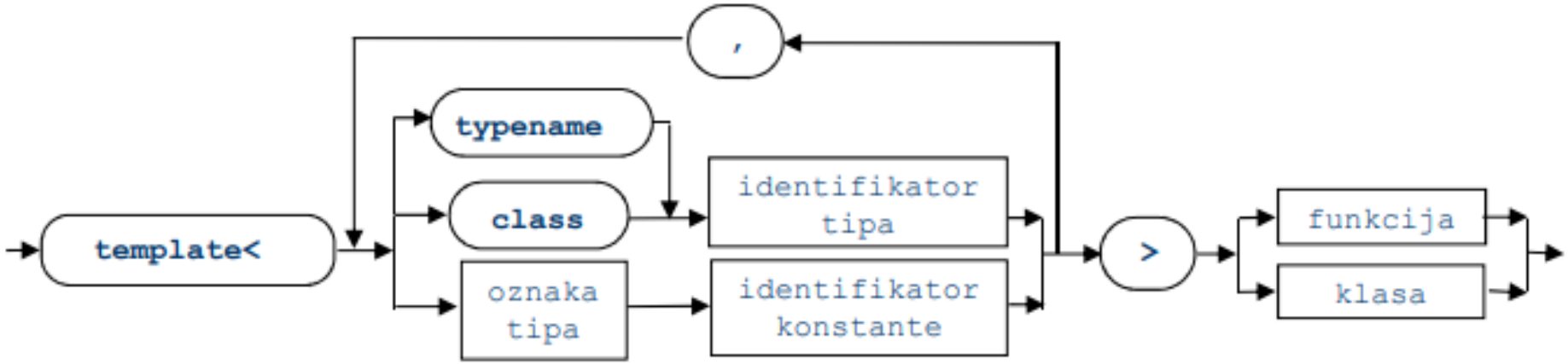
Generički mehanizam - šabloni – II deo

Predavanja br. 10

Teme

- Parametri šablona - vrste
- Vrednosti kao parametri šablona
- Pokazivači na funkcije kao parametri šablona
- Ograničavanja tipa parametara šablona
- Podrazumevane vrednosti parametara šablona
- Specijalizacija šablona
- Specijalizacija generičkih funkcija
- Nasleđivanje šablona
- Šabloni kao parametri šablona

Sintaksni dijagram



- class ili typename

Deklarisanje šablona

- Deklarisanjem šablona specificira se skup parametrizovanih klasa ili funkcija

```
template <lista_parametara>
                deklaracija_funkcije_ili_klase
```

- **`lista_parametara`** - zarezom razdvojeni parametri šablona.

Deklarisanje šablona

- Deklaracija šablona ne generiše kod, već specificira familiju funkcija ili klasa koje će biti generisane (instancirane) po potrebi.
- Parametri šablona mogu da budu:
- **A) tipovi**

```
class Identifikator,
typename Identifikator,
template <lista_parametara>
                        class Identifikator
```

- **B) vrednosti** (ovakvi parametri ne mogu da budu floating point tipa niti korisnički definisanog tipa – objekti klasa, struktura ili unija.
- **C) šabloni** mogu da budu parametri šablona

Generisanje funkcija iz šablona

- Generisanje funkcija iz šablona
  - Implicitno (automatski)
  - Generisanje na zahtev (eksplicitno)
- Eksplicitno generisanje se postiže
  - Navođenjem tipova stvarnih argumenata šablona u deklaraciji funkcije
  - **template** tip ime<stvarni argumenti> (lista_tipova);
  - Primer: `template int max<int>(int x, int y);`

Implicitno i eksplicitno instanciranje šablona funkcije - primer

```
template<typename T>
T AritmetickaSred (T x , T y )
{
  return ( x+y )/2;
}

a = AritmetickaSred(1, 2) ;  // implicitno
a = AritmetickaSred(1.2, 3.4) ; //implicitno
a = AritmetickaSred (3, 5.5) ; // Greška,
  dvosmisleno za kompajler !
a = AritmetickaSred <double>(3, 5.5) ; //
  Eksplicitno instanciranje šablona
```

Vrednosti kao parametri šablona

- Vrednosti kao parametri šablona mogu da budu:
  - A) celobrojnog tipa
  - B) enumeratori
  - C) reference
  - D) pokazivači na objekte
  - E) pokazivači na funkcije
  - F) pokazivači na članove
- **Ovakvi parametri mora da budu poznati u toku faze kompajliranja. Lokalne promenljive ne mogu da budu vrednosni parametri šablona.**
- Kao parametri šablona **ne mogu** da se koriste realni brojevi (od C++20), objekti klasa, struktura i unija, literali stringova(od C++ 20).
- **Nizovi** se kao parametri šablona prenose **kao pokazivači**.
- **Funkcije** se kao parametri šablona prenose **preko pokazivača na funkcije**.

