

Objektorientētā programmēšana

4. laboratorijas darbs

Šabloni. Parametriskais polimorfisms.

Dr. sc. ing. Pāvels Rusakovs

Mg. sc. ing. Vitālijs Zabiņako

Mg. sc. ing. Andrejs Jeršovs

Mg. sc. ing. Pāvels Semenčuks

Mg. sc. ing. Vladislavs Nazaruks

RTU 2012

Šabloni. Parametriskais polimorfisms.

(*generic programming, vispārinātā programmēšana*)

Vērtību apmaiņas funkcija ar *norādēm*

Uzdevums: uzrakstīt *universālo* funkciju, kas strādā ar *jebkura tipa (klases)* datiem.

```
template<class T>           // šablona "vizītkarte"  
void Swap (T& A, T& B) {  
    T 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 (C++ 4.5)

```
#include <cstring.h>
...
class Plane {
    private:
        string Type;
    public:
        ...
        Plane(const string& pType) : Type(pType) {
        }
        void Print() {
            cout << "Type: " << Type;
        }
};
```

Piezīme: lai iepriekš apskatītajā funkcijā `Swap (. . .)` ir tikai *deklarācija bez piešķires*:
`T C;`

Tad klasē obligāti jābūt *konstruktors pēc noklusējuma*.

Līdzīga situācija ir STL (*Standard Template Library*) bibliotēkas lietošanas gadījumā.

Funkcijas izsaukumi (*bez* parametru adresēm)

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

Funkcijas izsaukumi (ar *parametru adresēm*)

```
Swap(&iX, &iY); // iX ↔ iY
```

```
Swap(&fX, &fY); // fX ↔ fY
```

```
Swap(&P1, &P2); // P1 ↔ P2
```

Rezultāti sakrīt ar iepriekšējiem rezultātiem.

“Universālais” koordinātu punkts *CoordPoint*

```
int X;    // CoordPointInt ?
int Y;

long X;   // CoordPointLong ?
long Y;

float X;  // CoordPointFLoat ?
float Y;

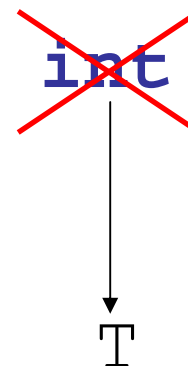
char X;   // CoordPointChar ?
char Y;

int X;    // CoordPointIntLong ?
long Y;
```

```
...      // ??????
```

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;
};
```



Šablonā var būt *ne tikai* noskaņojamie parametri.

Piemēram, punktam var būt *trešais* atribūts ar *konkrēto* datu tipu:
int Id;

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 radīšana

```
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<T>	- šablons
DisplayBrokenLine<int>, DisplayBrokenLine<long>	- klases
*IntLine, *LongLine	- objekti

Darbs ar objektiem

```
IntLine->Print();    // "parastais" stils  
LongLine->Print();   // "parastais" stils
```

Vairāku tipu nodošana vienam šablonam

```
template <class T1, class T2>
class CoordPoint {
    protected:
        T1 X;
        T2 Y;
        ...
};
```

```
template <class T1, class T2>
class DisplayPoint : public CoordPoint<T1, T2> {
    ...
};
```

```
template <class T1, class T2>
CoordPoint<T1, T2>::CoordPoint() : X(0), Y(0) {
}
```

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

STL (Standard Template Library) *bibliotēka*.

C++ 5.0, 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 (unsigned int i=0; i<N; i++)
```

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

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

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

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

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

```
cout << endl;
```

STL bibliotēka: šablons deque

```
#include <deque>
```

```
...
```

```
using namespace std;
```

```
...
```

```
deque<int> D;
```

```
D.push_front(1);
```

```
D.push_back(2);
```

```
for(unsigned 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
```

STL bibliotēka: šablons set

```
#include <set>
```

```
...
```

```
using namespace std;
```

```
typedef set<int, less<int> > IntSet;
```

```
IntSet IS;
```

```
IntSet::const_iterator Res;
```

```
IS.insert(1);IS.insert(2);IS.insert(3); // 1 2 3
```

```
IS.insert(1); // nav efekta
```

```
cout << "Total elements: " << IS.size() << endl; // 3
```

```
Res = IS.find(2);
```

```
cout << ( (Res != IS.end()) ? "OK":"NOT OK") <<  
    "." << endl;
```

```
// OK.
```

```
Res = IS.find(4);
```

```
cout << ( (Res != IS.end()) ? "OK":"NOT OK") <<  
    "." << endl;
```

```
// NOT OK.
```