

Šabloni

- n Uzdevums: realizēt funkciju $\min(x, y)$, kas atgriež minimālo vērtību no divu doto funkcijas parametru vērtībām.
- n Risinājums 1: Katram parametru tipam realizēt savu funkciju. Funkciju nosaukumi ir vienādi – notiek funkcijas pārlāde.
- n Piemēram:

```
int min(int a, int b) { return a < b ? a : b; }  
unsigned min(unsigned a, unsigned b){ return a < b ? a : b; }  
double min(double a, double b) { return a < b ? a : b; }
```

- n Risinājuma trūkumi:

- § Katram tipam jāraksta sava funkcija, kuras realizācija ir identiska
- § Jārealizē funkcija visiem paredzamajiem parametru tiem

Šabloni (turpinājums)

n Uzdevums: realizēt funkciju $\min(x, y)$, kas atgriež minimālo vērtību no divu doto funkcijas parametru vērtībām.

n Risinājums 2: Izmantot preprocesora definīciju.

n Piemēram:

```
#define min(X, Y) ((X) < (Y) ? (X) : (Y))
```

n Risinājuma trūkumi

§ nenotiek tipu kontrole

§ preprocesora definīcijas redzamības diapazonā nedrīkst būt deklarācija ar tādu pašu nosaukumu!

Piemēram, ja būs apraksts

```
int min(int, int);
```

tas tiks aizvietota ar

```
int ((int) < (int) ? (int) : (int));
```

§ grūti atklājami kļūdu iemesli

Šabloni (turpinājums)

- n Uzdevums: realizēt funkciju $\min(x, y)$, kas atgriež minimālo no diviem norādītiem funkcijas parametriem.
- n Risinājums 3: **Izmantot funkcijas šablonu.**
- n Piemēram:

```
template <class T>          // vai template <typename T>
T min(T x, T y) { return x < y ? x : y; }
```

- n Risinājuma priekšrocības:
- n Funkciju $\min()$ var lietot ar jebkuriem vienāda tipa argumentiem ar nosacījumu, ka argumenta tipam T ir definēta salīdzināšanas operācija $<$
- n Funkcijas kods tiek automātiski ģenerēts tikai tiem parametru tiptiem, ar kuriem reāli funkcija tiek izsaukta. Ja funkcija $\min()$ netiek izsaukta nevienu reizi, tad šīs funkcijas kods vispār netiek ģenerēts
- n Katram tipam tiek ģenerēta sava funkcija – notiek funkcijas parametru tipu kontrole
- n Šablona redzamības apgabalā papildus var definēt savas funkcijas, ar tādu pašu deklarāciju kā šablonam (skat. 1. risinājumu)

Šabloni (turpinājums)

n Šablona *min()* izmantošanas piemēri

```
double x;
```

```
// ...
```

```
double y = min(x, 3.14159);
```

```
// ...
```

```
int n;
```

```
// ...
```

```
int m = min(n, 0);
```

```
// ...
```

```
int k = 10; char c = 'A';
```

```
int m = min(k, c); // KĻŪDA! Dažādi parametru tipi
```

```
// ...
```

```
int min(int, int);
```

```
int k = 10; char c = 'A';
```

```
int m = min(k, c); // OK - c tips char tiek pārveidots uz int ('A' == 65)
```

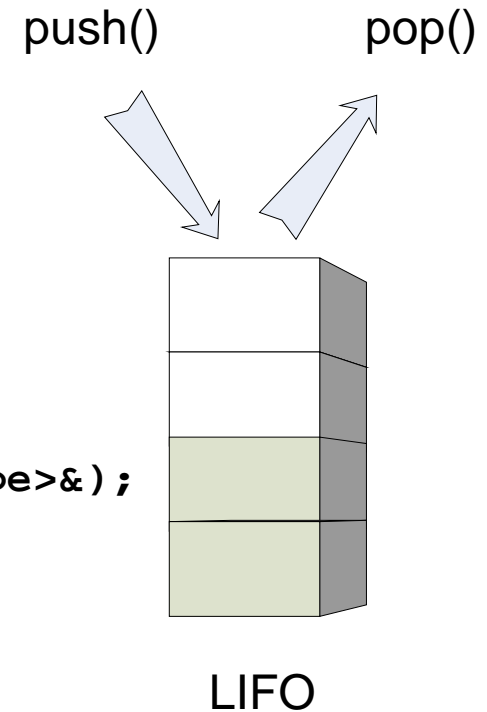
Šabloni (turpinājums)

n Klases šablons

```
template <class ElemType>
class Stack
{   ElemType* stp;
    int max_size;
    int top;
public:
    Stack(int max_size = 10);
    ~Stack() { delete [] stp; }
    Stack(const Stack<ElemType>&);
    Stack<ElemType>& operator=(const Stack<ElemType>&);

    void push(const ElemType newElem);
    ElemType pop(void);
};
```

```
template <class ElemType>
Stack<ElemType>::Stack(int size)
{   top = -1; max_size = size;
    stp = new ElemType[max_size];
}
```



