Razvoj softvera - Primeri, čas 7 - Polimorfizam

Saša Malkov

p11_sabloni.cpp

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

// pocetni primer, upotreba makroa
#define max1(x,y) ((x) > (y) ? (x) : (y))

int main()
{
   cout << max1(1,2) << endl;
   cout << max1(2.45, 2.54) << endl;

   int a=2;
   // koliko ce biti a?
   cout << max1(a++,a--) << endl;
   cout << a << endl;
}
```

p12_sabloni.cpp

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

// zamena makroa inline funkcijom
//   - jednako efikasno kao makro
//   - bezbedna sintaksa
//   - manja fleksibilnost - samo za jedan tip podataka
inline int max1( int x, int y )
{
   return x > y ? x : y;
}

int main()
{
   cout << max1(1,2) << endl;
   cout << max1(2.45, 2.54) << endl;

   int a=2;
   // koliko ce biti a?
   cout << max1(a++,a--) << endl;
```

```
    cout << a << endl;
}
```

p13_sabloni.cpp

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

// zamena funkcije sablonom funkcije
//   - jednako efikasno kao makro
//   - bezbedna sintaksa
//   - ista ili cak veca fleksibilnost
template <class T>
inline const T& max1( const T& x, const T& y )
{
    return x > y ? x : y;
}

int main()
{
    cout << max1(1,2) << endl;
    cout << max1(2.45, 2.54) << endl;

    // primer neispravnog implicitnog instanciranja
    // cout << max1(1, 2.5) << endl;
    // mora eksplicitno:
    cout << max1<double>(1, 2.5) << endl;
    cout << max1<char>( 'a', 53.2 ) << endl;

    int a=2;
    // koliko ce biti a?
    cout << max1(a++,a--) << endl;
    cout << a << endl;
}
```

p14_sabloni.cpp

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

// zamena funkcije sablonom funkcije
//   - jednako efikasno kao makro
//   - bezbedna sintaksa
//   - ista ili cak veca fleksibilnost
template <class T>
inline T max1( T x, T y )
{
    return x > y ? x : y;
}
```

```
// eksplicitna specijalizacija za const char*
template<>
inline const char* max1( const char* x, const char* y ) {
   cout << "poziv sablona" << endl;
   return strcmp(x,y)>0 ? x : y;
}

// naredna funkcija nije isto sto i prethodna, zato sto ona nije sablon
inline const char* max1( const char* x, const char* y ) {
   cout << "poziv funkcije" << endl;
   return x>y ? x : y;
}

// parametar sablona moze da bude i konstanta
// koja se vezuje u vreme prevodjenja
template<int N>
inline bool veceOd( int x )
{
   return x > N;
}


int main()
{
   cout << max1(1,2) << endl;
   cout << max1(2.45, 2.54) << endl;

   // primer neispravnog implicitnog instanciranja
   // cout << max1(1, 2.5) << endl;
   // mora eksplicitno:
   cout << max1<double>(1, 2.5) << endl;
   cout << max1<char>( 'a', 53.2 ) << endl;

   // ovde se poziva sablon za T=const char*
   cout << max1<>("aaa","bbb") << endl;

   // ovde se poziva funkcija
   cout << max1("aaa","bbb") << endl;

   // ovde se proverava da li je 10 vece od 5
   // 5 se vezuje u vreme prevodjenja, a 10 u vreme izvrsavanja
    // (u ovom slucaju se i 10 vezuje u vreme prevodjenja zbog 'inline')
   cout << veceOd<5>(10) << endl;
}
```

p21_sabloni_klasa.cpp

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

// primer jednostavne konkretne klase
```

```
class Tacka
{
public:
   int x,y,z;
   Tacka(int x0, int y0, int z0)
      : x(x0), y(y0), z(z0)
      {}
};

ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.x << "," << t.y << "," << t.z << ")";
   return ostr;
}

int main()
{
   Tacka t(1,2,3);
   cout << t << endl;
   cout << Tacka(1.2, 1.3, 1.4) << endl;
}
```

p22_sabloni_klasa.cpp

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

// pravljenje dve jednostavne konkretne klase
// koje se razlikuju samo po tipu elemenata

class Tacka
{
public:
   int x,y,z;
   Tacka(int x0, int y0, int z0)
      : x(x0), y(y0), z(z0)
      {}
};

ostream& operator<<( ostream& ostr, const Tacka& t )
{
   ostr << "(" << t.x << "," << t.y << "," << t.z << ")";
   return ostr;
}

class TackaR
{
public:
   float x,y,z;
   TackaR(float x0, float y0, float z0)
      : x(x0), y(y0), z(z0)
      {}
};

ostream& operator<<( ostream& ostr, const TackaR& t )
{
   ostr << "(" << t.x << "," << t.y << "," << t.z << ")";
   return ostr;
}

int main()
{
   Tacka t(1,2,3);
   cout << t << endl;
   cout << Tacka(1.2, 1.3, 1.4) << endl;
   cout << TackaR(1.2, 1.3, 1.4) << endl;
}
```

p23_sabloni_klasa.cpp

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

// uopstavanje razlicitih klasa jednim sablonom klase

template <class T>
class Tacka
{
public:
   T x,y,z;
   Tacka(T x0, T y0, T z0)
      : x(x0), y(y0), z(z0)
      {}
};

template <class T>
ostream& operator<<( ostream& ostr, const Tacka<T>& t )
{
   ostr << "(" << t.x << "," << t.y << "," << t.z << ")";
   return ostr;
}