Vrednosti kao parametri šablona-1

```
#include <iostream>
template <class T, T& ref>
class A {
   T &privRef;
public:
   A():privRef(ref){};
   void Print() const { std::cout << privRef << std::endl; }
};

double a = 3; // Korektna definicija (globalno a) koja
   omogućava prenos a kao vrednosnog parametra preko reference
int main(int argc, char* argv[])
{  /* double a = 3; - error C2971: 'A' : template parameter
   'ref' : 'a' : a local variable cannot be used as a non-type
   argument */
   A<double,a> aO;
   aO.Print();                  // štampa 3
   a=4;
   aO.Print();                  // štampa 4
   return 0;
}
```

Vrednosti kao parametri šablona-2/1

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

template <int i> class A {
   int array[i];
public:
   //statički niz na steku
   A(){}
};

int main() {
   A<10> a;
}
```

Vrednosti kao parametri šablona-2/2

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

template <int i> class A {
    int array[i];
public:
    //statički niz na steku
    A(){}
};

int main() {
    A<10> a;
}
```

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

template <int i> class A {
    int *array;
public:
    // niz na heap-u
    A():array(new int[i]){}
   ~A(){delete []array;}
};

int main() {
    A<10> a;
}
```

Vrednosti kao parametri šablona-2/3

```
template <int i> class B
{
   int *anArray;
   int size;

public:

// niz na heap-u
   B():size(i), anArray(new
   int[i]){}

   B<i>& operator=(const
   B<i>&);

   int& operator[](int i){
   return anArray[i];}

   ~B(){delete []anArray;}
};
```

```
template<int i>
B<i>& B<i>::operator=(const
   B<i>& b)
{
    if(this!=&b){
// brisanje starog niza na heapu
      delete [] anArray;
      size=b.size;
      anArray=new int[size];
      for(int i=0;i<size;i++)
        anArray[i]=b.anArray[i];
   }
  return *this;
}
```

Vrednosti kao parametri šablona-2/4

```
int main(int argc, char* argv[])
{
   B<3> b;
   for(int i=0;i<3;i++) b[i]=i;
   B<4> b1;
   b1=b;  // ne kompajlira se - b1 i b su objekti
   različitih tipova

   return 0;
}
```

Poruka kompajlera

error C2679: binary '=' : no operator found which takes a right-hand operand of type 'B<i>' (or there is no acceptable conversion)

Vrednosti kao parametri šablona-2/5

```
int main(int argc, char* argv[])
{
   B<3> b;
   for(int i=0;i<3;i++) b[i]=i;
   B<3> b1;
   b1=b;  // b i b1 su istog tipa
   for(int i=0;i<3;i++) cout<<b1[i];
   return 0;
}
```

Kompajlira se i izvršava se bez problema

Vrednosti kao parametri šablona-3/1

```
// Funkcija  se, kao parametar šablona, prenosi preko pokazivača
double Fun1(int x) { return x*2; }
double Fun2(double x) { return x/2;}

template<typename RezType, typename ArgType, RezType
   (*ptrToFun)(ArgType)>
void Izvrsi(ArgType x){
   RezType a = (*ptrToFun)(x);
}
// EKVIVALENTNO
template<typename RezType, typename ArgType, RezType Fun(ArgType)>
void Izvrsi(ArgType x){
   RezType a = Fun(x);
}
void main(){
   Izvrsi<double, double, Fun2>(4);
   Izvrsi<double, int, Fun1>(5);
}
```

Vrednosti kao parametri šablona-3/2

