Napredno programiranje i programski jezici

05 C++ (generičke)

Fakultet tehničkih nauka, Novi Sad
23-24/Z
Dunja Vrbaški

```
int saberi(int a, int b) {
    return a + b;
}

double saberi(double a, double b) {
    return a + b;
}

int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    return 0;
}
```

```
int saberi(int a, int b) {
    return a + b;
}

double saberi(double a, double b) {
    return a + b;
}




void ispis(int a, int b) {
    cout << "INT" << endl;
    cout << "*********" << endl;
    cout << a << " " << b << endl;
}

void ispis(double a, double b) {
    cout << "a = " << a << ", " << "b = " << b << endl;
}
```

U čemu je razlika između ova dva para funkcija?

Kada je logika drugačija za različite parametre → preklapanje (function overloading)

Kada je logika identična → šablon (template)

Cilj je napisati funkciju koja isto radi bez obzira koji tipovi su u pitanju.
Nekako izbeći navođenje tipova (int, double, Complex…).

```
... saberi(... a, ... b) {
    return a + b;
}
```

```
... saberi(... a, ... b) {
    return a + b;
}
```

```
...
T saberi(T a, T b) {
    return a + b;
}
```

```
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    return 0;
}
```

```
... saberi(... a, ... b) {
    return a + b;
}
```

```
...
T saberi(T a, T b) {
    return a + b;
}
```

```
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    return 0;
}
```

```
... saberi(... a, ... b) {
    return a + b;
}
```

```
...
T saberi(T a, T b) {
    return a + b;
}
```

```
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi<int>(x, y) << endl;
    cout << saberi<double>(p, q) << endl;

    return 0;
}
```

Može se izostaviti ako je kompajler u stanju da odredi odgovarajuće

```cpp
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```cpp
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    return 0;
}
```

template definition
definicija

template specialization
konkretizacija

→ definisana je čitava familija preklopljenih funkcija
(isti broj parametara, različiti tipovi)

```cpp
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```cpp
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    return 0;
}
```

Šta pretpostavljamo?

```
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```
class A {
private:
    int x;
public:
    ...
};
```

```
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    A a1, a2, a3;
    a3 = saberi(a1, a2);

    return 0;
}
```

```cpp
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```cpp
class A {
private:
    int x;
public:
    A(int xx = 1) : x(xx) { }
    int getX() const { return x;}
    friend A operator+(const A &, const A&);
};

A operator+(const A &a1, const A &a2) {
    A temp;
    temp.x = a1.x + a2.x;
    return temp;
}
```

```cpp
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;

    cout << saberi(x, y) << endl;
    cout << saberi(p, q) << endl;

    A a1, a2, a3;
    a3 = saberi(a1, a2);
    cout << a3.getX() << endl;

    return 0;
}
```

```
template<class T>
T saberi(T a, T b) {
    return a + b;
}
```

```
template<int A>
void ponavljaj() {
    for(int i = 0; i < A; i++) {
        cout << i << " ";
    }
}
```

```
int main()
{
int main()
{
    int x = 3, y = 5;
    double p = 1.2, q = 3.4;
    cout << saberi<int>(x, y) << endl;
    cout << saberi<double>(p, q) << endl;

    A a1, a2, a3;
    a3 = saberi<A>(a1, a2);
    cout << a3.getX() << endl;

    ponavljaj<5>();
}
```

ZADATAK

Realizovati klasu koja modeluje uređeni par dve vrednosti koje ne moraju biti istog tipa.