int main()
{
   Tacka<int> t(1,2,3);
   cout << t << endl;
   cout << Tacka<int>(1.2, 1.3, 1.4) << endl;
   cout << Tacka<float>(1.2, 1.3, 1.4) << endl;
}
```

p24_sabloni_klasa.cpp

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

// podrazumevane vrednosti parametara sablona

template <class T = int>
class Tacka
{
public:
   T x,y,z;
   Tacka(T x0, T y0, T z0)
      : x(x0), y(y0), z(z0)
      {}
};

template <class T>
ostream& operator<<( ostream& ostr, const Tacka<T>& t )
{
   ostr << "(" << t.x << "," << t.y << "," << t.z << ")";
   return ostr;
}

int main()
{
   Tacka<> t(1,2,3);
   cout << t << endl;
   cout << Tacka<>(1.2, 1.3, 1.4) << endl;
   cout << Tacka<float>(1.2, 1.3, 1.4) << endl;
}
```

p401_funkcionali.cpp

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

// prebrojavanje neparnih podataka u nizu, klasican pristup

using namespace std;

unsigned prebrojNeparne( const vector<int>& niz )
{
   unsigned n=0;
   for( int x : niz  )
      if( x % 2 )
         n++;
   return n;
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebrojNeparne(niz) << endl;

   return 0;
}
```

p402_funkcionali.cpp

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

// izdvajanje uslova u funkciju

using namespace std;

bool neparan( int n )
{
   return n%2;
}

unsigned prebrojNeparne( const vector<int>& niz )
{
   unsigned n=0;
   for( int x : niz  )
      if( neparan( x ))
         n++;
   return n;
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebrojNeparne(niz) << endl;

   return 0;
}
```

p403_funkcionali.cpp

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

// upotreba funkcijskog parametra

using namespace std;
```

```
bool neparan( int n )
{
   return n%2;
}

unsigned prebroj( const vector<int>& niz, bool(*uslov)(int) )
{
   unsigned n=0;
   for( int x : niz  )
      if( uslov( x ))
         n++;
   return n;
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;

   return 0;
}
```

p404_funkcionali.cpp

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

// uopstavanje tipova pomocu sablona

using namespace std;

bool neparan( int n )
{
   return n%2;
}

template<typename T>
unsigned prebroj( const vector<T>& niz, bool(*uslov)(T) )
{
   unsigned n=0;
   for( const T& x : niz  )
      if( uslov( x ))
         n++;
   return n;
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;

   return 0;
}
```

p405_funkcionali.cpp

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

// uopstavanje obilaska kolekcije pomocu iteratora
// ovaj korak je nepotreban kada se koristi ranged-for
using namespace std;

bool neparan( int n )
{
   return n%2;
}

template<typename T>
unsigned prebroj( const vector<T>& niz, bool(*uslov)(T) )
{
   unsigned n=0;
   for( const T& x : niz )
      if( uslov( x ))
         n++;
   return n;
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;

   return 0;
}
```

p406_funkcionali.cpp

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

```
// uopstavanje tipa kolekcije, zato sto se i druge kolekcije
// obilaze na isti nacin pomocu iteratora
// i ovaj korak je nepotreban kada se koristi ranged-for
using namespace std;

bool neparan( int n )
{
   return n%2;
}

template<typename T, typename TK>
unsigned prebroj( const TK& kolekcija , bool(*uslov)(T) )
{
   unsigned n=0;
   for( const T& x : kolekcija )
      if( uslov( x ))
         n++;

    // ako se koriste iteratori, mora ovako:
   // kljucna rec 'typename' je nephodna po standardu da bi se
   // prevodiocu naglasilo da je TK::const_iterator ime tipa
// typename TK::const_iterator
//    i = kolekcija.begin(),
//    e = kolekcija.end();
// for( ; i!=e; i++ )
//    if( uslov( *i ))
//       n++;

   return n;
}

int main()
{
   // vector<int> niz;
   list<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;

   return 0;
}
```

p407_funkcionali.cpp

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

// izdvajanje obilaska pomocu iteratora
// veliki deo standardne bibilioteke pociva na tom pristupu
```

```
using namespace std;

bool neparan( int n )
{
   return n%2;
}

template<typename T, typename Iterator>
unsigned prebroj( Iterator beg, Iterator end, bool(*uslov)(T) )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename T, typename TK>
unsigned prebroj( const TK& kolekcija , bool(*uslov)(T) )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
    vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz.begin(), niz.end(),neparan) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35,neparan) << endl;

   // ovo ne moze sa svim kolekcijama, jer ne mogu svi iteratori da idu unazad
   cout << prebroj(niz.begin()+20, niz.end()-25,neparan) << endl;

   return 0;
}
```

p408_funkcionali.cpp

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

// parametrizujemo tip funkcije sablonom
// tj. parametarskim tipom Predikat
// "Predikat" je unarna logicka funkcija

using namespace std;

bool neparan( int n )
```

```
{
   return n%2;
}

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija , Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz.begin(), niz.end(),neparan) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35,neparan) << endl;