```
// Funkcija članica klase se, kao parametar šablona,
   prenosi preko pokazivača na funkciju članicu


class A {
   int x;
   double dx;
public:
   A(int y = 0):x(y){}
   int VrednostX() const { return x;}
   double VrednostDx() const { return dx;}
   void PostaviX(int y) { x = y;}
   void PostaviDx(double dy) { x = dy;}
   int PostaviIVratiX(int y) { x = y; return x;}
};
```

Vrednosti kao parametri šablona-3/3

```
template <typename Tip, typename RezType, typename ArgType,
   RezType (Tip::*ptrToMetod)(ArgType)>
void Pozovi(Tip &object, ArgType x){
     (object.*ptrToMetod)(x);
}

void main(){
   A a;
   void (A::*ptr)(int) = &A::PostaviX; // Definisanje
   pokazivača na funkciju članicu klase A
   (a.*ptr)(2); // Poziv članice nad objektom klase a preko
   pokazivača   a.x = 2;

   Pozovi<A, void, int, &A::PostaviX>(a,4);  // a.x = 4;

   Pozovi<A, int, int, &A::PostaviIVratiX>(a,5);  // a.x = 5;
}
```

Primer ograničavanja tipa parametara šablona

```
template <typename T> struct Ograniceni { };
template <> struct Ograniceni<float> { typedef float Tip; };
template <> struct Ograniceni<double> { typedef double Tip; };

// definišemo šablonsku funkciju Rastojanje sa ograničenim
   tipom
template <typename T>
typename Ograniceni<T>::Tip // Tip rezultata funkcije
   Rastojanje
Rastojanje(T a1, T a2, T b1, T b2)
{
   T tmp1 = a1 - b1;
   T tmp2 = a2 - b2;
   return sqrt( tmp1*tmp1 + tmp2*tmp2 );
}
```

Primer ograničavanja tipa parametara šablona

```
template< typename T> struct Ograniceni{};
template<> struct Ograniceni<short> { typedef short Tip;};
template<typename T> T Fun(T a) { return a<<2;}
// Dozvoljava izvršavanje i nad double tipom parametara

template<typename T>
typename Ograniceni<T>::Tip FunO( typename
   Ograniceni<T>::Tip a)
{
   return a << 2;
}
// NAPOMENA: PARAMETAR SABLONA NIJE PARAMETAR FUNKCIJE PA
   KOMPAJLER NE MOŽE IMPLICITNO DA ZAKLJUČI KOJU FUNKCIJU
   DA INSTANCIRA. FunO obavezno mora da se instancira
   eksplicitno
std::cout<< FunO<short>(4);
```

Podrazumevane vrednosti parametara šablona – 1/1

- Kao i parametri funkcija i parametri šablona mogu da imaju podrazumevane vrednosti.

DEKLARACIJA PARAMETARA TIPOVA

```
class Identifikator [=ImeTipa],
typename Identifikator [=ImeTipa],
template <lista_parametara>
        class Identifikator [=ImeSablona]
```

DEKLARACIJA VREDNOSNIH PARAMETARA

```
ImeTipa imeParametra [=Vrednost]
```

Podrazumevane vrednosti parametara šablona – 1/2

```
template<class T = char, int i=100>
class TestClass{
   T bafer[i];
public:
   T& operator [](int);
   // Destruktor nije potreban, bafer je statički niz na
   koji se ne primenjuje operator delete
};

