new tipElementa( n );
    if( !_Pocetak )
        _Kraj = _Pocetak = novi;
    else{
        _Kraj->_Sledeci = novi;
        _Kraj = novi;
```

```
        }
    _Velicina++;
    }

const T& operator [] ( unsigned n ) const
    {
    const tipElementa* p = _Pocetak;
    for( unsigned i=0; i<n && p; i++ )
        p = p->Sledeci();
    return p ? p->Vrednost() : 0;
    }

const tipElementa* Pocetak() const
    { return _Pocetak; }

bool Prazna() const
    { return !_Pocetak; }

unsigned Velicina() const
    { return _Velicina; }

void ObrisiPrviElement()
    {
    if( _Pocetak ){
        tipElementa* p = _Pocetak->_Sledeci;
        delete _Pocetak;
        _Pocetak = p;
        if( !_Pocetak )
            _Kraj = 0;
        _Velicina--;
        }
    }

void ObrisiPoslednjiElement()
    {
    if( _Velicina >=2 ){
        tipElementa* p = _Pocetak;
        while( p->Sledeci() != _Kraj )
            p = p->_Sledeci;
        delete p->Sledeci();
        p->_Sledeci = 0;
        _Kraj = p;
        }
    else if( _Velicina == 1 ){
```

```
            delete _Pocetak;
            _Pocetak = _Kraj = 0;
            }
        _Velicina--;
        }

private: /* ovako se moze izbeci pisanje konstruktora kopije i
operatora dodeljivanja
    Lista( const Lista& l );
    Lista& operator = ( const Lista& l );
*/

    void init( const Lista& l )
        {
        const tipElementa* stari = l._Pocetak;
        tipElementa** novi = &_Pocetak;
        _Kraj = 0;
        while( stari ){
            _Kraj = (*novi) = new tipElementa( stari->Vrednost() );
            stari = stari->Sledeci();
            novi = &_Kraj->_Sledeci;
            }
        *novi = 0;
        _Velicina = l._Velicina;
        }

    void deinit()
        {
        for( tipElementa* p = _Pocetak; p; ){
            tipElementa* pl = p->_Sledeci;
            delete p;
            p = pl;
            }
        }

    tipElementa*    _Pocetak;
    tipElementa*    _Kraj;
    unsigned        _Velicina;
};

template <class T> class Skup { public:
    void Dodaj( const T& n )
        {
        if( !Sadrzi(n) )
```

```
            Elementi.DodajNaKraj(n);
        }

    bool Sadrzi( const T& n ) const
        {
        for( const Lista<T>::tipElementa*
                p=Elementi.Pocetak(); p; p = p->Sledeci() )
            if( p->Vrednost() == n )
                return true;
        return false;
        }

private:
    Lista<T> Elementi;
};

main() {
    Skup<int> s;
    for( int i=0; i<20; i+=2 )
        s.Dodaj(i);

    for( int i=0; i<20; i++ )
        cout << "Skup " << (s.Sadrzi(i) ? "" : "ne ")
        << "sadrzi element " << i << endl;

    Skup<int> s2(s);
    s.Dodaj(123);
    cout << "Skup s " << (s.Sadrzi(123) ? "" : "ne ")
    << "sadrzi element " << 123 << endl;
    cout << "Skup s2 " << (s2.Sadrzi(123) ? "" : "ne ")
    << "sadrzi element " << 123 << endl;

    Skup s3;

    return 0;
}
```

Sadržaj