   // ovo ne moze sa svim kolekcijama, jer ne mogu svi iteratori da idu unazad
   cout << prebroj(niz.begin()+20, niz.end()-25,neparan) << endl;

   return 0;
}
```

p409_funkcionali.cpp

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

// dodajemo novu funkciju koju koristimo kao predikat

using namespace std;

bool neparan( int n )
{
   return n%2;
}

bool veciOd5( int n )
{
   return n > 5;
}

bool veciOd3( double n )
{
   return n > 5;
}

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija, Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz.begin(), niz.end(),neparan) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35,neparan) << endl;

   cout << prebroj(niz,veciOd5) << endl;
   cout << prebroj(niz.begin(), niz.end(),veciOd5) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35,veciOd5) << endl;

   // Primetimo da ovde mozemo da koristimo i funkciju koja ne ocekuje ceo broj.
   // To ne bi moglo da je "uslov" ostao sa fiksnim tipom.
   cout << prebroj(niz,veciOd3) << endl;

   return 0;
}
```

p410_funkcionali.cpp

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

// Dodajemo i koristimo funkcional
// "Funkcional" je objekat (klasa) koja moze da se koristi
// kao funkcija, tj. ima definisan operator().
// Ovaj operator moze da postoji u razlicitim verzijama,
// za razlicite brojeve i tipove argumenata.

using namespace std;

bool neparan( int n )
{
   return n%2;
}

bool veciOd5( int n )
{
   return n > 5;
}

class Neparan
{
public:
   bool operator()( int n )
      { return n%2; }
};

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija, Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz.begin(), niz.end(),neparan) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35,neparan) << endl;

   cout << prebroj(niz,veciOd5) << endl;
   cout << prebroj(niz.begin(), niz.end(),veciOd5) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35,veciOd5) << endl;

   // koristimo objekat funkcional
   Neparan nep;
   cout << prebroj(niz,nep) << endl;
   // koristimo privremeni objekat funkcional
   cout << prebroj(niz.begin(), niz.end(), Neparan() ) << endl;
   cout << prebroj(niz.begin()+20, niz.begin()+35, Neparan()) << endl;

   return 0;
}
```

p411_funkcionali.cpp

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

// Dodajemo parametrizovan funkcional.

using namespace std;

bool neparan( int n )
{
   return n%2;
}

bool veciOd5( int n )
{
   return n > 5;
}

class Neparan
{
public:
   bool operator()( int n )
      { return n%2; }
};

class VeciOd
{
public:
   VeciOd( int n )
      : _N(n)
   {}

   bool operator()( int n )
      { return n > _N; }

private:
   int _N;
};

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija, Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz,veciOd5) << endl;
   cout << prebroj(niz,Neparan()) << endl;

   cout << prebroj(niz,VeciOd(10)) << endl;

   for( int i=0; i<100; i+=5 )
      cout << i << " : " << prebroj(niz,VeciOd(i)) << endl;

   return 0;
}
```

p412_funkcionali.cpp

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

// Ako hocemo da funkciju parametrizujemo samo staticki
// (tj. u vreme prevodjenja)
// onda umesto funkcionala mozemo da koristimo sablon.

using namespace std;

bool neparan( int n )
{
   return n%2;
}

bool veciOd5( int n )
{
   return n > 5;
}
```

```
class Neparan
{
public:
   bool operator()( int n )
      { return n%2; }
};

class VeciOd
{
public:
   VeciOd( int n )
      : _N(n)
   {}

   bool operator()( int n )
      { return n > _N; }

private:
   int _N;
};

template<int n>
bool veciOd( int x )
{
   return x > n;
}

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija, Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz,veciOd5) << endl;
   cout << prebroj(niz,Neparan()) << endl;

   cout << prebroj(niz,veciOd<5>) << endl;
   for( int i=0; i<100; i+=5 )
      cout << i << " : " << prebroj(niz,VeciOd(i)) << endl;

   return 0;
}
```

p413_funkcionali.cpp

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

// Funkcionali su problematicni, zato sto se pisu udaljeno od mesta
// na kome se koriste, cak i onda kada se trivijalno definisu i kada
// smo sigurni da ih nikada vise necemo koristiti.
// Zbog toga C++ 11 uvodi pojednostavljenu "inline" sintaksu funkcionala,
// koja omogucava njihovo pisanje na mestu upotrebe, a prevod je prakticno
// ekvivalentan. To su tzv. "lambda funkcije".
// (mora da se prevodi za C++11)

using namespace std;

bool neparan( int n )
{
   return n%2;
}

bool veciOd5( int n )
{
   return n > 5;
}

class Neparan
{
public:
   bool operator()( int n )
      { return n%2; }
};

class VeciOd
{
public:
   VeciOd( int n )
      : _N(n)
   {}

   bool operator()( int n )
      { return n > _N; }

private:
   int _N;
};
```

```
template<int n>
bool veciOd( int x )
{
   return x > n;
}

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija, Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,neparan) << endl;
   cout << prebroj(niz,veciOd5) << endl;
   cout << prebroj(niz,Neparan()) << endl;

   cout << prebroj(niz,veciOd<5>) << endl;

   for( int i=0; i<100; i+=5 )
      cout << i << " : " << prebroj(niz,VeciOd(i)) << endl;

   // Ako je u pitanju jednostavna funkcija, tako se i prevodi.
   // Ovaj primer je ekvivalentan funkciji "neparan".
   //      [] - pocetak funkcije, ime nije potrebno zato sto je necemo drugde
koristiti
   //      (int n) - argumenti funkcije
   //      {...} - telo funkcije
   cout << prebroj( niz, [](int n){ return n%2; }) << endl;

   // Ako je u pitanju nesto slozenije, pa zelimo da funkciji prenesemo neke
   // promenljive koje su vidljive na mestu definisanja, onda je zapravo u pitanju
   // funkcional.
   // Ovaj primer je ekvivalentan funkcionalu "VeciOd".
   // [i] - naglasava da funkcional cuva parametar "i"
   //       kao da smo napisali:
   //       class VeciOd {
   //          int i;
   //          VeciOd(int _i) : i(_i) {}
   //          ...
   //       }
   //    i zatim napravili objekat sa "VeciOd(i)"
   for( int i=0; i<100; i+=5 )
      cout << i << " : "
          << prebroj( niz, [i](int n){ return n>i; } ) << endl;

   return 0;
}
```