template<class T, int i>
T& TestClass<T,i>::operator[](int j) {return bafer[j];}

int  main()
{
   TestClass<> t;
   t[0]='A';
   return 0;
}
```

Specijalizacija

- Za neke specifične vrednosti argumenata šablona se specifično definišu generička funkcija/klasa
- Specijalizacija postojećeg šablona za neke kombinacije njegovih argumenata
- Specijalizacija generičke klase
  - Deklaracija: template<*parametri*>class K<*argumenti*>;
  - Definicija: template<*parametri*>class K<*argumenti*> { ... }
  - K – generička specijalizovana klasa
  - *parametri* - parametri specijalizovanog šablona
    - Svi ili samo neki parametri opšteg šablona
    - Ne mogu imati podrazumavane vrednosti
  - *argumenti* – određuju konkretnu verziju opšteg šablona
    - Fiksiraju neke/sve vrednosti parametara opšteg šablona

Vrste specijalizacije šablona

- Vrste specijalizacije šablona
  - Delimična – ako specijalizovani šablon ima bar jedan parametar – tada može da se generiše više klasa iz specijalizovanog šablona
  - Potpuna – ako specijalizovani šablon nema ni jedan parametar – tada na osnovu šablona može da se generiše samo jedna klasa
- **Specijalizacija je moguća samo ako je prethodno navedena deklaracija/definicija opšte generičke klase**
- Specijalizacija određenim parametrima je moguća – samo pre kreiranja bilo koje klase na osnovu opšteg šablona sa istim parametrima

Primer

- **template**< **typename** T, **int** k> **class** Kon; // opšti šablon
- **template**< **typename** T, **int** k> **class** Kon<T*, k>; // spec1
- **template**< **typename** T> **class** Kon<T, 100>; // spec2
- **template**< **int** k> **class** Kon<**int**, k>; // spec3
- **template**< > **class** Kon<void*, 100>; // spec4
- Kon<int*, 15> p1; // koristi se spec1
- Kon<float, 100> p2; // koristi se spec2
- Kon<int, 25> p3; // koristi se spec3
- Kon<void*, 100> p4; // koristi se spec4

Biranje šablona

- Pri generisanju se navodi onoliko elemenata koliko ima opšti šablon
- Prevodilac bira šablon:
  - Najviše specijalizovani (sa najmanje parametara), pa manje specijalizovani, itd.
  - Opšti šablon (ako ni jedan specijalizovani ne odgovara)
  - Ako postoji nejednoznačnost (više podjednako specijalizovanih šablona odgovara) – javlja se greška
  - Primer:

```
Kon<int, 10>  p5 // spec 3
```

Specijalizacija generičkih funkcija

- Za generičke funkcije je moguća **samo potpuna specijalizacija**
  - Deklaracija: **template**<> tip GSfunkcija<argument> (...) ;
  - Definicija: **template**<> tip GSfunkcija<argument> (...) { ...}
- Argumenti (uključujući i < >) mogu da se izostave – ako se oni mogu odrediti na osnovu tipova argumenata funkcije
- **Primer:** umesto posebne definicije, koristi se specijalizacija, da se izbegne poređenje pokazivača
  - **template**<>**char*** max<**char***>(**char*** x, **char*** y); // ili
  - **template**<>**char*** max(**char*** x, **char*** y);
  - char* m=max<char*>(“ime1”, “ime2”); // ili
  - char* n=max(“ime1”, “ime2”);

    - poziva definisanu (negeneričku) funkciju max ako takva postoji

Nasledjivanje šablona (I)

Prosleđivanje izvedene klase kao parametra šablona osnovne klase.

Primer: **Praćenje broja živih objekata klase(jednostavno inkrementiranje statičke promenljive prilikom kreiranja, odnosno dekrementiranje u konstruktoru). Problem je što to treba ponoviti za svaku klasu. Umesto takvog pristupa predlažemo sledeće rešenje.**

```
template <typename CountedType>
class ObjectCounter {
  private:
   static unsigned count;  // broj postojećih objekata
  protected:
   // podrazumevani konstruktor
   ObjectCounter() {++ObjectCounter<CountedType>::count;}
…
```

Nasledjivanje šablona (II)

```
…
protected:
 // Konstruktor kopije
   ObjectCounter (ObjectCounter<CountedType> const&) {
     ++ObjectCounter<CountedType>::count;
   }
public:
   // funkcija članica vraća broj postojećih objekata
   static unsigned live() {
     return ObjectCounter<CountedType>::count;
   }
 };

 // Inicijalizacija brojača (statičke promenljive)
 template <typename CountedType>
 unsigned ObjectCounter<CountedType>::count = 0;
```

Nasledjivanje šablona (III)

```
// nasledjivanje Counter klase šablona
#include "objectcounter.hpp"
#include <iostream>
template <typename CharType>
class CountedString : public ObjectCounter<MyString<CharType>>
{ ... };

int main()
{
  CountedString<char> s1, s2;
  CountedString<wchar_t> ws;
  printf("Broj objekata klase CountedString<char>: %d\n",
        CountedString<char>::live()); //Poziv statičke funkcije
  printf("Broj objekata klase MyString<wchar_t>: %d\n",
        ws.live()); // Još jedan dozvoljen način poziva statičke
   funkcije
}
```

Šabloni kao parametri šablona - 1