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

```
template <class T1, class T2>
class Pair {
private:
        T1 first;
        T2 second;
public:
    Pair(const T1&, const T2&);

    T1 getFirst() const;
    T2 getSecond() const;

    void setFirst(const T1&);
    void setSecond(const T2&);

    Pair& operator=(const Pair&);

    void printPair() const;
};
```

```
implementacija...
```

```
template <class T1, class T2>
class Pair {
private:
        T1 first;
        T2 second;
public:
    Pair(const T1&, const T2&);

    T1 getFirst() const;
    T2 getSecond() const;

    void setFirst(const T1&);
    void setSecond(const T2&);

    Pair& operator=(const Pair&);

    void printPair() const;
};
```

```
implementacija...
```

```
int main()
{
    Pair<int,int> x1(1,2);
    x1.printPair();
}
```

```
template <class T1, class T2>
class Pair {
private:
        T1 first;
        T2 second;
public:
    Pair(const T1&, const T2&);

    T1 getFirst() const;
    T2 getSecond() const;

    void setFirst(const T1&);
    void setSecond(const T2&);

    Pair& operator=(const Pair&);

    void printPair() const;
};
```

```
implementacija...
```

```
typedef Pair<int,int> IIPair;
typedef Pair<int,double> IDPair;
typedef Pair<double,double> DDPair;

int main()
{
    Pair<int,int> x1(1,2);
    x1.printPair();

    IDPair x2(1,2.3), x3(5,7.8);
    x2.printPair();
    x3.printPair();

}
```

```cpp
template <class T1, class T2>
class Pair {
private:
        T1 first;
        T2 second;
public:
    Pair(const T1&, const T2&);

    T1 getFirst() const;
    T2 getSecond() const;

    void setFirst(const T1&);
    void setSecond(const T2&);

    Pair& operator=(const Pair&);

    void printPair() const;
};
```

```
implementacija...
```

```cpp
typedef Pair<int,int> IIPair;
typedef Pair<int,double> IDPair;
typedef Pair<double,double> DDPair;

int main()
{
    Pair<int,int> x1(1,2);
    x1.printPair();

    IDPair x2(1,2.3), x3(5,7.8);
    x2.printPair();
    x3.printPair();

    x3 = x2;
    if(x2 == x3)
        cout<<"x2 i x3 su jednaki"<<endl;
    else
        cout<<"x2 i x3 nisu jednaki"<<endl;
}
```

```
template<class T1, class T2>
Pair<T1,T2>::Pair(const T1 &f, const T2 &s) : first(f), second(s) { }
```

```cpp
template<class T1, class T2>
T1 Pair<T1,T2>::getFirst() const {
     return first;
}

template<class T1, class T2>
T2 Pair<T1,T2>::getSecond() const {
     return second;
}
```

```cpp
template<class T1, class T2>
void Pair<T1,T2>::setFirst(const T1 &f) {
     first = f;
}

template<class T1, class T2>
void Pair<T1,T2>::setSecond(const T2 &s) {
     second = s;
}
```

```
template<class T1, class T2>
Pair<T1,T2>& Pair<T1,T2>::operator=(const Pair<T1,T2> &p) {
    first = p.first;
    second = p.second;
    return *this;
}
```

```
template<class T1, class T2>
bool operator==(const Pair<T1,T2> &p1, const Pair<T1,T2> &p2) {
    if((p1.getFirst()==p2.getFirst()) &&
        (p1.getSecond()==p2.getSecond()))
        return true;
    else
        return false;
}
```

```
template<class T1, class T2>
bool operator==(const Pair<T1,T2> &p1, const Pair<T1,T2> &p2) {
    if((p1.getFirst()==p2.getFirst()) &&
        (p1.getSecond()==p2.getSecond()))
        return true;
    else
        return false;
}
```

```
template<class T1, class T2>
Pair<T1,T2>& Pair<T1,T2>::operator=(const Pair<T1,T2> &p) {
    first = p.first;
    second = p.second;
    return *this;
}
```

```
template<class T1, class T2>
void Pair<T1,T2>::printPair() const {
    cout << "( " << first << ", " << second << " )" << endl;
}
```

ZADATAK

Realizovati klasu koja modeluje uređenu kolekciju elemenata istog tipa i proizvoljne, zadate dužine.

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

```
template <class T, int D>
class Niz {
private:
    T el[D];
    int brEl;
public:
    Niz() { brEl=0; }
    ~Niz() {}

    int getBrEl() const { return brEl; }

    T operator[](int i) const { return el[i]; }
    T& operator[](int i) { return el[i]; }

    Niz<T,D>& operator=(const Niz<T,D>&);

    void printNiz() const;
    bool insertNiz(const T&);
};
```

```
implementacija...
```

```
template <class T, int D>
class Niz {
private:
    T el[D];
    int brEl;
public:
    Niz() { brEl = 0; }
    ~Niz() {}

    int getBrEl() const { return brEl; }

    T operator[](int i) const { return el[i]; }
    T& operator[](int i) { return el[i]; }

    Niz<T,D>& operator=(const Niz<T,D>&);

    void printNiz() const;
    bool insertNiz(const T&);
};
```

```
int main()
{
    Niz<int, 10> iNiz1, iNiz2;
    Niz<int, 5> iNiz3;

    iNiz1.insertNiz(1);
    ...

    iNiz2 = iNiz1;
    if(iNiz1 == iNiz2)
        cout << ...;
    else
        cout << ...;
}
```

```cpp
template <class T, int D>
Niz<T,D>& Niz<T,D>::operator=(const Niz<T,D> &rn) {
    for(brEl = 0; brEl < rn.brEl; brEl++)
        el[brEl] = rn[brEl];
    return *this;
}
```

```
template <class T, int D>
void Niz<T,D>::printNiz() const {
    cout << "( ";
    for(int i = 0; i < brEl - 1; i++)
        cout << el[i] << ", ";
    cout << el[brEl-1] << " )" << endl;
}
```

```cpp
template <class T, int D>
bool operator==(const Niz<T,D> &rn1, const Niz<T,D> &rn2) {
    if(rn1.getBrEl() != rn2.getBrEl())
        return false;
    for(int i = 0; i < rn1.getBrEl(); i++)
        if(rn1[i] != rn2[i])
            return false;
    return true;
}
```

```
template <class T, int D>
class Niz {
private:
    T el[D];
    int brEl;
public:
    Niz() { brEl = 0; }
    ~Niz() {}

    int getBrEl() const { return brEl; }

    T operator[](int i) const { return el[i]; }
    T& operator[](int i) { return el[i]; }

    Niz<T,D>& operator=(const Niz<T,D>&);

    void printNiz() const;
    bool insertNiz(const T&);
};
```

```
int main()
{
    Niz<int, 10> iNiz1, iNiz2;
    Niz<int, 5> iNiz3;

    iNiz1.insertNiz(1);
    ...

    iNiz2 = iNiz1;
    if(iNiz1 == iNiz2)
        cout << ...;
    else
        cout << ...;
}
```

ZADATAK

(neobavezno)

Napisati template funkciju swap koja menja vrednost dvema zadatim promenljivama.

Da li su neophodni operatori dodele za klase Pair i Niz?

Da li je u navedenim implementacijama potrebna provera (if) da li se radi o dodeli objekta samom sebi (p = p;)?

Proglasiti operator== kao friend funkciju u klasama Pair i NIz.