p414_funkcionali.cpp

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

// Cesto imamo na raspolaganju funkciju koja je binarna
// a zelimo da je upotrebimo za definisanje unarne funkcije.
// Na primer, ako imamo "veciOd(x,y)" i zelimo da fiksiramo "y".
// Osim lambda funkcijama, to moze da se uradi i posebnim funkcionalom,
// koji 'pamti' i funkciju i argument.

using namespace std;

bool veciOd( int n, int m )
{
   return n > m;
}

// Funkcional Bind_2_2 pamti binarnu funkciju i njen 2. argument
template<typename Fn, typename T>
class Bind_2_2
{
public:
   Bind_2_2( Fn fn, T arg2 )
      : _Fn(fn), _Arg2(arg2)
   {}

   bool operator()( T n )
      { return _Fn(n,_Arg2); }

private:
   Fn _Fn;
   T _Arg2;
};

// Pravljenje sablonskog funkcionala zahteva eksplicitno navodjenje tipa
// na primer:
//       Bind_2_2<bool(*)(int,int),int>(veciOd,5)
// sto bas i nije mnogo zgodno za upotrebu,
// pa se zato pravi pomocni sablon funkcije, koji automatski raspoznaje
upotreblene tipove
```

```
template<typename Fn, typename T>
Bind_2_2<Fn,T> bind_2_2( Fn fn, T arg2 )
{
   return Bind_2_2<Fn,T>( fn, arg2 );
}

template<typename Iterator, typename Predikat>
unsigned prebroj( Iterator beg, Iterator end, Predikat uslov )
{
   unsigned n=0;
   for( Iterator i=beg; i!=end; i++ )
      if( uslov( *i ))
         n++;
   return n;
}

template<typename TK, typename Predikat>
unsigned prebroj( const TK& kolekcija, Predikat uslov )
{
   return prebroj( kolekcija.begin(), kolekcija.end(), uslov );
}

int main()
{
   vector<int> niz;
   for( unsigned i=0; i<100; i++ )
      niz.push_back(i);

   cout << prebroj(niz,Bind_2_2<bool(*)(int,int),int>(veciOd,17)) << endl;
   cout << prebroj(niz,bind_2_2(veciOd,17)) << endl;
   cout << prebroj(niz,bind_2_2([](int x,int y){return x>y;},17)) << endl;

   // Standardna biblioteka sadrzi napredniju verziju
   // funkcionala "bind", koja omogucava da se argumentom opise
   // koji se argument vezuje.

   return 0;
}
```

p61.niz.cpp

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

using namespace std;

//---------------------------------------------------------------
//  niz sa proveravanjem opsega

template<typename T>
class Niz : public vector<T>
```

```
{
public:
   Niz()
      : vector<T>()
   {}

   Niz( unsigned int sz )
      : vector<T>( sz )
   {}

   T& operator[]( unsigned i )
   {
      if( i >= vector<T>::size() )
         throw out_of_range("Indeks van opsega");
      return vector<T>::operator[](i);
   }

   const T& operator[]( unsigned i ) const
   {
      if( i >= vector<T>::size() )
         throw out_of_range("Indeks van opsega");
      return vector<T>::operator[](i);
   }
};

//---------------------------------------------------------------
int main()
{
   try   {
      Niz<int> niz(20);

      for( unsigned i=0; i<niz.size(); i++ )
         niz[i] = i*i;
      for( unsigned i=0; i<=niz.size(); i++ )
         cout << i << " : " << niz[i] << endl;

   }catch( exception& e ){
      cerr << "*** GRESKA: " << e.what() << endl;
   }

   return 0;
}
```

p62.niz.cpp

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

using namespace std;

// nizovi sa i bez provere indeksa
```

```
//---------------------------------------------------------------
class ProveraIndeksa {
public:
   static void Provera( unsigned indeks, unsigned opseg ){
      if( indeks >= opseg )
         throw out_of_range("Indeks van opsega");
   }
};

class BezProvereIndeksa{
public:
   static void Provera( unsigned indeks, unsigned opseg ){}
};

//---------------------------------------------------------------
template<typename T,typename ProveravacIndeksa>
class Niz : public vector<T>
{
public:
   Niz( unsigned n )
      : vector<T>(n)
      {}

   T& operator[]( unsigned i )
   {
      ProveravacIndeksa::Provera( i, vector<T>::size() );
      return vector<T>::operator[](i);
   }

   const T& operator[]( unsigned i ) const
   {
      ProveravacIndeksa::Provera( i, vector<T>::size() );
      return vector<T>::operator[](i);
   }
};

//---------------------------------------------------------------
int main()
{
   try   {
      Niz<int,ProveraIndeksa> niz(10);
      for( unsigned i=0; i<niz.size(); i++ )
         niz[i] = i*i;
      for( unsigned i=0; i<=niz.size(); i++ )
         cout << i << " : " << niz[i] << endl;

      cout << niz[9999999] << endl;