```
template< typename T>
struct Cvor {
   T info;
   Cvor<T> *sled;
   Cvor(T n_info, Cvor<T> *n_sled = 0):info(n_info),
   sled(n_sled){}
};
template<typename T>
class LancanaLista {
   Cvor<T> *glava;
public:
   bool JePrazna() { return (glava == 0); }
   LancanaLista():glava(0){};
   // dodavanje na početak liste
  void Dodaj(T info) {  glava = new Cvor<T>(info, glava); }

...
```

Šabloni kao parametri šablona - 2

```
    ...
    T Uzmi() // uzimanje sa početka liste
    {
        T tmp = glava->info;
        Cvor<T> *p = glava;
        glava = glava->sled;
        delete p;
        return tmp;
    }
    ~LancanaLista()
    {
        while(glava) {
                Cvor<T> *p = glava;
                glava = glava->sled;
                delete p;
        }
    }
};
```

Šabloni kao parametri šablona - 3

```
template<typename T>
class Niz {
   T *ptr;
   int topIdx;
   int trenVel;
   static const int DFLT_SIZE = 2;
   void Realociraj(){
        T *p = new T[trenVel << 1];
        for(int i = 0; i < trenVel; i++) p[i]= ptr[i];
        trenVel <<= 1;
        delete [] ptr;
        ptr = p;
   }
public:
...
```

Šabloni kao parametri šablona - 4

```
public:
   Niz():ptr(new T[DFLT_SIZE]),topIdx(0),
   trenVel(DFLT_SIZE){}

   void Dodaj(T info)       {
        if(topIdx == trenVel) Realociraj();
        ptr[topIdx++] = info;
   }

   T Uzmi()    {
        return ptr[--topIdx];
   }
   ~Niz() { delete [] ptr; }
};
```

Šabloni kao parametri šablona - 5

```
// AGREGACIJA
template<typename T, template<typename> class Memorija >
class StekAgr {
   Memorija<T> mem;
public:
   void Push(T info){ mem.Dodaj(info);}
   T Pop() { return mem.Uzmi(); }
};
// Instanciranje šablona
StekAgr<int,Niz> s;
s1.Push(3);
//----------------------
StekAgr<double, LancanaLista> s2;
s2.Push(4.2);
```

Šabloni kao parametri šablona - 6

```
// NASLEDJIVANJE ŠABLONA
template< typename T, template<typename> class Memorija>
class StekNas: private Memorija<T>
{
public:
   void Push(T info){ Memorija<T>::Dodaj(info);}
   T Pop() { return Memorija<T>::Uzmi(); }
};

// Instanciranje šablona
StekNas<int,Niz> s;
s1.Push(3);
//----------------------
StekNas<double, LancanaLista> s2;
s2.Push(4.2);
```

Šabloni: primer sa matricama -1

```
class Matrix {  // STANDARDAN pristup implementaciji matrice
public:
   Matrix ();
   Matrix (int in_rows, int in_cols);
   Matrix(const Matrix& X);
   const Matrix& operator=(const Matrix& X);
   const Matrix operator+(const Matrix& A, const Matrix& B);
...
Private:
   int rows;
   int cols;
   double* data;
};
...
}
```

Šabloni: primer sa matricama - 2

```
// PREKLAPANJE OPERATORA SABIRANJA MATRICA
const Matrix operator+(const Matrix& A, const Matrix& B) {
// ... Proveri da li A i B imaju iste dimenzije
   if(A.rows == B.rows && A.cols == B.cols){
        Matrix X(A.rows, A.cols);
        for(int i=0; i < A.rows * A.cols; ++i){
               X.data[i] = A.data[i] + B.data[i];
        }
        return X;
   }
   else return Matrix();
}
// ANALIZA EFEKTA PRIMENE OPERATORA SABIRANJA
Matrix X;
...
X = A + B;
// A+B kreira privremeni objekat (veličine cele matrice zbira)
// PRIMENOM OPERATORA PRIDRUŽIVANJA TAJ OBJEKAT SE KOPIRA U X
// DVA PUTA VIŠE UTROŠENE MEMORIJE I SKORO DUPLO SPORIJE NEGO
   SABIRANJE MATRICA U C-u.
```

Šabloni: primer sa matricama - 3