Šabloni (turpinājums)

```
class Vehicle
{   long reg_num;
    int type;
public:
    Vehicle();
    Vehicle (r_n, int t);
    // ...
};

void main()
{   Stack<int> ist(12);   // steks, kurā var glabāt līdz 12 int tipa objektiem
    Stack<Vehicle> vst(20); // steks, kurā var glabāt līdz 20 Vehicle tipa objektiem

    ist.push(100);
    ist.push(200);

    Vehicle v1(111, 1);
    Vehicle v2(222, 2);
    vst.push(v1);
    vst.push(v2);

    Vehicle v3 = vst.pop(); // v3.reg_num == 222; v3.type == 2
}
```

Šabloni (turpinājums)

n Terminoloģija

```
template <class ElemType>  
class Stack  
{  
    // ...  
};
```

Šablons

```
Stack<Vehicle> vst(20);
```

Tips (šablona eksemplārs)

Objekts (tipa eksemplārs)

Šabloni (turpinājums)

n Klases šablona parametri, kas nav tipi

§ Papildus parametriem–tipiem, var norādīt arī parametrus–konstantes, norādot tipu un nosaukumu:

```
template <class ElemType, int s>
class Stack
{
    ElemType stp[s];
    // ...
};
// ...
Stack<Vehicle, 50> carStack;
```

Parametrs s nav tips.
Veidojot klases objektu, šā
parametra vietā jāraksta
konstants lielums.

Šabloni (turpinājums)

n Klases Vector2 šablons, izmantojot parametru-konstanti

```
template <class T, int n>
class Vector2
{
    T data[n];
public:
    T& operator[] (int i) { return data[i]; }
};

void main()
{
    Vector2<int, 5> x;

    for (int i = 0; i < 5; ++i)    x[i] = i;

    Vector2<int, 10> y;
    y[3] = x[4];

    // ...
}
```

Šabloni (turpinājums)

- n Šablona parametriem iespējams norādīt noklusētās vērtības:

```
template <class Type = int, int s = 5>
class Array
{
    Type elements[s];
    // ...
};

// ...
Array<double, 10> x; // 10 elementu double masīvs
Array<long>       y; // 5 elementu long masīvs
Array<>           z; // 5 elementu int masīvs
```

Šabloni (turpinājums)

n Viegļākai koda uztverei šablonu eksemplāriem bieži definē pseidonīmus, kurus pēc tam izmanto objektu definēšanai:

```
typedef Array<int> IntArray;  
// ...  
IntArray ia1; // veselu skaitļu masīvs
```

Šabloni (turpinājums)

n Klases šabloni mantošanas mehānismā

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

```
template <class T>
CoordPoint<T>::CoordPoint(){X = 0; Y = 0;} // iespējamās problēmas !!!
```

```
template <class T>
CoordPoint<T>::CoordPoint(T PX, T PY) { X = PX; Y = PY;}
```

```
template <class T>
CoordPoint<T>::SetX(T PX) { this->X = PX; }
//...
```

Šabloni (turpinājums)

n Klases šabloni mantošanas mehānismā

```
#include "CoordPoint.h"
```

```
template < class CT>
class DisplayPoint : CoordPoint<CT>
{
    char* color;
public:
    DisplayPoint() : CoordPoint<CT>() {color = "White";}
    DisplayPoint(CT, CT, char*);
    // ...
};
```

```
template < class CT>
DisplayPoint<CT>::DisplayPoint(CT PX, CT PY, char* c)
    : CoordPoint<CT>(PX, PY) { this->color = c; }
// ...
```

Šabloni (turpinājums)

- n Šabloni reprezentē formas, no kurām, izmantojot šablona parametrus, kompilators automātiski ģenerē kodu funkcijām un klasēm
- n Šablonam obligāti ir jābūt vismaz vienam parametram, kuram obligāti jābūt arī lietotam funkcijas/klares definīcijā
- n Pats šablons kodu neģenerē – tas tikai norāda kompilatoram, kāds kods ir jāģenerē
- n Katrai šablona parametru vērtību kombinācijai tiek ģenerēts unikāls funkcijas vai klares kods, kuru var izmantot programmā
- n Parasti šablona deklarāciju un definīciju raksta kopā vienā failā ar paplašinājumu .h, un šo failu iekļauj kompilēšanas vienībās, kur šo šablonu lieto