Šablony

Šablony jsou generické funkce a třídy, ve kterých jsou parametrem datové typy a celočíselné hodnoty.

Šablony funkcí

Zápis šablony:

```
template<..parametry..> definice_funkce
```

Je-li parametrem datový typ, lze pro jeho označení použít klíčová slova:

```
class
typename
```

Významově není mezi oběma klíčovými slovy žádný rozdíl.

Pro jména parametrů, které označují datové typy, se obvykle používají velká písmena (T apod.).

Příklad. Funkce počítající absolutní hodnotu.

```
template<typename T>
inline T AH(T a) { return a>=0 ? a : -a; }

template<class T>
inline T AH(T a) { return a>=0 ? a : -a; }
```

Volání funkce:

```
jméno_funkce<..skutečné_datové_typy..>(..argumenty..)
```

Pokud lze skutečné datové typy odvodit z argumentů volání funkce, lze jejich uvedení ve volání funkce vynechat.

Příklad. Volání funkce z předchozího příkladu.

```
AH<int>(-3)          AH(-3)          // celé číslo má typ int
AH<double>(-3.1)     AH(-3.1)        // desetinné číslo má typ double
AH<float>(-3.1)      AH(-3.1f)       // přípona f znamená typ float
```

Příklad. Šablona funkce pro vyhledání maximálního prvku.

```
template<class T>
T maxPrvek(T a[],int n)          // T..datový typ prvků pole
{
  T v=a[0];                      // maximální prvek
  for (int i=1;i<n;++i) if (a[i]>v) v=a[i];
  return v;
}

float a[] = { 3.5,1,8.4,2.2,5,7 };
maxPrvek(a,6);     // z šablony je vygenerována funkce pro typ float
```

Příklad. Šablona funkce, která má dva parametry datových typů.

```
template<typename T,typename S>
inline bool jeVIntervalu(T x,S a,S b) { return a<=x && x<=b; }

jeVIntervalu<double,int>(-3.1,0,10)

jeVIntervalu<double>(-3.1,0,10)

jeVIntervalu(-3.1,0,10)
```

Vedle parametrů reprezentující datové typy šablona může mít parametry reprezentující celočíselné hodnoty. Patří sem všechny datové typy, které mají celočíselný charakter:

- celočíselné datové typy (`int` a další)
- výčtový typ
- ukazatel
- reference

Příklad. Šablona funkce pro transpozici matice. Vstupní matice má řád $m \times n$, výstupní matice má řád $n \times m$.

```
template<class T, int m, int n>
void transp(const T a[m][n], T b[n][m])
{
  for (int i=0;i<m;++i) for (int j=0;j<n;++j) b[j][i]=a[i][j];
}

float a[2][3]={{7,4,1},{2,5,3}},b[3][2];

transp<float,2,3>(a,b);
```

Šablony tříd

Zápis šablony:

```
template<..parametry..> definice_třídy
```

Deklarace parametru šablony, který označuje typ, může být:

```
class jmeno
typename jmeno
template<..parametry..> class jmeno
```

Šablona třídy může mít implicitní hodnoty parametrů. Platí pro ně obdobné zásady jako pro implicitní hodnoty parametrů funkcí:

- Datový typ implicitní hodnoty musí odpovídat typu parametru.
- Implicitní hodnoty lze uvést od libovolného parametru.
- Při použití šablony lze argumenty od libovolného parametru s implicitní hodnotou vynechat. Použijí se implicitní hodnoty.

Příklad. Sestavíme šablonu datové struktury nazývané mapa. V mapě jsou údaje uloženy jako dvojice klíč + data. Klíč přitom slouží pro vyhledávání. Pro uložení údajů bude použito pole.
Na konci je ukázka, jak lze přetížit operátor << výstupu na *stream* pro uživatelský datový typ.

```
template<class K,class D,unsigned n=10>
class Mapa { struct Prvek { K klic; D data; };
             Prvek pole[n];
             unsigned i=0;
     public: bool pridat(const K &,const D &);
             D *najit(const K &); };

template<class K,class D,unsigned n>
bool Mapa<K,D,n>::pridat(const K &k,const D &d)
{ if (i==n) return false;
  pole[i].klic=k; pole[i++].data=d;
  return true; }

template<class K,class D,unsigned n>
D *Mapa<K,D,n>::najit(const K &k)
{ for (unsigned j=0;j<i;++j)
  { if (pole[j].klic==k) return &pole[j].data; }
  return nullptr; }

struct Zlomek { unsigned c,j;
                Zlomek() { }
                Zlomek(unsigned c,unsigned j):c(c),j(j) { } };

ostream & operator << (ostream &os,const Zlomek *z)
{ cout << z->c << '/' << z->j;
  return os; }

Mapa<double,Zlomek> M;
  M.pridat(.5,Zlomek(1,2));
  M.pridat(.75,Zlomek(3,4));
  M.pridat(1.4,Zlomek(7,5));
  auto z=M.najit(.75);
  if (z!=nullptr) cout << z << endl;
  3/4
```

Dědění šablon

Při dědění šablon můžeme vytvářet šablony se stejným počtem parametrů nebo v případě, kdy šablona má více parametrů, můžeme za některé parametry dosadit argumenty.

```
template<class T,unsigned n>
class Hash { ... };                    // šablona pro hašování
```

```
template<class T,unsigned n>          // specifické hašování
class HashA: public Hash<T,n> { };

class Zlomek { ... };

template<unsigned n>                                  // hašování zlomků
class HashZlomek: public Hash<Zlomek,n> { ... };

template<class T>                                     // velikost tabulky 100
class Hash100: public Hash<T,100> { ... };
```