   }catch( exception& e ){
      cerr << "*** GRESKA: " << e.what() << endl;
   }
   return 0;
}
```

p71.matrica.cpp

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

using namespace std;

// nizovi i matrice sa proverom indeksa
// 'matrice' sa vecim brojem dimenzija

//---------------------------------------------------------------
class ProveraIndeksa {
public:
   static void Provera( unsigned indeks, unsigned opseg ){
      if( indeks >= opseg )
         throw out_of_range("Indeks van opsega");
   }
};

class BezProvereIndeksa{
public:
   static void Provera( unsigned indeks, unsigned opseg ){}
};

//---------------------------------------------------------------
template<typename T,typename ProveravacIndeksa>
class Niz : public vector<T>
{
public:
   T& operator[]( unsigned i )
   {
      ProveravacIndeksa::Provera( i, vector<T>::size() );
      return vector<T>::operator[](i);
   }

   const T& operator[]( unsigned i ) const
   {
      ProveravacIndeksa::Provera( i, vector<T>::size() );
      return vector<T>::operator[](i);
   }
};

//---------------------------------------------------------------
template<typename T, unsigned Dim, typename ProveravacIndeksa>
class Matrica
{
public:
   typedef Matrica<T,Dim-1,ProveravacIndeksa> tElementa;
   Matrica()
   {}

   Matrica( unsigned dims[] ) {
      PostaviVelicinu( dims );
   }

   Matrica(unsigned d1,
         unsigned d2=0,
         unsigned d3=0,
         unsigned d4=0,
         unsigned d5=0,
         unsigned d6=0,
         unsigned d7=0,
         unsigned d8=0
         )
   {
      unsigned d[] = { d1, d2, d3, d4, d5, d6, d7, d8 };
      PostaviVelicinu(d);
   }

   void PostaviVelicinu( unsigned dims[] ) {
      _Elementi.resize(dims[0]);
      for( unsigned i=0; i<dims[0]; i++ )
         _Elementi[i].PostaviVelicinu(dims+1);
   }

   tElementa& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const tElementa& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      if( d==1 )
         return _Elementi.size();
      else
         return _Elementi[0].size(d-1);
   }

private:
   Niz<tElementa,ProveravacIndeksa> _Elementi;
};

//------------------------------------------------------------
template<typename T, typename ProveravacIndeksa>
class Matrica<T,1,ProveravacIndeksa>
{
public:
   Matrica()
   {}
   void PostaviVelicinu( unsigned dims[] ) {
      _Elementi.resize(dims[0]);
   }

   T& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const T& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      ProveravacIndeksa::Provera( d-1, 1 );
      return _Elementi.size();
   }

private:
   Niz<T,ProveravacIndeksa> _Elementi;
};


//---------------------------------------------------------------
int main()
{
   try   {
      Matrica<int,2,ProveraIndeksa> m(10,5);
      for(unsigned i=0;i<m.size(1);i++)
         for(unsigned j=0; j<m.size(2); j++ )
            m[i][j] = i*100 + j;

      for(unsigned i=0;i<m.size(1);i++){
         for(unsigned j=0; j<m.size(2); j++ )
            cout << m[i][j] << ' ';
         cout << endl;
      }
      cout << endl;

      cout << m[5][10] << endl;

   }catch( exception& e ){
      cerr << "*** GRESKA: " << e.what() << endl;
   }

   return 0;
}
```

p72.matrica.cpp

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

```cpp
#include <stdexcept>

using namespace std;

// nizovi i matrice sa proverom indeksa
// 'matrice' sa vecim brojem dimenzija

//---------------------------------------------------------------
class ProveraIndeksa {
public:
   static void Provera( unsigned indeks, unsigned opseg ){
      if( indeks >= opseg )
         throw out_of_range("Indeks van opsega");
   }
};

class BezProvereIndeksa{
public:
   static void Provera( unsigned indeks, unsigned opseg ){}
};

//---------------------------------------------------------------
template<typename T,typename ProveravacIndeksa>
class Niz : public vector<T>
{
public:
   T& operator[]( unsigned i )
   {
      ProveravacIndeksa::Provera( i, size() );
      return vector<T>::operator[](i);
   }

   const T& operator[]( unsigned i ) const
   {
      ProveravacIndeksa::Provera( i, size() );
      return vector<T>::operator[](i);
   }

   using vector<T>::size;
};

//---------------------------------------------------------------
template<typename T, unsigned Dim, typename ProveravacIndeksa>
class Matrica
{
public:
   typedef Matrica<T,Dim-1,ProveravacIndeksa> tElementa;

   Matrica()
   {}

   Matrica( unsigned dims[] ) {
      PostaviVelicinu( dims );
   }
   Matrica(unsigned d1,
         unsigned d2=0,
         unsigned d3=0,
         unsigned d4=0,
         unsigned d5=0,
         unsigned d6=0,
         unsigned d7=0,
         unsigned d8=0
         )
   {
      ProveravacIndeksa::Provera( Dim-1, 8 );
      unsigned d[] = { d1, d2, d3, d4, d5, d6, d7, d8 };
         PostaviVelicinu(d);
   }

   void PostaviVelicinu( unsigned dims[] ) {
      _Elementi.resize(dims[0]);
      for( unsigned i=0; i<dims[0]; i++ )
         _Elementi[i].PostaviVelicinu(dims+1);
   }

   tElementa& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const tElementa& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      if( d==1 )
         return _Elementi.size();
      else
         return _Elementi[0].size(d-1);
   }

private:
   Niz<tElementa,ProveravacIndeksa> _Elementi;
};

//---------------------------------------------------------------
template<typename T, typename ProveravacIndeksa>
class Matrica<T,1,ProveravacIndeksa>
{
public:
   Matrica()
   {}

   void PostaviVelicinu( unsigned dims[] ) {
      _Elementi.resize(dims[0]);
   }

   T& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const T& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      ProveravacIndeksa::Provera( d-1, 1 );
      return _Elementi.size();
   }

private:
   Niz<T,ProveravacIndeksa> _Elementi;
};


//-------------------------------------------------------------
int main()
{
   try   {
      Matrica<int,3,ProveraIndeksa> m(10,2,5);
      cout << m.size(1) << endl;
      cout << m.size(2) << endl;
      cout << m.size(3) << endl;
      cout << endl;

   }catch( exception& e ){
      cerr << "*** GRESKA: " << e.what() << endl;
   }

   return 0;
}
```

