Informatika 3

Šablóny
9

Šablóny (template)

Vzor pre:

- Popis množiny funkcii
- Popis množiny tried
- Skupina premenných – odlíšené typom

Explicitné vytvorenie inštancie šablóny

- inštancie šablón sa vytvárajú automaticky, keď sú potrebné
- Musíme dávať definíciu tela šablóny do hlavičkových súborov (pred volanie)
- Vytvorenie inštancie šablóny v module bez volania

```
template long max<long>(long, long);
```

Šablóny (template)

- Náhrada makier
- Inštancia šablóny

Šablóny - deklarácia

template <zoznam-formálnych-parametrov> deklarácia

- Hodnotové
- Typové
- Šablónové

Šablóny – typové parametre

```
template <typename T = double> class komplex;
```

alebo

```
template <class T = double> class komplex;
```

Šablóny – hodnotové parametre

- **Celočíselné typy**
- **Vymenovacie typy**
- **Smerník na objekt**
- **Odkaz objekt**
- **Smerník na funkciu**
- **Odkaz na funkciu**
- **Smerník na triedu**
- **std::nullptr_t**

- ako konštanty

```
template <auto hodnota> void f() {  … }
f<10>();
```

Šablóny – šablónové parametre

- **Iná šablóna objektového typu**

```
template <class S, template <class T> class R>
class kontajner
 {
      R<S> data;
 }
```

Inštancia šablóny - parameter inej šablóny

```
template <typename T = vector<int>> struct A
 {
  …
 }
```

Šablóna tried – triedna šablóna

- Môže byť použité na class, struct aj union

```
template<typename T>
class List{
    T *v;
    int size;
    static T nula;
public:
    List(int);
    T& operator[ ](int i) {return i>=0 && i<size ? v[ i ] : nula ;}
};

template < typename T> List<T>::List(int n)
{
    v = new T[n];
    size=n;
}

template< typename T> T List<T>::nula;

List<int> pole(10);
pole[-3] = 3;
int a = pole[100];        // čo bude v 'a' ?
int q = sizeof( pole );   // ??
```

- Bežne sa používa operátor priradenia, copy-konštr. – ak je to nutné je treba ich preťažiť

Šablónová trieda – inštancia šablóny

```
template<class T, int N>
class Fixed{
   T v[N];
   static T nula;
public:
   Fixed();
   T& operator[ ](int i) {return i>=0 && i<N ? v[ i ] : nula ;}
};
template<class T, int N> T Fixed<T,N>::nula;
template <class T, int N> Fixed<T,N>::Fixed() { }

Fixed<int,5> pole;
pole[100] = 10;
Int a = pole[-4];
int q = sizeof( pole );    // ??
```

Šablónová trieda – inštancia šablóny

template <class T, class R> struct A;

A<double, bool> p(3.8, false);

C++17

A p(3.8, false);

Funkčná šablóna

Namiesto písania viacerých funkcií

```
int max(int x, int y)                          {return x > y ? x : y;}
float max(float x, float y)                    {return x > y ? x : y;}
double max(double x, double y)                 {return x > y ? x : y;}
long max(long x, long y)                       {return x > y ? x : y;}
complex max(complex x, complex y)              {return x > y ? x : y;}
```

Príklad

- Všetky funkcie môžeme nahradiť šablónou

```
template <class T>
T max(T x, Ty)
{
   return (x > y) ? x : y;
}

main()
{
   int i=3, j=5;
   int k=max<int>(i, j);      // volanie šablony
   int k=max (i, j);          // nie je nutné písať max<int>
}
```

- Aby sme mohli zavolať šablónu:

```
complex p(3,5), q(10,20);
complex d = max(p, q);
```

- musí existovať operátor

```
bool operator>(const complex& x, const complex& y);
```

Príklad

```
template<class T>
void Pole(T* pole, int n)
{
   …
}
```

Priorita volania

- Ak nadefinujeme funkciu max s parametrami **double**:

```
double max(double x, double y) { return 55; }

main()
{
   int x=10, y=20;
   int k=max(x,y);  // volá sa šablóna (k=20)
   double p=30, q=40;
   double w=max(p,q); // volá sa funkcia (w=55)
}
```

- Pravidlo:
  - Ak existuje pre daný typ parametrov funkcia, zavolá táto funkcia
  - Ak neexistuje, prekladač zistí, či nemôže vytvoriť inštanciu (špecializáciu) šablóny.
  - Ak nie je k dispozicii ani funkcia, ani šablóna, prekladač vyhlási chybu

Šablóny aliasov

template <zoznam-parametrov> using identifkátor = označenie typu;

template<class T, int pocet> class A { … };

template<class T> using Aa = A<T,3>;

Aa<float> x;

Šablóny premenných

template <zoznam-parametrov> deklarácia-premennej;

```
// C++14
template<typename T>
// Presnosť je daná šablonovým parametrom.
constexpr T pi = T(3.14159265358979323846);


 // Použitie:
template<typename T>
T PlochaKruhu(T r) { return pi<T> * r * r; }

double p = PlochaKruhu<double>(10.3);
float p = PlochaKruhu<float>(10.3);
```

constexpr

```
// C++98
int f() { return 2; }
int arr[f()]; // ! chyba


// C++11
constexpr int f() { return 2; }
int arr[f()]; // OK
```