```
// PRIMENA META PROGRAMIRANJA
template<typename T> class Matrica;
template<typename T> class Zbir{
public:
   const Matrica<T>& A;
   const Matrica<T>& B;
   Zbir(const Matrica<T>& in_A, const Matrica<T>& in_B):
   A(in_A), B(in_B){}
};
template<typename T> class Razlika{
public:
   const Matrica<T>& A;
   const Matrica<T>& B;
   Razlika(const Matrica<T>& n_A, const Matrica<T>& n_B):
   A(n_A), B(n_B){}
};
```

Šabloni: primer sa matricama - 4

```
// Šablonske operatorske funkcije

template<typename T>
const Zbir<T> operator+(const Matrica<T>& A, const Matrica<T>& B)
{
   return Zbir<T>(A,B);
}

template<typename T>
const Razlika<T> operator-(const Matrica<T>& A, const Matrica<T>&B)
{
   return Razlika<T>(A,B);
}
```

Šabloni: primer sa matricama - 5

```
template<typename T> class Matrica {
private:
   int brRedova;
   int brKolona;
   T* data;
public:
   Matrica():data(0),brRedova(0),brKolona(0){}
   Matrica(int n_brRedova, int n_brKolona):data(new T[n_brRedova *
   n_brKolona]),brRedova(n_brRedova), brKolona(n_brKolona){}
private:
   void Saberi(T *rez, T *ptr1, T *ptr2, int dim){
        for(int i = 0; i < dim; i++)
                *rez++ = *ptr1++ + *ptr2++;
   }
   void Oduzmi(T *rez, T *ptr1, T *ptr2, int dim){
        for(int i = 0; i < dim; i++)
                *rez++ = *ptr1++ - *ptr2++;
   }
```

Šabloni: primer sa matricama - 6

```
public:
  Matrica(const Zbir<T>& Z)  {
    if((Z.A.brRedova == Z.B.brRedova) && (Z.A.brKolona == Z.B.brKolona)) {
              brRedova = Z.A.brRedova;
              brKolona = Z.B.brKolona;
              data = new T[brRedova * brKolona];
              Saberi(data, Z.A.data, Z.B.data, brRedova * brKolona);
    }
  }
  const Matrica& operator=(const Zbir<T>& Z){
    if((Z.A.brRedova == Z.B.brRedova) && (Z.A.brKolona == Z.B.brKolona)) {
              delete [] data;
              brRedova = Z.A.brRedova;
              brKolona = Z.B.brKolona;
              data = new T[brRedova * brKolona];
              Saberi(data, Z.A.data, Z.B.data, brRedova * brKolona);
    }
    return *this;
  }
```

Šabloni: primer sa matricama - 7

```
public:
   Matrica(const Razlika<T>& Z) {
     if((Z.A.brRedova == Z.B.brRedova) && (Z.A.brKolona == Z.B.brKolona)){
               brRedova = Z.A.brRedova;
               brKolona = Z.B.brKolona;
               data = new T[brRedova * brKolona];
               Oduzmi(data, Z.A.data, Z.B.data, brRedova * brKolona);
     }
   }
   const Matrica& operator=(const Razlika<T>& Z) {
     if((Z.A.brRedova == Z.B.brRedova) && (Z.A.brKolona == Z.B.brKolona)){
               delete [] data;
               brRedova = Z.A.brRedova;
               brKolona = Z.B.brKolona;
               data = new T[brRedova * brKolona];
               Oduzmi(data, Z.A.data, Z.B.data, brRedova * brKolona);
     }
     return *this;
   }
```

Šabloni: primer sa matricama - 8

```
// Upotreba kreiranog šablona
Matrica <int> B(100,100), C(100,100);

// Popunjavanje elemenata matrica B i C
Matrica <int> A;
A = B + C;

// OPERATOR + kreira privremeni objekat ZBIR koji
   sadrži reference na B i C .
// PREKLOPLJENI OPERATOR = dobija objekat ZBIR i nad
   referencama koje se nalaze unutar ovog objekta izvodi
   sabiranje i smeštanje u predvidjeni prostor.
// EFIKASNOST ekvivalentna C kodu
```