p73.matrica.cpp

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

using namespace std;

// dodati testovi

//-------------------------------------------------------------
class ProveraIndeksa {
public:
   static void Provera( unsigned indeks, unsigned opseg ){
      if( indeks >= opseg )
         throw out_of_range("Indeks van opsega");
   }
};
```

```cpp
class BezProvereIndeksa{
public:
   static void Provera( unsigned indeks, unsigned opseg ){}
};

//---------------------------------------------------------------
template<typename T,typename ProveravacIndeksa>
class Niz : public vector<T>
{
public:
   T& operator[]( unsigned i )
   {
      ProveravacIndeksa::Provera( i, size() );
      return vector<T>::operator[](i);
   }

   const T& operator[]( unsigned i ) const
   {
      ProveravacIndeksa::Provera( i, size() );
      return vector<T>::operator[](i);
   }

   using vector<T>::size;
};

//---------------------------------------------------------------
template<typename T, unsigned Dim, typename ProveravacIndeksa>
class Matrica
{
public:
   typedef Matrica<T,Dim-1,ProveravacIndeksa> tElementa;

   Matrica()
   {}

   Matrica( unsigned dims[] ) {
      PostaviVelicinu( dims );
   }

   Matrica(unsigned d1,
         unsigned d2=0,
         unsigned d3=0,
         unsigned d4=0,
         unsigned d5=0,
         unsigned d6=0,
         unsigned d7=0,
         unsigned d8=0
         )
   {
      ProveravacIndeksa::Provera( Dim-1, 8 );
      unsigned d[] = { d1, d2, d3, d4, d5, d6, d7, d8 };
         PostaviVelicinu(d);
   }
   void PostaviVelicinu( unsigned dims[] ) {
      _Elementi.resize(dims[0]);
      for( unsigned i=0; i<dims[0]; i++ )
         _Elementi[i].PostaviVelicinu(dims+1);
   }

   tElementa& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const tElementa& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      if( d==1 )
         return _Elementi.size();
      else
         return _Elementi[0].size(d-1);
   }

private:
   Niz<tElementa,ProveravacIndeksa> _Elementi;
};

//---------------------------------------------------------------
template<typename T, typename ProveravacIndeksa>
class Matrica<T,1,ProveravacIndeksa>
{
public:
   Matrica()
   {}

   void PostaviVelicinu( unsigned dims[] ) {
      _Elementi.resize(dims[0]);
   }

   T& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const T& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      ProveravacIndeksa::Provera( d-1, 1 );
      return _Elementi.size();
   }

private:
   Niz<T,ProveravacIndeksa> _Elementi;
};
```

```
//---------------------------------------------------------------
void test2()
{
   Matrica<int,2,ProveraIndeksa> m(10,5);
   for(unsigned i=0;i<m.size(1);i++)
      for(unsigned j=0; j<m.size(2); j++ )
         m[i][j] = i*100 + j;

   for(unsigned i=0;i<m.size(1);i++){
      for(unsigned j=0; j<m.size(2); j++ )
         cout << m[i][j] << ' ';
      cout << endl;
   }
}

void test3()
{
   Matrica<int,3,ProveraIndeksa> m(4,5,6);
   for(unsigned i=0;i<m.size(1);i++)
      for(unsigned j=0; j<m.size(2); j++ )
         for(unsigned k=0; k<m.size(3); k++ )
            m[i][j][k] = (i+1)*100 + (j+1)*10 + (k+1);

   for(unsigned i=0;i<m.size(1);i++){
      for(unsigned j=0; j<m.size(2); j++ ){
         for(unsigned k=0; k<m.size(3); k++ )
            cout << m[i][j][k] << ' ';
         cout << endl;
      }
      cout << "------------------" << endl;
   }
}

void test3a()
{
   Matrica<int,3,ProveraIndeksa> m(4,5,6);
   for(unsigned i=0;i<m.size(1);i++){
      Matrica<int,2,ProveraIndeksa> m2(5,6);
      for(unsigned k=0;k<m2.size(1);k++)
         for(unsigned j=0; j<m2.size(2); j++ )
            m2[k][j] = (i+1)*1000 + k*100 + j;
      m[i] = m2;
   }

   for(unsigned i=0;i<m.size(1);i++){
      for(unsigned j=0; j<m.size(2); j++ ){
         for(unsigned k=0; k<m.size(3); k++ )
            cout << m[i][j][k] << ' ';
         cout << endl;
      }
      cout << "------------------" << endl;
   }
}
```

