

Šabloni. Parametriskais polimorfisms (*generic programming, vispārinātā programmēšana*)

Vērtību apmaiņas funkcija ar *norādēm*

```
template<class T>
void Swap(T& A, T& B) {
    T C;

    C = A;
    A = B;
    B = C;
}
```

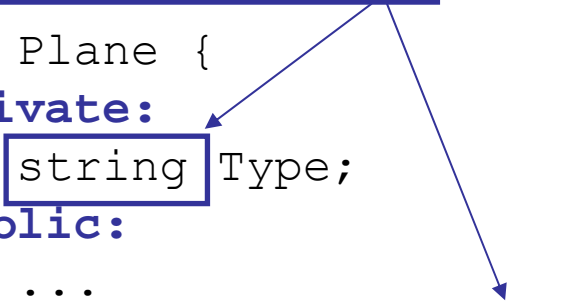
Mainīgie un objekti

```
int    iX = 2,    iY = 3;
float  fX = 2.5,  fY = 3.5;
Plane P1("Boeing - 747"), P2("Il - 96");
```

Klases *Plane* deklarācijas fragments

```
#include <cstring.h>
```

```
class Plane {  
    private:  
        string Type;  
    public:  
        ...  
        Plane(const string& pType) : Type(pType) {  
        }  
        void PrintInfo() {  
            cout << "Type: " << Type;  
        }  
};
```



string - teksta rinda (klase)

cstring.h - header-file

Funkcijas izsaukumi

```
Swap(iX, iY); //iX ↔ iY
```

```
Swap(fX, fY); //fX ↔ fY
```

```
Swap(P1, P2); //P1 ↔ P2
```

Rezultāti

```
X: 2, Y: 3.
```

```
X: 3, Y: 2.
```

```
X: 2.5, Y: 3.5.
```

```
X: 3.5, Y: 2.5.
```

```
Type: Boeing - 747, Type: Il - 96.
```

```
Type: Il - 96, Type: Boeing - 747.
```

Vērtību apmaiņas funkcija ar *rādītājiem*

```
template<class T>
void Swap(T* A, T* B) {
    T C;

    C = *A;
    *A = *B;
    *B = C;
}
```

Funkcijas izsaukumi

```
Swap(&iX, &iY); //iX ↔ iY
Swap(&fX, &fY); //fX ↔ fY
Swap(&P1, &P2); //P1 ↔ P2
```

Šablona *CoordPoint* deklarācijas fragments

```
template <class T>
class CoordPoint {
    protected:
        T X;
        T Y;
    public:
        ...
        CoordPoint(T, T);
        T GetX() const {
            return X;
        }
        void SetX(T X) {
            this->X = X;
        }
        T GetY() const;
        void SetY(T);
        virtual void Print() const;
};
```

The diagram illustrates the relationship between the template parameter `T` and the member variables `X` and `Y`. A vertical dashed line separates the code from the diagram. To the right of the line, the word `int` is crossed out with a large red 'X'. An arrow points from the crossed-out `int` down to the letter `T`, indicating that the type `T` is used for the member variables instead of a fixed type like `int`.

Metožu realizācijas piemēri

```
template <class T>
inline T CoordPoint<T>::GetY() const {
    return Y;
}
```

```
template <class T>
inline void DisplayPoint<T>::Print() const {
    CoordPoint<T>::Print();
    cout << ", Color = " << Color;
}
```

Šablonu mantošana

```
template <class T>
class DisplayPoint : public CoordPoint<T> {
```

Klašu un objektu izveidošana

```
DisplayBrokenLine<int> *IntLine = new  
    DisplayBrokenLine<int>(2, 1);  
DisplayBrokenLine<long> *LongLine = new  
    DisplayBrokenLine<long>(2, 1);
```

```
DisplayPoint<int> *IntD1 = new DisplayPoint<int>  
    (10, 11, 12);  
DisplayPoint<long> *LongD1 = new DisplayPoint<long>  
    (10L, 11L, 12);
```

DisplayBrokenLine	- šablons
DisplayBrokenLine<int>	- klase
*IntLine	- objekts

Darbs ar objektiem

```
IntLine->Print();  
LongLine->Print();
```

STL bibliotēkas izmantošanas piemērs (šablons *vector*)

```
#include <vector>
```

```
#include <string>
```

```
using namespace std;
```

```
...
```

```
const N = 3;
```

```
int I[N] = {1, -2, 3};
```

```
vector<int> IntVector;
```

```
vector<Plane> PlaneVector;
```

```
for(int i=0; i<N; i++)
```

```
    IntVector.push_back(I[i]);
```

```
PlaneVector.push_back(Plane("Boeing-747"));
```

```
PlaneVector.push_back(Plane("Il-96"));
```

```
for(int i=0; i<IntVector.size(); i++)
```

```
    cout << IntVector.at(i) << " ";    // 1 -2 3
```

```
cout << endl;
```


STL bibliotēkas izmantošanas piemērs (šablons *deque*)

```
#include <deque>
...
using namespace std;
...
deque<int> D;
D.push_front(1);
D.push_back(2);
for(int i=0; i<D.size(); i++) {
    cout << D[i] << " "; //1 2
}
cout << endl;

cout << "Deque is empty?" << D.empty() << endl; //0
D.pop_front();
D.pop_front();
cout << "Deque is empty?" << D.empty() << endl; //1
```