```
int main()
{
   try   {
      test2();
      test3();
      test3a();
   }catch( exception& e ){
      cerr << "*** GRESKA: " << e.what() << endl;
   }

   return 0;
}
```

p74.matrica.cpp

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

using namespace std;

// dodati testovi

//--------------------------------------------------------------
class ProveraIndeksa {
public:
   static void Provera( unsigned indeks, unsigned opseg ){
      if( indeks >= opseg )
         throw out_of_range("Indeks van opsega");
   }
};

class BezProvereIndeksa{
public:
   static void Provera( unsigned indeks, unsigned opseg ){}
};

//--------------------------------------------------------------
template<typename T,typename ProveravacIndeksa>
class Niz : public vector<T>
{
public:
   T& operator[]( unsigned i )
   {
      ProveravacIndeksa::Provera( i, size() );
      return vector<T>::operator[](i);
   }

   const T& operator[]( unsigned i ) const
   {
      ProveravacIndeksa::Provera( i, size() );
      return vector<T>::operator[](i);
   }

   using vector<T>::size;
};

//-----------------------------------------------------------------
template<typename T, unsigned Dim, typename ProveravacIndeksa>
class Matrica
{
public:
   typedef Matrica<T,Dim-1,ProveravacIndeksa> tElementa;

   Matrica()
   {}

   template<typename... Args>
   Matrica( unsigned n, Args... args )
   {
      PostaviVelicinu( n, args... );
   }

   template<typename... Args>
   void PostaviVelicinu( unsigned dim, Args... args ) {
      _Elementi.resize( dim );
      for( auto& m: _Elementi )
         m.PostaviVelicinu( args... );
   }

   tElementa& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const tElementa& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      if( d==1 )
         return _Elementi.size();
      else
         return _Elementi[0].size(d-1);
   }

private:
   Niz<tElementa,ProveravacIndeksa> _Elementi;
};

//-----------------------------------------------------------------
template<typename T, typename ProveravacIndeksa>
class Matrica<T,1,ProveravacIndeksa>
{
public:
   Matrica()
   {}

   Matrica( unsigned n  )
   {
      PostaviVelicinu( n );
   }

   void PostaviVelicinu( unsigned dim ) {
      _Elementi.resize(dim);
   }

   T& operator[]( unsigned d1 ) {
      return _Elementi[d1];
   }

   const T& operator[]( unsigned d1 ) const {
      return _Elementi[d1];
   }

   unsigned size( unsigned d ) const {
      ProveravacIndeksa::Provera( d-1, 1 );
      return _Elementi.size();
   }

private:
   Niz<T,ProveravacIndeksa> _Elementi;
};

//-------------------------------------------------------------
void test2()
{
   Matrica<int,2,ProveraIndeksa> m(10,5);
   for(unsigned i=0;i<m.size(1);i++)
      for(unsigned j=0; j<m.size(2); j++ )
         m[i][j] = i*100 + j;

   for(unsigned i=0;i<m.size(1);i++){
      for(unsigned j=0; j<m.size(2); j++ )
         cout << m[i][j] << ' ';
      cout << endl;
   }
}

void test3()
{
   Matrica<int,3,ProveraIndeksa> m(4,5,6);
   for(unsigned i=0;i<m.size(1);i++)
      for(unsigned j=0; j<m.size(2); j++ )
         for(unsigned k=0; k<m.size(3); k++ )
            m[i][j][k] = (i+1)*100 + (j+1)*10 + (k+1);

   for(unsigned i=0;i<m.size(1);i++){
      for(unsigned j=0; j<m.size(2); j++ ){
         for(unsigned k=0; k<m.size(3); k++ )
            cout << m[i][j][k] << ' ';
         cout << endl;
      }
      cout << "------------------" << endl;
   }
}

void test3a()
{
   Matrica<int,3,ProveraIndeksa> m(4,5,6);
   for(unsigned i=0;i<m.size(1);i++){
      Matrica<int,2,ProveraIndeksa> m2(5,6);
      for(unsigned k=0;k<m2.size(1);k++)
         for(unsigned j=0; j<m2.size(2); j++ )
            m2[k][j] = (i+1)*1000 + k*100 + j;
      m[i] = m2;
   }

   for(unsigned i=0;i<m.size(1);i++){
      for(unsigned j=0; j<m.size(2); j++ ){
         for(unsigned k=0; k<m.size(3); k++ )
            cout << m[i][j][k] << ' ';
         cout << endl;
      }
      cout << "------------------" << endl;
   }
}


int main()
{
   try   {
      test2();
      test3();
      test3a();
   }catch( exception& e ){
      cerr << "*** GRESKA: " << e.what() << endl;
   }

   return 0;
}
```

p79.matrica.cpp

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

using namespace std;

// Primer sa efikasnom organizacijom memorije
// Matrica je niz elemenata sa slozenim indeksiranjem

//---------------------------------------------------------------
// Politike provere indeksa
//---------------------------------------------------------------
struct SaProveromIndeksa {
   static void Provera( int indeks, int velicina ) {
      if( indeks >= velicina )
         throw new out_of_range("Indeks van opsega!");
   }
};

struct BezProvereIndeksa {
   static void Provera( int indeks, int velicina ){}
};


//---------------------------------------------------------------
// Struktura svih matrica je ista
//---------------------------------------------------------------
template<class T,unsigned Dim,class ProveravacIndeksa>
class MatricaStructBase
{
public:
   typedef T tElement;

   static unsigned Dimenzija()
      { return Dim; }

   void PostaviVelicinu( unsigned* dimenzije )
   {
      _Velicina.clear();
      _Podvelicina.clear();
      unsigned n = 1;
      for( unsigned i=0; i<Dim; i++ ){
         _Podvelicina.push_back(n);
         _Velicina.push_back(dimenzije[i]);
         n *= dimenzije[i];
      }
      _Elementi.resize(n);
   }

   unsigned Velicina( unsigned n ) const
      { return _Velicina[n]; }


   unsigned RelativniIndeks( unsigned dim, unsigned i )
   {
      ProveravacIndeksa::Provera( i, _Velicina[dim] );
      return i * _Podvelicina[dim];
   }

   tElement& Element( unsigned i )
      { return _Elementi[i]; }
protected:
   vector< tElement >   _Elementi;     // elementi matrice
   vector< unsigned >   _Velicina;     // koliki je raspon kog indeksa
   vector< unsigned >   _Podvelicina;  // koliko ima elemenata podmatrica (*this)
[0], (*this)[0][1], (*this)[0][1][2],...
};


//---------------------------------------------------------------
// Referenca na podmatricu
//---------------------------------------------------------------
template<class tMatrica, unsigned Dim>
class ReferencaPodmatrice
{
public:
   ReferencaPodmatrice( tMatrica& mat, unsigned i )
      : _Matrica( mat ), _RelativniIndeks( i )
      {}

   ReferencaPodmatrice<tMatrica,Dim-1> operator[]( unsigned i )
   {
      unsigned indeks = _RelativniIndeks + _Matrica.RelativniIndeks(
tMatrica::Dimenzija() - Dim, i );
      return ReferencaPodmatrice<tMatrica,Dim-1>( _Matrica, indeks );
   }

protected:
   tMatrica&   _Matrica;
   unsigned _RelativniIndeks;
};


template<class tMatrica>
class ReferencaPodmatrice<tMatrica,1>
{
public:
   ReferencaPodmatrice( tMatrica& mat, unsigned i )
      : _Matrica( mat ), _RelativniIndeks( i )
      {}

   typename tMatrica::tElement& operator[]( unsigned i )
   {
      unsigned indeks = _RelativniIndeks + _Matrica.RelativniIndeks(
tMatrica::Dimenzija() - 1, i );
      return _Matrica.Element( indeks );
   }

protected:
   tMatrica&   _Matrica;
   unsigned _RelativniIndeks;
};
```

```
//-----------------------------------------------------------------
// Univerzalna matrica sa dim dimenzija
//-----------------------------------------------------------------
template<class tElement,unsigned Dim,class ProveravacIndeksa=BezProvereIndeksa>
class Matrica : public MatricaStructBase< tElement,Dim,ProveravacIndeksa >
{
public:
   Matrica()
      {}

   Matrica( unsigned* dimenzije )
      { MatricaStructBase< tElement,Dim,ProveravacIndeksa >::PostaviVelicinu(
dimenzije ); }

   ReferencaPodmatrice<Matrica,Dim-1> operator[]( unsigned i )
   {
      unsigned indeks = MatricaStructBase< tElement,Dim,ProveravacIndeksa
>::RelativniIndeks( 0, i );
      return ReferencaPodmatrice<Matrica,Dim-1>( *this, indeks );
   }
};

//-----------------------------------------------------------------
// Primeri
//-----------------------------------------------------------------
void test2()
{
   unsigned dim[] = {2,5};
   Matrica<int,2,SaProveromIndeksa> m(dim);

   for( unsigned i=0; i<m.Velicina(0); i++ )
      for( unsigned j=0; j<m.Velicina(1); j++ )
         m[i][j] = i*10 + j;

   for( unsigned i=0; i<m.Velicina(0); i++ ){
      for( unsigned j=0; j<m.Velicina(1); j++ )
         cout << m[i][j] << ' ';
      cout << endl;
   }
}

void test3()
{
   unsigned dim[] = {2,5,7};
   Matrica<int,3,SaProveromIndeksa> m(dim);

   for( unsigned i=0; i<m.Velicina(0); i++ )
      for( unsigned j=0; j<m.Velicina(1); j++ )
         for( unsigned k=0; k<m.Velicina(2); k++ )
            m[i][j][k] = i*100 + j*10 + k;

   for( unsigned i=0; i<m.Velicina(0); i++ ){
      for( unsigned j=0; j<m.Velicina(1); j++ ){
         for( unsigned k=0; k<m.Velicina(2); k++ )
```

```
            cout << m[i][j][k] << ' ';
         cout << endl;
      }
      cout << "------------------------" << endl;
   }
}

void test4()
{
   unsigned dim[] = {2,5,7,2};
   Matrica<int,4,SaProveromIndeksa> m(dim);

   for( unsigned i=0; i<m.Velicina(0); i++ )
      for( unsigned j=0; j<m.Velicina(1); j++ )
         for( unsigned k=0; k<m.Velicina(2); k++ )
            for( unsigned l=0; l<m.Velicina(3); l++ )
               m[i][j][k][l] = i*1000 + j*100 + k*10 + l;

   for( unsigned i=0; i<m.Velicina(0); i++ ){
      for( unsigned j=0; j<m.Velicina(1); j++ ){
         for( unsigned k=0; k<m.Velicina(2); k++ ){
            for( unsigned l=0; l<m.Velicina(3); l++ )
               cout << m[i][j][k][l] << ' ';
            cout << endl;
         }
         cout << "------------------------" << endl;
      }
      cout << "===========================" << endl;
   }
}

int main()
{
   test2();
   cout << endl;

   test3();
   cout << endl;

   test4();

   return 0;
}
```